

0001 THE\_URL:file://localhost/Users/jehodges/documents/work/standards/W3C/webauthn/index-  
 0002 master-3c5e383.html  
 0003 THE\_TITLE:Web Authentication: An API for accessing Public Key Credentials - Level 1  
 0004 W3C  
 0005 Web Authentication:  
 0006 An API for accessing Public Key Credentials  
 0007 Level 1  
 0008  
 0009 Editor's Draft, 29 May 2018  
 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/2018/CR-webauthn-20180320/>  
 0019 <https://www.w3.org/TR/2018/WD-webauthn-20180315/>  
 0020 <https://www.w3.org/TR/2018/WD-webauthn-20180306/>  
 0021 <https://www.w3.org/TR/2017/WD-webauthn-20171205/>  
 0022 <https://www.w3.org/TR/2017/WD-webauthn-20170811/>  
 0023 <https://www.w3.org/TR/2017/WD-webauthn-20170505/>  
 0024 <https://www.w3.org/TR/2017/WD-webauthn-20170216/>  
 0025 <https://www.w3.org/TR/2016/WD-webauthn-20161207/>  
 0026 <https://www.w3.org/TR/2016/WD-webauthn-20160928/>  
 0027 <https://www.w3.org/TR/2016/WD-webauthn-20160902/>  
 0028 <https://www.w3.org/TR/2016/WD-webauthn-20160531/>  
 0029  
 0030 Issue Tracking:  
 0031 GitHub  
 0032  
 0033 Editors:  
 0034 Dirk Balfanz (Google)  
 0035 Alexei Czeskis (Google)  
 0036 Jeff Hodges (PayPal)  
 0037 J.C. Jones (Mozilla)  
 0038 Michael B. Jones (Microsoft)  
 0039 Akshay Kumar (Microsoft)  
 0040 Angelo Liao (Microsoft)  
 0041 Rolf Lindemann (Nok Nok Labs)  
 0042 Emil Lundberg (Yubico)  
 0043  
 0044 Former Editors:  
 0045 Vijay Bharadwaj (Microsoft)  
 0046 Arnar Birgisson (Google)  
 0047 Hubert Le Van Gong (PayPal)  
 0048  
 0049 Contributors:  
 0050 Christiaan Brand (Google)  
 0051 Adam Langley (Google)  
 0052 Giridhar Mandyam (Qualcomm)  
 0053 Mike West (Google)  
 0054 Jeffrey Yasskin (Google)  
 0055  
 0056 Tests:  
 0057 web-platform-tests webauthn/ (ongoing work)  
 0058  
 0059 Copyright 2018 W3C<sup>^</sup> (MIT, ERCIM, Keio, Beihang). W3C liability,  
 0060 trademark and document use rules apply.  
 0061  
 0062

0063 Abstract

0064  
 0065 This specification defines an API enabling the creation and use of  
 0066 strong, attested, scoped, public key-based credentials by web  
 0067 applications, for the purpose of strongly authenticating users.  
 0068 Conceptually, one or more public key credentials, each scoped to a  
 0069 given Relying Party, are created and stored on an authenticator by the

0001 THE\_URL:file://localhost/Users/jehodges/documents/work/standards/W3C/agl/webauthn/index-agl-  
 0002 issue905-0244f7c.html  
 0003 THE\_TITLE:Web Authentication: An API for accessing Public Key Credentials - Level 1  
 0004 W3C  
 0005 Web Authentication:  
 0006 An API for accessing Public Key Credentials  
 0007 Level 1  
 0008  
 0009 Editor's Draft, 29 May 2018  
 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/2018/CR-webauthn-20180320/>  
 0019 <https://www.w3.org/TR/2018/WD-webauthn-20180315/>  
 0020 <https://www.w3.org/TR/2018/WD-webauthn-20180306/>  
 0021 <https://www.w3.org/TR/2017/WD-webauthn-20171205/>  
 0022 <https://www.w3.org/TR/2017/WD-webauthn-20170811/>  
 0023 <https://www.w3.org/TR/2017/WD-webauthn-20170505/>  
 0024 <https://www.w3.org/TR/2017/WD-webauthn-20170216/>  
 0025 <https://www.w3.org/TR/2016/WD-webauthn-20161207/>  
 0026 <https://www.w3.org/TR/2016/WD-webauthn-20160928/>  
 0027 <https://www.w3.org/TR/2016/WD-webauthn-20160902/>  
 0028 <https://www.w3.org/TR/2016/WD-webauthn-20160531/>  
 0029  
 0030 Issue Tracking:  
 0031 GitHub  
 0032  
 0033 Editors:  
 0034 Dirk Balfanz (Google)  
 0035 Alexei Czeskis (Google)  
 0036 Jeff Hodges (PayPal)  
 0037 J.C. Jones (Mozilla)  
 0038 Michael B. Jones (Microsoft)  
 0039 Akshay Kumar (Microsoft)  
 0040 Angelo Liao (Microsoft)  
 0041 Rolf Lindemann (Nok Nok Labs)  
 0042 Emil Lundberg (Yubico)  
 0043  
 0044 Former Editors:  
 0045 Vijay Bharadwaj (Microsoft)  
 0046 Arnar Birgisson (Google)  
 0047 Hubert Le Van Gong (PayPal)  
 0048  
 0049 Contributors:  
 0050 Christiaan Brand (Google)  
 0051 Adam Langley (Google)  
 0052 Giridhar Mandyam (Qualcomm)  
 0053 Mike West (Google)  
 0054 Jeffrey Yasskin (Google)  
 0055  
 0056 Tests:  
 0057 web-platform-tests webauthn/ (ongoing work)  
 0058  
 0059 Copyright 2018 W3C<sup>^</sup> (MIT, ERCIM, Keio, Beihang). W3C liability,  
 0060 trademark and document use rules apply.  
 0061  
 0062

0063 Abstract

0064  
 0065 This specification defines an API enabling the creation and use of  
 0066 strong, attested, scoped, public key-based credentials by web  
 0067 applications, for the purpose of strongly authenticating users.  
 0068 Conceptually, one or more public key credentials, each scoped to a  
 0069 given Relying Party, are created and stored on an authenticator by the

0070 user agent in conjunction with the web application. The user agent  
0071 mediates access to public key credentials in order to preserve user  
0072 privacy. Authenticators are responsible for ensuring that no operation  
0073 is performed without user consent. Authenticators provide cryptographic  
0074 proof of their properties to relying parties via attestation. This  
0075 specification also describes the functional model for WebAuthn  
0076 conformant authenticators, including their signature and attestation  
0077 functionality.

0078 Status of this document

0080 This section describes the status of this document at the time of its  
0081 publication. Other documents may supersede this document. A list of  
0082 current W3C publications and the latest revision of this technical  
0083 report can be found in the W3C technical reports index at  
0084 <http://www.w3.org/TR/>.

0086 This document was published by the Web Authentication Working Group as  
0087 an Editors' Draft. This document is intended to become a W3C  
0088 Recommendation. Feedback and comments on this specification are  
0089 welcome. Please use Github issues. Discussions may also be found in the  
0090 public-webauthn@w3.org archives.

0092 Publication as an Editors' Draft does not imply endorsement by the W3C  
0093 Membership. This is a draft document and may be updated, replaced or  
0094 obsolete by other documents at any time. It is inappropriate to cite  
0095 this document as other than work in progress.

0097 This document was produced by a group operating under the W3C Patent  
0098 Policy. W3C maintains a public list of any patent disclosures made in  
0099 connection with the deliverables of the group; that page also includes  
0100 instructions for disclosing a patent. An individual who has actual  
0101 knowledge of a patent which the individual believes contains Essential  
0102 Claim(s) must disclose the information in accordance with section 6 of  
0103 the W3C Patent Policy.

0105 This document is governed by the 1 February 2018 W3C Process Document.

0107 Table of Contents

0108 1. 1 Introduction  
0109 1. 1.1 Use Cases  
0110 1. 1.1.1 Registration  
0111 2. 1.1.2 Authentication  
0112 3. 1.1.3 Other use cases and configurations  
0113 2. 2 Conformance  
0114 1. 2.1 User Agents  
0115 2. 2.2 Authenticators  
0116 1. 2.2.1 Backwards Compatibility with FIDO U2F  
0117 3. 2.3 Relying Parties  
0118 4. 2.4 All Conformance Classes  
0119 3. 3 Dependencies  
0120 4. 4 Terminology  
0121 5. 5 Web Authentication API  
0122 1. 5.1 PublicKeyCredential Interface  
0123 1. 5.1.1 CredentialCreationOptions Dictionary Extension  
0124 2. 5.1.2 CredentialRequestOptions Dictionary Extension  
0125 3. 5.1.3 Create a new credential - PublicKeyCredential's  
0126 `[[Create]](origin, options, sameOriginWithAncestors)`  
0127 method  
0128 4. 5.1.4 Use an existing credential to make an assertion -  
0129 `PublicKeyCredential's [[Get]](options) method`  
0130 1. 5.1.4.1 `PublicKeyCredential's`  
0131 `[[DiscoverFromExternalSource]](origin, options,`  
0132 `sameOriginWithAncestors) method`  
0133 5. 5.1.5 Store an existing credential -  
0134 `PublicKeyCredential's [[Store]](credential,`  
0135 `sameOriginWithAncestors) method`  
0136 6. 5.1.6 Preventing silent access to an existing credential  
0137 - `PublicKeyCredential's`

0070 user agent in conjunction with the web application. The user agent  
0071 mediates access to public key credentials in order to preserve user  
0072 privacy. Authenticators are responsible for ensuring that no operation  
0073 is performed without user consent. Authenticators provide cryptographic  
0074 proof of their properties to relying parties via attestation. This  
0075 specification also describes the functional model for WebAuthn  
0076 conformant authenticators, including their signature and attestation  
0077 functionality.

0078 Status of this document

0080 This section describes the status of this document at the time of its  
0081 publication. Other documents may supersede this document. A list of  
0082 current W3C publications and the latest revision of this technical  
0083 report can be found in the W3C technical reports index at  
0084 <http://www.w3.org/TR/>.

0086 This document was published by the Web Authentication Working Group as  
0087 an Editors' Draft. This document is intended to become a W3C  
0088 Recommendation. Feedback and comments on this specification are  
0089 welcome. Please use Github issues. Discussions may also be found in the  
0090 public-webauthn@w3.org archives.

0092 Publication as an Editors' Draft does not imply endorsement by the W3C  
0093 Membership. This is a draft document and may be updated, replaced or  
0094 obsolete by other documents at any time. It is inappropriate to cite  
0095 this document as other than work in progress.

0097 This document was produced by a group operating under the W3C Patent  
0098 Policy. W3C maintains a public list of any patent disclosures made in  
0099 connection with the deliverables of the group; that page also includes  
0100 instructions for disclosing a patent. An individual who has actual  
0101 knowledge of a patent which the individual believes contains Essential  
0102 Claim(s) must disclose the information in accordance with section 6 of  
0103 the W3C Patent Policy.

0105 This document is governed by the 1 February 2018 W3C Process Document.

0107 Table of Contents

0108 1. 1 Introduction  
0109 1. 1.1 Use Cases  
0110 1. 1.1.1 Registration  
0111 2. 1.1.2 Authentication  
0112 3. 1.1.3 Other use cases and configurations  
0113 2. 2 Conformance  
0114 1. 2.1 User Agents  
0115 2. 2.2 Authenticators  
0116 1. 2.2.1 Backwards Compatibility with FIDO U2F  
0117 3. 2.3 Relying Parties  
0118 4. 2.4 All Conformance Classes  
0119 3. 3 Dependencies  
0120 4. 4 Terminology  
0121 5. 5 Web Authentication API  
0122 1. 5.1 PublicKeyCredential Interface  
0123 1. 5.1.1 CredentialCreationOptions Dictionary Extension  
0124 2. 5.1.2 CredentialRequestOptions Dictionary Extension  
0125 3. 5.1.3 Create a new credential - PublicKeyCredential's  
0126 `[[Create]](origin, options, sameOriginWithAncestors)`  
0127 method  
0128 4. 5.1.4 Use an existing credential to make an assertion -  
0129 `PublicKeyCredential's [[Get]](options) method`  
0130 1. 5.1.4.1 `PublicKeyCredential's`  
0131 `[[DiscoverFromExternalSource]](origin, options,`  
0132 `sameOriginWithAncestors) method`  
0133 5. 5.1.5 Store an existing credential -  
0134 `PublicKeyCredential's [[Store]](credential,`  
0135 `sameOriginWithAncestors) method`  
0136 6. 5.1.6 Preventing silent access to an existing credential  
0137 - `PublicKeyCredential's`

```

014C [[preventSilentAccess]](credential,
0141 sameOriginWithAncestors) method
0142 7. 5.1.7 Availability of User-Verifying Platform
0143 Authenticator - PublicKeyCredential's
0144 isUserVerifyingPlatformAuthenticatorAvailable() method
0145 2. 5.2 Authenticator Responses (interface AuthenticatorResponse)
0146 1. 5.2.1 Information about Public Key Credential (interface
0147 AuthenticatorAttestationResponse)
0148 2. 5.2.2 Web Authentication Assertion (interface
0149 AuthenticatorAssertionResponse)
0150 3. 5.3 Parameters for Credential Generation (dictionary
0151 PublicKeyCredentialParameters)
0152 4. 5.4 Options for Credential Creation (dictionary
0153 PublicKeyCredentialCreationOptions)
0154 1. 5.4.1 Public Key Entity Description (dictionary
0155 PublicKeyCredentialEntity)
0156 2. 5.4.2 RP Parameters for Credential Generation (dictionary
0157 PublicKeyCredentialRpEntity)
0158 3. 5.4.3 User Account Parameters for Credential Generation
0159 (dictionary PublicKeyCredentialUserEntity)
0160 4. 5.4.4 Authenticator Selection Criteria (dictionary
0161 AuthenticatorSelectionCriteria)
0162 5. 5.4.5 Authenticator Attachment enumeration (enum
0163 AuthenticatorAttachment)
0164 6. 5.4.6 Attestation Conveyance Preference enumeration (enum
0165 AttestationConveyancePreference)
0166 5. 5.5 Options for Assertion Generation (dictionary
0167 PublicKeyCredentialRequestOptions)
0168 6. 5.6 Abort operations with AbortSignal
0169 7. 5.7 Authentication Extensions Client Inputs (typedef
0170 AuthenticationExtensionsClientInputs)
0171 8. 5.8 Authentication Extensions Client Outputs (typedef
0172 AuthenticationExtensionsClientOutputs)
0173 9. 5.9 Authentication Extensions Authenticator Inputs (typedef
0174 AuthenticationExtensionsAuthenticatorInputs)
0175 10. 5.10 Supporting Data Structures
0176 1. 5.10.1 Client data used in WebAuthn signatures
0177 (dictionary CollectedClientData)
0178 2. 5.10.2 Credential Type enumeration (enum
0179 PublicKeyCredentialType)
0180 3. 5.10.3 Credential Descriptor (dictionary
0181 PublicKeyCredentialDescriptor)
0182 4. 5.10.4 Authenticator Transport enumeration (enum
0183 AuthenticatorTransport)
0184 5. 5.10.5 Cryptographic Algorithm Identifier (typedef
0185 COSEAlgorithmIdentifier)
0186 6. 5.10.6 User Verification Requirement enumeration (enum
0187 UserVerificationRequirement)
0188 6. 6 WebAuthn Authenticator Model
0189 1. 6.1 Authenticator data
0190 1. 6.1.1 Signature Counter Considerations
0191 2. 6.1.2 FIDO U2F signature format compatibility
0192 2. 6.2 Authenticator operations
0193 1. 6.2.1 Lookup Credential Source by Credential ID algorithm
0194 2. 6.2.2 The authenticatorMakeCredential operation
0195 3. 6.2.3 The authenticatorGetAssertion operation
0196 4. 6.2.4 The authenticatorCancel operation
0197 3. 6.3 Attestation
0198 1. 6.3.1 Attested credential data
0199 1. 6.3.1.1 Examples of credentialPublicKey Values
0200 encoded in COSE_Key format
0201 2. 6.3.2 Attestation Statement Formats
0202 3. 6.3.3 Attestation Types
0203 4. 6.3.4 Generating an Attestation Object
0204 5. 6.3.5 Signature Formats for Packed Attestation, FIDO U2F
0205 Attestation, and Assertion Signatures
0206 7. 7 Relying Party Operations
0207 1. 7.1 Registering a new credential
0208 2. 7.2 Verifying an authentication assertion
0209 8. 8 Defined Attestation Statement Formats

```

```

014C [[preventSilentAccess]](credential,
0141 sameOriginWithAncestors) method
0142 7. 5.1.7 Availability of User-Verifying Platform
0143 Authenticator - PublicKeyCredential's
0144 isUserVerifyingPlatformAuthenticatorAvailable() method
0145 2. 5.2 Authenticator Responses (interface AuthenticatorResponse)
0146 1. 5.2.1 Information about Public Key Credential (interface
0147 AuthenticatorAttestationResponse)
0148 2. 5.2.2 Web Authentication Assertion (interface
0149 AuthenticatorAssertionResponse)
0150 3. 5.3 Parameters for Credential Generation (dictionary
0151 PublicKeyCredentialParameters)
0152 4. 5.4 Options for Credential Creation (dictionary
0153 PublicKeyCredentialCreationOptions)
0154 1. 5.4.1 Public Key Entity Description (dictionary
0155 PublicKeyCredentialEntity)
0156 2. 5.4.2 RP Parameters for Credential Generation (dictionary
0157 PublicKeyCredentialRpEntity)
0158 3. 5.4.3 User Account Parameters for Credential Generation
0159 (dictionary PublicKeyCredentialUserEntity)
0160 4. 5.4.4 Authenticator Selection Criteria (dictionary
0161 AuthenticatorSelectionCriteria)
0162 5. 5.4.5 Authenticator Attachment enumeration (enum
0163 AuthenticatorAttachment)
0164 6. 5.4.6 Attestation Conveyance Preference enumeration (enum
0165 AttestationConveyancePreference)
0166 5. 5.5 Options for Assertion Generation (dictionary
0167 PublicKeyCredentialRequestOptions)
0168 6. 5.6 Abort operations with AbortSignal
0169 7. 5.7 Authentication Extensions Client Inputs (typedef
0170 AuthenticationExtensionsClientInputs)
0171 8. 5.8 Authentication Extensions Client Outputs (typedef
0172 AuthenticationExtensionsClientOutputs)
0173 9. 5.9 Authentication Extensions Authenticator Inputs (typedef
0174 AuthenticationExtensionsAuthenticatorInputs)
0175 10. 5.10 Supporting Data Structures
0176 1. 5.10.1 Client data used in WebAuthn signatures
0177 (dictionary CollectedClientData)
0178 2. 5.10.2 Credential Type enumeration (enum
0179 PublicKeyCredentialType)
0180 3. 5.10.3 Credential Descriptor (dictionary
0181 PublicKeyCredentialDescriptor)
0182 4. 5.10.4 Authenticator Transport enumeration (enum
0183 AuthenticatorTransport)
0184 5. 5.10.5 Cryptographic Algorithm Identifier (typedef
0185 COSEAlgorithmIdentifier)
0186 6. 5.10.6 User Verification Requirement enumeration (enum
0187 UserVerificationRequirement)
0188 6. 6 WebAuthn Authenticator Model
0189 1. 6.1 Authenticator data
0190 1. 6.1.1 Signature Counter Considerations
0191 2. 6.1.2 FIDO U2F signature format compatibility
0192 2. 6.2 Authenticator operations
0193 1. 6.2.1 Lookup Credential Source by Credential ID algorithm
0194 2. 6.2.2 The authenticatorMakeCredential operation
0195 3. 6.2.3 The authenticatorGetAssertion operation
0196 4. 6.2.4 The authenticatorCancel operation
0197 3. 6.3 Attestation
0198 1. 6.3.1 Attested credential data
0199 1. 6.3.1.1 Examples of credentialPublicKey Values
0200 encoded in COSE_Key format
0201 2. 6.3.2 Attestation Statement Formats
0202 3. 6.3.3 Attestation Types
0203 4. 6.3.4 Generating an Attestation Object
0204 5. 6.3.5 Signature Formats for Packed Attestation, FIDO U2F
0205 Attestation, and Assertion Signatures
0206 7. 7 Relying Party Operations
0207 1. 7.1 Registering a new credential
0208 2. 7.2 Verifying an authentication assertion
0209 8. 8 Defined Attestation Statement Formats

```

- 0210 1. 8.1 Attestation Statement Format Identifiers
- 0211 2. 8.2 Packed Attestation Statement Format
- 0212 1. 8.2.1 Packed attestation statement certificate requirements
- 0213 3. 8.3 TPM Attestation Statement Format
- 0214 1. 8.3.1 TPM attestation statement certificate requirements
- 0215 4. 8.4 Android Key Attestation Statement Format
- 0216 5. 8.5 Android SafetyNet Attestation Statement Format
- 0217 6. 8.6 FIDO U2F Attestation Statement Format
- 0218 7. 8.7 None Attestation Statement Format
- 0219 9. 9 WebAuthn Extensions
- 0220 1. 9.1 Extension Identifiers
- 0221 2. 9.2 Defining extensions
- 0222 3. 9.3 Extending request parameters
- 0223 4. 9.4 Client extension processing
- 0224 5. 9.5 Authenticator extension processing
- 0225 10. 10 Defined Extensions
- 0226 1. 10.1 FIDO AppID Extension (appid)
- 0227 2. 10.2 Simple Transaction Authorization Extension (txAuthSimple)
- 0228 3. 10.3 Generic Transaction Authorization Extension (txAuthGeneric)
- 0229 4. 10.4 Authenticator Selection Extension (authnSel)
- 0230 5. 10.5 Supported Extensions Extension (exts)
- 0231 6. 10.6 User Verification Index Extension (uvi)
- 0232 7. 10.7 Location Extension (loc)
- 0233 8. 10.8 User Verification Method Extension (uvm)
- 0234 9. 10.9 Biometric Authenticator Performance Bounds Extension (biometricPerfBounds)
- 0235 11. 11 IANA Considerations
- 0236 1. 11.1 WebAuthn Attestation Statement Format Identifier Registrations
- 0237 2. 11.2 WebAuthn Extension Identifier Registrations
- 0238 3. 11.3 COSE Algorithm Registrations
- 0239 12. 12 Sample scenarios
- 0240 1. 12.1 Registration
- 0241 2. 12.2 Registration Specifically with User Verifying Platform Authenticator
- 0242 3. 12.3 Authentication
- 0243 4. 12.4 Aborting Authentication Operations
- 0244 5. 12.5 Decommissioning
- 0245 13. 13 Security Considerations
- 0246 1. 13.1 Cryptographic Challenges
- 0247 2. 13.2 Attestation Security Considerations
- 0248 1. 13.2.1 Attestation Certificate Hierarchy
- 0249 2. 13.2.2 Attestation Certificate and Attestation Certificate CA Compromise
- 0250 3. 13.3 Security Benefits for Relying Parties
- 0251 1. 13.3.1 Considerations for Self and None Attestation Types and Ignoring Attestation
- 0252 4. 13.4 credentialId Unsigned
- 0253 5. 13.5 Browser Permissions Framework and Extensions
- 0254 14. 14 Privacy Considerations
- 0255 1. 14.1 Attestation Privacy
- 0256 2. 14.2 Registration Ceremony Privacy
- 0257 3. 14.3 Authentication Ceremony Privacy
- 0258 15. 15 Acknowledgements
- 0259 16. Index
- 0260 1. Terms defined by this specification
- 0261 2. Terms defined by reference
- 0262 17. References
- 0263 1. Normative References
- 0264 2. Informative References
- 0265 18. IDL Index
- 0266 19. Issues Index
- 0267 1. Introduction
- 0268 This section is not normative.
- 0269 This specification defines an API enabling the creation and use of

- 0210 1. 8.1 Attestation Statement Format Identifiers
- 0211 2. 8.2 Packed Attestation Statement Format
- 0212 1. 8.2.1 Packed attestation statement certificate requirements
- 0213 3. 8.3 TPM Attestation Statement Format
- 0214 1. 8.3.1 TPM attestation statement certificate requirements
- 0215 4. 8.4 Android Key Attestation Statement Format
- 0216 5. 8.5 Android SafetyNet Attestation Statement Format
- 0217 6. 8.6 FIDO U2F Attestation Statement Format
- 0218 7. 8.7 None Attestation Statement Format
- 0219 9. 9 WebAuthn Extensions
- 0220 1. 9.1 Extension Identifiers
- 0221 2. 9.2 Defining extensions
- 0222 3. 9.3 Extending request parameters
- 0223 4. 9.4 Client extension processing
- 0224 5. 9.5 Authenticator extension processing
- 0225 10. 10 Defined Extensions
- 0226 1. 10.1 FIDO AppID Extension (appid)
- 0227 2. 10.2 Simple Transaction Authorization Extension (txAuthSimple)
- 0228 3. 10.3 Generic Transaction Authorization Extension (txAuthGeneric)
- 0229 4. 10.4 Authenticator Selection Extension (authnSel)
- 0230 5. 10.5 Supported Extensions Extension (exts)
- 0231 6. 10.6 User Verification Index Extension (uvi)
- 0232 7. 10.7 Location Extension (loc)
- 0233 8. 10.8 User Verification Method Extension (uvm)
- 0234 9. 10.9 Biometric Authenticator Performance Bounds Extension (biometricPerfBounds)
- 0235 11. 11 IANA Considerations
- 0236 1. 11.1 WebAuthn Attestation Statement Format Identifier Registrations
- 0237 2. 11.2 WebAuthn Extension Identifier Registrations
- 0238 3. 11.3 COSE Algorithm Registrations
- 0239 12. 12 Sample scenarios
- 0240 1. 12.1 Registration
- 0241 2. 12.2 Registration Specifically with User Verifying Platform Authenticator
- 0242 3. 12.3 Authentication
- 0243 4. 12.4 Aborting Authentication Operations
- 0244 5. 12.5 Decommissioning
- 0245 13. 13 Security Considerations
- 0246 1. 13.1 Cryptographic Challenges
- 0247 2. 13.2 Attestation Security Considerations
- 0248 1. 13.2.1 Attestation Certificate Hierarchy
- 0249 2. 13.2.2 Attestation Certificate and Attestation Certificate CA Compromise
- 0250 3. 13.3 Security Benefits for Relying Parties
- 0251 1. 13.3.1 Considerations for Self and None Attestation Types and Ignoring Attestation
- 0252 4. 13.4 credentialId Unsigned
- 0253 5. 13.5 Browser Permissions Framework and Extensions
- 0254 14. 14 Privacy Considerations
- 0255 1. 14.1 Attestation Privacy
- 0256 2. 14.2 Registration Ceremony Privacy
- 0257 3. 14.3 Authentication Ceremony Privacy
- 0258 15. 15 Acknowledgements
- 0259 16. Index
- 0260 1. Terms defined by this specification
- 0261 2. Terms defined by reference
- 0262 17. References
- 0263 1. Normative References
- 0264 2. Informative References
- 0265 18. IDL Index
- 0266 19. Issues Index
- 0267 1. Introduction
- 0268 This section is not normative.
- 0269 This specification defines an API enabling the creation and use of

0280 strong, attested, scoped, public key-based credentials by web  
0281 applications, for the purpose of strongly authenticating users. A  
0282 public key credential is created and stored by an authenticator at the  
0283 behest of a Relying Party, subject to user consent. Subsequently, the  
0284 public key credential can only be accessed by origins belonging to that  
0285 Relying Party. This scoping is enforced jointly by conforming User  
0286 Agents and authenticators. Additionally, privacy across Relying Parties  
0287 is maintained; Relying Parties are not able to detect any properties,  
0288 or even the existence, of credentials scoped to other Relying Parties.

0289  
0290 Relying Parties employ the Web Authentication API during two distinct,  
0291 but related, ceremonies involving a user. The first is Registration,  
0292 where a public key credential is created on an authenticator, and  
0293 associated by a Relying Party with the present user's account (the  
0294 account MAY already exist or MAY be created at this time). The second  
0295 is Authentication, where the Relying Party is presented with an  
0296 Authentication Assertion proving the presence and consent of the user  
0297 who registered the public key credential. Functionally, the Web  
0298 Authentication API comprises a PublicKeyCredential which extends the  
0299 Credential Management API [CREDENTIAL-MANAGEMENT-1], and infrastructure  
0300 which allows those credentials to be used with  
0301 navigator.credentials.create() and navigator.credentials.get(). The  
0302 former is used during Registration, and the latter during  
0303 Authentication.

0304  
0305 Broadly, compliant authenticators protect public key credentials, and  
0306 interact with user agents to implement the Web Authentication API. Some  
0307 authenticators MAY run on the same computing device (e.g., smart phone,  
0308 tablet, desktop PC) as the user agent is running on. For instance, such  
0309 an authenticator might consist of a Trusted Execution Environment (TEE)  
0310 applet, a Trusted Platform Module (TPM), or a Secure Element (SE)  
0311 integrated into the computing device in conjunction with some means for  
0312 user verification, along with appropriate platform software to mediate  
0313 access to these components' functionality. Other authenticators MAY  
0314 operate autonomously from the computing device running the user agent,  
0315 and be accessed over a transport such as Universal Serial Bus (USB),  
0316 Bluetooth Low Energy (BLE) or Near Field Communications (NFC).

0317  
0318 1.1. Use Cases

0319  
0320 The below use case scenarios illustrate use of two very different types  
0321 of authenticators, as well as outline further scenarios. Additional  
0322 scenarios, including sample code, are given later in 12 Sample  
0323 scenarios.

0324  
0325 1.1.1. Registration

0326 \* On a phone:

- 0327 + User navigates to example.com in a browser and signs in to an  
0328 existing account using whatever method they have been using  
0329 (possibly a legacy method such as a password), or creates a  
0330 new account.
- 0331 + The phone prompts, "Do you want to register this device with  
0332 example.com?"
- 0333 + User agrees.
- 0334 + The phone prompts the user for a previously configured  
0335 authorization gesture (PIN, biometric, etc.); the user  
0336 provides this.
- 0337 + Website shows message, "Registration complete."

0338  
0339 1.1.2. Authentication

0340 \* On a laptop or desktop:

- 0341 + User pairs their phone with the laptop or desktop via  
0342 Bluetooth.
- 0343 + User navigates to example.com in a browser and initiates  
0344 signing in.
- 0345 + User gets a message from the browser, "Please complete this  
0346 action on your phone."

0347 \* Next, on their phone:

0348  
0349

0280 strong, attested, scoped, public key-based credentials by web  
0281 applications, for the purpose of strongly authenticating users. A  
0282 public key credential is created and stored by an authenticator at the  
0283 behest of a Relying Party, subject to user consent. Subsequently, the  
0284 public key credential can only be accessed by origins belonging to that  
0285 Relying Party. This scoping is enforced jointly by conforming User  
0286 Agents and authenticators. Additionally, privacy across Relying Parties  
0287 is maintained; Relying Parties are not able to detect any properties,  
0288 or even the existence, of credentials scoped to other Relying Parties.

0289  
0290 Relying Parties employ the Web Authentication API during two distinct,  
0291 but related, ceremonies involving a user. The first is Registration,  
0292 where a public key credential is created on an authenticator, and  
0293 associated by a Relying Party with the present user's account (the  
0294 account MAY already exist or MAY be created at this time). The second  
0295 is Authentication, where the Relying Party is presented with an  
0296 Authentication Assertion proving the presence and consent of the user  
0297 who registered the public key credential. Functionally, the Web  
0298 Authentication API comprises a PublicKeyCredential which extends the  
0299 Credential Management API [CREDENTIAL-MANAGEMENT-1], and infrastructure  
0300 which allows those credentials to be used with  
0301 navigator.credentials.create() and navigator.credentials.get(). The  
0302 former is used during Registration, and the latter during  
0303 Authentication.

0304  
0305 Broadly, compliant authenticators protect public key credentials, and  
0306 interact with user agents to implement the Web Authentication API. Some  
0307 authenticators MAY run on the same computing device (e.g., smart phone,  
0308 tablet, desktop PC) as the user agent is running on. For instance, such  
0309 an authenticator might consist of a Trusted Execution Environment (TEE)  
0310 applet, a Trusted Platform Module (TPM), or a Secure Element (SE)  
0311 integrated into the computing device in conjunction with some means for  
0312 user verification, along with appropriate platform software to mediate  
0313 access to these components' functionality. Other authenticators MAY  
0314 operate autonomously from the computing device running the user agent,  
0315 and be accessed over a transport such as Universal Serial Bus (USB),  
0316 Bluetooth Low Energy (BLE) or Near Field Communications (NFC).

0317  
0318 1.1. Use Cases

0319  
0320 The below use case scenarios illustrate use of two very different types  
0321 of authenticators, as well as outline further scenarios. Additional  
0322 scenarios, including sample code, are given later in 12 Sample  
0323 scenarios.

0324  
0325 1.1.1. Registration

0326 \* On a phone:

- 0327 + User navigates to example.com in a browser and signs in to an  
0328 existing account using whatever method they have been using  
0329 (possibly a legacy method such as a password), or creates a  
0330 new account.
- 0331 + The phone prompts, "Do you want to register this device with  
0332 example.com?"
- 0333 + User agrees.
- 0334 + The phone prompts the user for a previously configured  
0335 authorization gesture (PIN, biometric, etc.); the user  
0336 provides this.
- 0337 + Website shows message, "Registration complete."

0338  
0339 1.1.2. Authentication

0340 \* On a laptop or desktop:

- 0341 + User pairs their phone with the laptop or desktop via  
0342 Bluetooth.
- 0343 + User navigates to example.com in a browser and initiates  
0344 signing in.
- 0345 + User gets a message from the browser, "Please complete this  
0346 action on your phone."

0347 \* Next, on their phone:

0348  
0349

0350 + User sees a discrete prompt or notification, "Sign in to  
0351 example.com."  
0352 + User selects this prompt / notification.  
0353 + User is shown a list of their example.com identities, e.g.,  
0354 "Sign in as Alice / Sign in as Bob."  
0355 + User picks an identity, is prompted for an authorization  
0356 gesture (PIN, biometric, etc.) and provides this.  
0357 \* Now, back on the laptop:  
0358 + Web page shows that the selected user is signed in, and  
0359 navigates to the signed-in page.  
0360  
0361 1.1.3. Other use cases and configurations  
0362  
0363 A variety of additional use cases and configurations are also possible,  
0364 including (but not limited to):  
0365 \* A user navigates to example.com on their laptop, is guided through  
0366 a flow to create and register a credential on their phone.  
0367 \* A user obtains a discrete, roaming authenticator, such as a "fob"  
0368 with USB or USB+NFC/BLE connectivity options, loads example.com in  
0369 their browser on a laptop or phone, and is guided through a flow to  
0370 create and register a credential on the fob.  
0371 \* A Relying Party prompts the user for their authorization gesture in  
0372 order to authorize a single transaction, such as a payment or other  
0373 financial transaction.  
0374  
0375 2. Conformance  
0376  
0377 This specification defines three conformance classes. Each of these  
0378 classes is specified so that conforming members of the class are secure  
0379 against non-conforming or hostile members of the other classes.  
0380  
0381 2.1. User Agents  
0382  
0383 A User Agent MUST behave as described by 5 Web Authentication API in  
0384 order to be considered conformant. Conforming User Agents MAY implement  
0385 algorithms given in this specification in any way desired, so long as  
0386 the end result is indistinguishable from the result that would be  
0387 obtained by the specification's algorithms.  
0388  
0389 A conforming User Agent MUST also be a conforming implementation of the  
0390 IDL fragments of this specification, as described in the "Web IDL"  
0391 specification. [WebIDL-1]  
0392  
0393 2.2. Authenticators  
0394  
0395 An authenticator MUST provide the operations defined by 6 WebAuthn  
0396 Authenticator Model, and those operations MUST behave as described  
0397 there. This is a set of functional and security requirements for an  
0398 authenticator to be usable by a Conforming User Agent.  
0399  
0400 As described in 1.1 Use Cases, an authenticator may be implemented in  
0401 the operating system underlying the User Agent, or in external  
0402 hardware, or a combination of both.  
0403  
0404 2.2.1. Backwards Compatibility with FIDO U2F  
0405  
0406 Authenticators that only support the 8.6 FIDO U2F Attestation  
0407 Statement Format have no mechanism to store a user handle, so the  
0408 returned userHandle will always be null.  
0409  
0410 2.3. Relying Parties  
0411  
0412 A Relying Party MUST behave as described in 7 Relying Party Operations  
0413 to obtain all the security benefits offered by this specification. See  
0414 13.3 Security Benefits for Relying Parties for further discussion of  
0415 this.  
0416  
0417 2.4. All Conformance Classes  
0418  
0419 All CBOR encoding performed by the members of the above conformance  
0420

0350 + User sees a discrete prompt or notification, "Sign in to  
0351 example.com."  
0352 + User selects this prompt / notification.  
0353 + User is shown a list of their example.com identities, e.g.,  
0354 "Sign in as Alice / Sign in as Bob."  
0355 + User picks an identity, is prompted for an authorization  
0356 gesture (PIN, biometric, etc.) and provides this.  
0357 \* Now, back on the laptop:  
0358 + Web page shows that the selected user is signed in, and  
0359 navigates to the signed-in page.  
0360  
0361 1.1.3. Other use cases and configurations  
0362  
0363 A variety of additional use cases and configurations are also possible,  
0364 including (but not limited to):  
0365 \* A user navigates to example.com on their laptop, is guided through  
0366 a flow to create and register a credential on their phone.  
0367 \* A user obtains a discrete, roaming authenticator, such as a "fob"  
0368 with USB or USB+NFC/BLE connectivity options, loads example.com in  
0369 their browser on a laptop or phone, and is guided through a flow to  
0370 create and register a credential on the fob.  
0371 \* A Relying Party prompts the user for their authorization gesture in  
0372 order to authorize a single transaction, such as a payment or other  
0373 financial transaction.  
0374  
0375 2. Conformance  
0376  
0377 This specification defines three conformance classes. Each of these  
0378 classes is specified so that conforming members of the class are secure  
0379 against non-conforming or hostile members of the other classes.  
0380  
0381 2.1. User Agents  
0382  
0383 A User Agent MUST behave as described by 5 Web Authentication API in  
0384 order to be considered conformant. Conforming User Agents MAY implement  
0385 algorithms given in this specification in any way desired, so long as  
0386 the end result is indistinguishable from the result that would be  
0387 obtained by the specification's algorithms.  
0388  
0389 A conforming User Agent MUST also be a conforming implementation of the  
0390 IDL fragments of this specification, as described in the "Web IDL"  
0391 specification. [WebIDL-1]  
0392  
0393 2.2. Authenticators  
0394  
0395 An authenticator MUST provide the operations defined by 6 WebAuthn  
0396 Authenticator Model, and those operations MUST behave as described  
0397 there. This is a set of functional and security requirements for an  
0398 authenticator to be usable by a Conforming User Agent.  
0399  
0400 As described in 1.1 Use Cases, an authenticator may be implemented in  
0401 the operating system underlying the User Agent, or in external  
0402 hardware, or a combination of both.  
0403  
0404 2.2.1. Backwards Compatibility with FIDO U2F  
0405  
0406 Authenticators that only support the 8.6 FIDO U2F Attestation  
0407 Statement Format have no mechanism to store a user handle, so the  
0408 returned userHandle will always be null.  
0409  
0410 2.3. Relying Parties  
0411  
0412 A Relying Party MUST behave as described in 7 Relying Party Operations  
0413 to obtain all the security benefits offered by this specification. See  
0414 13.3 Security Benefits for Relying Parties for further discussion of  
0415 this.  
0416  
0417 2.4. All Conformance Classes  
0418  
0419 All CBOR encoding performed by the members of the above conformance  
0420

0420 classes MUST be done using the CTAP2 canonical CBOR encoding form. All  
0421 decoders of the above conformance classes SHOULD reject CBOR that is  
0422 not validly encoded in the CTAP2 canonical CBOR encoding form and  
0423 SHOULD reject messages with duplicate map keys.  
0424

0425 **3. Dependencies**  
0426

0427 This specification relies on several other underlying specifications,  
0428 listed below and in Terms defined by reference.  
0429

0430 **Base64url encoding**  
0431 The term Base64url Encoding refers to the base64 encoding using  
0432 the URL- and filename-safe character set defined in Section 5 of  
0433 [RFC4648], with all trailing '=' characters omitted (as  
0434 permitted by Section 3.2) and without the inclusion of any line  
0435 breaks, whitespace, or other additional characters.  
0436

0437 **CBOR**  
0438 A number of structures in this specification, including  
0439 attestation statements and extensions, are encoded using the  
0440 CTAP2 canonical CBOR encoding form of the Compact Binary Object  
0441 Representation (CBOR) [RFC7049], as defined in [FIDO-CTAP].  
0442

0443 **CDDL**  
0444 This specification describes the syntax of all CBOR-encoded data  
0445 using the CBOR Data Definition Language (CDDL) [CDDL].  
0446

0447 **COSE**  
0448 CBOR Object Signing and Encryption (COSE) [RFC8152]. The IANA  
0449 COSE Algorithms registry established by this specification is  
0450 also used.  
0451

0452 **Credential Management**  
0453 The API described in this document is an extension of the  
0454 Credential concept defined in [CREDENTIAL-MANAGEMENT-1].  
0455

0456 **DOM**  
0457 DOMException and the DOMException values used in this  
0458 specification are defined in [DOM4].  
0459

0460 **ECMAScript**  
0461 %ArrayBuffer% is defined in [ECMAScript].  
0462

0463 **HTML**  
0464 The concepts of relevant settings object, origin, opaque origin,  
0465 and is a registrable domain suffix of or is equal to are defined  
0466 in [HTML52].  
0467

0468 **Web IDL**  
0469 Many of the interface definitions and all of the IDL in this  
0470 specification depend on [WebIDL-1]. This updated version of the  
0471 Web IDL standard adds support for Promises, which are now the  
0472 preferred mechanism for asynchronous interaction in all new web  
0473 APIs.  
0474

0475 **FIDO AppID**  
0476 The algorithms for determining the FacetID of a calling  
0477 application and determining if a caller's FacetID is authorized  
0478 for an AppID (used only in the AppID extension) are defined by  
0479 [FIDO-APPID].  
0480

0481 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",  
0482 "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this  
0483 document are to be interpreted as described in [RFC2119].  
0484

0485 **4. Terminology**  
0486

0487 **Assertion**  
0488 See Authentication Assertion.  
0489

0420 classes MUST be done using the CTAP2 canonical CBOR encoding form. All  
0421 decoders of the above conformance classes SHOULD reject CBOR that is  
0422 not validly encoded in the CTAP2 canonical CBOR encoding form and  
0423 SHOULD reject messages with duplicate map keys.  
0424

0425 **3. Dependencies**  
0426

0427 This specification relies on several other underlying specifications,  
0428 listed below and in Terms defined by reference.  
0429

0430 **Base64url encoding**  
0431 The term Base64url Encoding refers to the base64 encoding using  
0432 the URL- and filename-safe character set defined in Section 5 of  
0433 [RFC4648], with all trailing '=' characters omitted (as  
0434 permitted by Section 3.2) and without the inclusion of any line  
0435 breaks, whitespace, or other additional characters.  
0436

0437 **CBOR**  
0438 A number of structures in this specification, including  
0439 attestation statements and extensions, are encoded using the  
0440 CTAP2 canonical CBOR encoding form of the Compact Binary Object  
0441 Representation (CBOR) [RFC7049], as defined in [FIDO-CTAP].  
0442

0443 **CDDL**  
0444 This specification describes the syntax of all CBOR-encoded data  
0445 using the CBOR Data Definition Language (CDDL) [CDDL].  
0446

0447 **COSE**  
0448 CBOR Object Signing and Encryption (COSE) [RFC8152]. The IANA  
0449 COSE Algorithms registry established by this specification is  
0450 also used.  
0451

0452 **Credential Management**  
0453 The API described in this document is an extension of the  
0454 Credential concept defined in [CREDENTIAL-MANAGEMENT-1].  
0455

0456 **DOM**  
0457 DOMException and the DOMException values used in this  
0458 specification are defined in [DOM4].  
0459

0460 **ECMAScript**  
0461 %ArrayBuffer% is defined in [ECMAScript].  
0462

0463 **HTML**  
0464 The concepts of relevant settings object, origin, opaque origin,  
0465 and is a registrable domain suffix of or is equal to are defined  
0466 in [HTML52].  
0467

0468 **Web IDL**  
0469 Many of the interface definitions and all of the IDL in this  
0470 specification depend on [WebIDL-1]. This updated version of the  
0471 Web IDL standard adds support for Promises, which are now the  
0472 preferred mechanism for asynchronous interaction in all new web  
0473 APIs.  
0474

0475 **FIDO AppID**  
0476 The algorithms for determining the FacetID of a calling  
0477 application and determining if a caller's FacetID is authorized  
0478 for an AppID (used only in the AppID extension) are defined by  
0479 [FIDO-APPID].  
0480

0481 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",  
0482 "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this  
0483 document are to be interpreted as described in [RFC2119].  
0484

0485 **4. Terminology**  
0486

0487 **Assertion**  
0488 See Authentication Assertion.  
0489

049C **Attestation**  
 0491 Generally, attestation is a statement serving to bear witness,  
 0492 confirm, or authenticate. In the WebAuthn context, attestation  
 0493 is employed to attest to the provenance of an authenticator and  
 0494 the data it emits; including, for example: credential IDs,  
 0495 credential key pairs, signature counters, etc. An attestation  
 0496 statement is conveyed in an attestation object during  
 0497 registration. See also 6.3 Attestation and Figure 3. Whether or  
 0498 how the client platform conveys the attestation statement and  
 0499 AAGUID portions of the attestation object to the Relying Party  
 0500 is described by attestation conveyance.  
 0501  
 0502 **Attestation Certificate**  
 0503 A X.509 Certificate for the attestation key pair used by an  
 0504 authenticator to attest to its manufacture and capabilities. At  
 0505 registration time, the authenticator uses the attestation  
 0506 private key to sign the Relying Party-specific credential public  
 0507 key (and additional data) that it generates and returns via the  
 0508 authenticatorMakeCredential operation. Relying Parties use the  
 0509 attestation public key conveyed in the attestation certificate  
 0510 to verify the attestation signature. Note that in the case of  
 0511 self attestation, the authenticator has no distinct attestation  
 0512 key pair nor attestation certificate, see self attestation for  
 0513 details.  
 0514  
 0515 **Authentication**  
 0516 The ceremony where a user, and the user's computing device(s)  
 0517 (containing at least one authenticator) work in concert to  
 0518 cryptographically prove to a Relying Party that the user  
 0519 controls the credential private key associated with a  
 0520 previously-registered public key credential (see Registration).  
 0521 Note that this includes a test of user presence or user  
 0522 verification.  
 0523  
 0524 **Authentication Assertion**  
 0525 The cryptographically signed AuthenticatorAssertionResponse  
 0526 object returned by an authenticator as the result of an  
 0527 authenticatorGetAssertion operation.  
 0528  
 0529 This corresponds to the [CREDENTIAL-MANAGEMENT-1]  
 0530 specification's single-use credentials.  
 0531  
 0532 **Authenticator**  
 0533 A cryptographic entity used by a WebAuthn Client to (i) generate  
 0534 a public key credential and register it with a Relying Party,  
 0535 and (ii) authenticate by potentially verifying the user, and  
 0536 then cryptographically signing and returning, in the form of an  
 0537 Authentication Assertion, a challenge and other data presented  
 0538 by a Relying Party (in concert with the WebAuthn Client).  
 0539  
 0540 **Authorization Gesture**  
 0541 An authorization gesture is a physical interaction performed by  
 0542 a user with an authenticator as part of a ceremony, such as  
 0543 registration or authentication. By making such an authorization  
 0544 gesture, a user provides consent for (i.e., authorizes) a  
 0545 ceremony to proceed. This MAY involve user verification if the  
 0546 employed authenticator is capable, or it MAY involve a simple  
 0547 test of user presence.  
 0548  
 0549 **Biometric Recognition**  
 0550 The automated recognition of individuals based on their  
 0551 biological and behavioral characteristics  
 0552 [ISOBiometricVocabulary].  
 0553  
 0554 **Biometric Authenticator**  
 0555 Any authenticator that implements biometric recognition.  
 0556  
 0557 **Ceremony**  
 0558 The concept of a ceremony [Ceremony] is an extension of the  
 0559 concept of a network protocol, with human nodes alongside

049C **Attestation**  
 0491 Generally, attestation is a statement serving to bear witness,  
 0492 confirm, or authenticate. In the WebAuthn context, attestation  
 0493 is employed to attest to the provenance of an authenticator and  
 0494 the data it emits; including, for example: credential IDs,  
 0495 credential key pairs, signature counters, etc. An attestation  
 0496 statement is conveyed in an attestation object during  
 0497 registration. See also 6.3 Attestation and Figure 3. Whether or  
 0498 how the client platform conveys the attestation statement and  
 0499 AAGUID portions of the attestation object to the Relying Party  
 0500 is described by attestation conveyance.  
 0501  
 0502 **Attestation Certificate**  
 0503 A X.509 Certificate for the attestation key pair used by an  
 0504 authenticator to attest to its manufacture and capabilities. At  
 0505 registration time, the authenticator uses the attestation  
 0506 private key to sign the Relying Party-specific credential public  
 0507 key (and additional data) that it generates and returns via the  
 0508 authenticatorMakeCredential operation. Relying Parties use the  
 0509 attestation public key conveyed in the attestation certificate  
 0510 to verify the attestation signature. Note that in the case of  
 0511 self attestation, the authenticator has no distinct attestation  
 0512 key pair nor attestation certificate, see self attestation for  
 0513 details.  
 0514  
 0515 **Authentication**  
 0516 The ceremony where a user, and the user's computing device(s)  
 0517 (containing at least one authenticator) work in concert to  
 0518 cryptographically prove to a Relying Party that the user  
 0519 controls the credential private key associated with a  
 0520 previously-registered public key credential (see Registration).  
 0521 Note that this includes a test of user presence or user  
 0522 verification.  
 0523  
 0524 **Authentication Assertion**  
 0525 The cryptographically signed AuthenticatorAssertionResponse  
 0526 object returned by an authenticator as the result of an  
 0527 authenticatorGetAssertion operation.  
 0528  
 0529 This corresponds to the [CREDENTIAL-MANAGEMENT-1]  
 0530 specification's single-use credentials.  
 0531  
 0532 **Authenticator**  
 0533 A cryptographic entity used by a WebAuthn Client to (i) generate  
 0534 a public key credential and register it with a Relying Party,  
 0535 and (ii) authenticate by potentially verifying the user, and  
 0536 then cryptographically signing and returning, in the form of an  
 0537 Authentication Assertion, a challenge and other data presented  
 0538 by a Relying Party (in concert with the WebAuthn Client).  
 0539  
 0540 **Authorization Gesture**  
 0541 An authorization gesture is a physical interaction performed by  
 0542 a user with an authenticator as part of a ceremony, such as  
 0543 registration or authentication. By making such an authorization  
 0544 gesture, a user provides consent for (i.e., authorizes) a  
 0545 ceremony to proceed. This MAY involve user verification if the  
 0546 employed authenticator is capable, or it MAY involve a simple  
 0547 test of user presence.  
 0548  
 0549 **Biometric Recognition**  
 0550 The automated recognition of individuals based on their  
 0551 biological and behavioral characteristics  
 0552 [ISOBiometricVocabulary].  
 0553  
 0554 **Biometric Authenticator**  
 0555 Any authenticator that implements biometric recognition.  
 0556  
 0557 **Ceremony**  
 0558 The concept of a ceremony [Ceremony] is an extension of the  
 0559 concept of a network protocol, with human nodes alongside



056C computer nodes and with communication links that include user  
0561 interface(s), human-to-human communication, and transfers of  
0562 physical objects that carry data. What is out-of-band to a  
0563 protocol is in-band to a ceremony. In this specification,  
0564 Registration and Authentication are ceremonies, and an  
0565 authorization gesture is often a component of those ceremonies.

0566 **Client**  
0567 See WebAuthn Client, Conforming User Agent.

0568 **Client-Side**  
0569 This refers in general to the combination of the user's platform  
0570 device, user agent, authenticators, and everything gluing it all  
0571 together.

0572 **Client-side-resident Credential Private Key**  
0573 A Client-side-resident Credential Private Key is stored either  
0574 on the client platform, or in some cases on the authenticator  
0575 itself, e.g., in the case of a discrete first-factor roaming  
0576 authenticator. Such client-side credential private key storage  
0577 has the property that the authenticator is able to select the  
0578 credential private key given only an RP ID, possibly with user  
0579 assistance (e.g., by providing the user a pick list of  
0580 credentials associated with the RP ID). By definition, the  
0581 private key is always exclusively controlled by the  
0582 authenticator. In the case of a Client-side-resident Credential  
0583 Private Key, the authenticator might offload storage of wrapped  
0584 key material to the client platform, but the client platform is  
0585 not expected to offload the key storage to remote entities (e.g.  
0586 RP Server).

0587 **Conforming User Agent**  
0588 A user agent implementing, in conjunction with the underlying  
0589 platform, the Web Authentication API and algorithms given in  
0590 this specification, and handling communication between  
0591 authenticators and Relying Parties.

0592 **Credential ID**  
0593 A probabilistically-unique byte sequence identifying a public  
0594 key credential source and its authentication assertions.  
0595  
0596 Credential IDs are generated by authenticators in two forms:  
0597  
0598 1. At least 16 bytes that include at least 100 bits of entropy,  
0599 or  
0600 2. The public key credential source, without its Credential ID,  
0601 encrypted so only its managing authenticator can decrypt it.  
0602 This form allows the authenticator to be nearly stateless, by  
0603 having the Relying Party store any necessary state.  
0604 Note: [FIDO-UAF-AUTHNR-CMDS] includes guidance on encryption  
0605 techniques under "Security Guidelines".

0606 Relying Parties do not need to distinguish these two Credential  
0607 ID forms.

0608 **Credential Public Key**  
0609 **User Public Key**  
0610 The public key portion of a Relying Party-specific credential  
0611 key pair, generated by an authenticator and returned to a  
0612 Relying Party at registration time (see also public key  
0613 credential). The private key portion of the credential key pair  
0614 is known as the credential private key. Note that in the case of  
0615 self attestation, the credential key pair is also used as the  
0616 attestation key pair, see self attestation for details.

0617 Note: The credential public key is referred to as the user  
0618 public key in FIDO UAF [UAFProtocol], and in FIDO U2F  
0619 [FIDO-U2F-Message-Formats] and some parts of this specification  
0620 that relate to it.

056C computer nodes and with communication links that include user  
0561 interface(s), human-to-human communication, and transfers of  
0562 physical objects that carry data. What is out-of-band to a  
0563 protocol is in-band to a ceremony. In this specification,  
0564 Registration and Authentication are ceremonies, and an  
0565 authorization gesture is often a component of those ceremonies.

0566 **Client**  
0567 See WebAuthn Client, Conforming User Agent.

0568 **Client-Side**  
0569 This refers in general to the combination of the user's platform  
0570 device, user agent, authenticators, and everything gluing it all  
0571 together.

0572 **Client-side-resident Credential Private Key**  
0573 A Client-side-resident Credential Private Key is stored either  
0574 on the client platform, or in some cases on the authenticator  
0575 itself, e.g., in the case of a discrete first-factor roaming  
0576 authenticator. Such client-side credential private key storage  
0577 has the property that the authenticator is able to select the  
0578 credential private key given only an RP ID, possibly with user  
0579 assistance (e.g., by providing the user a pick list of  
0580 credentials associated with the RP ID). By definition, the  
0581 private key is always exclusively controlled by the  
0582 authenticator. In the case of a Client-side-resident Credential  
0583 Private Key, the authenticator might offload storage of wrapped  
0584 key material to the client platform, but the client platform is  
0585 not expected to offload the key storage to remote entities (e.g.  
0586 RP Server).

0587 **Conforming User Agent**  
0588 A user agent implementing, in conjunction with the underlying  
0589 platform, the Web Authentication API and algorithms given in  
0590 this specification, and handling communication between  
0591 authenticators and Relying Parties.

0592 **Credential ID**  
0593 A probabilistically-unique byte sequence identifying a public  
0594 key credential source and its authentication assertions.  
0595  
0596 Credential IDs are generated by authenticators in two forms:  
0597  
0598 1. At least 16 bytes that include at least 100 bits of entropy,  
0599 or  
0600 2. The public key credential source, without its Credential ID,  
0601 encrypted so only its managing authenticator can decrypt it.  
0602 This form allows the authenticator to be nearly stateless, by  
0603 having the Relying Party store any necessary state.  
0604 Note: [FIDO-UAF-AUTHNR-CMDS] includes guidance on encryption  
0605 techniques under "Security Guidelines".

0606 Relying Parties do not need to distinguish these two Credential  
0607 ID forms.

0608 **Credential Public Key**  
0609 **User Public Key**  
0610 The public key portion of a Relying Party-specific credential  
0611 key pair, generated by an authenticator and returned to a  
0612 Relying Party at registration time (see also public key  
0613 credential). The private key portion of the credential key pair  
0614 is known as the credential private key. Note that in the case of  
0615 self attestation, the credential key pair is also used as the  
0616 attestation key pair, see self attestation for details.

0617 Note: The credential public key is referred to as the user  
0618 public key in FIDO UAF [UAFProtocol], and in FIDO U2F  
0619 [FIDO-U2F-Message-Formats] and some parts of this specification  
0620 that relate to it.

0630 Human Palatability  
0631 An identifier that is human-palatable is intended to be  
0632 rememberable and reproducible by typical human users, in  
0633 contrast to identifiers that are, for example, randomly  
0634 generated sequences of bits [EduPersonObjectClassSpec].  
0635

0636 Public Key Credential Source  
0637 A credential source ([CREDENTIAL-MANAGEMENT-1]) used by an  
0638 authenticator to generate authentication assertions. A public  
0639 key credential source consists of a struct with the following  
0640 items:

0641 type  
0642 whose value is of PublicKeyCredentialType, defaulting to  
0643 public-key.  
0644  
0645 id  
0646 A Credential ID.  
0647  
0648 privateKey  
0649 The credential private key.  
0650  
0651 rpId  
0652 The Relying Party Identifier, for the Relying Party this  
0653 public key credential source is associated with.  
0654  
0655 userHandle  
0656 The user handle associated when this public key credential  
0657 source was created. This item is nullable.  
0658  
0659 otherUI  
0660 Optional other information used by the authenticator to  
0661 inform its UI. For example, this might include the user's  
0662 displayName.  
0663  
0664 The authenticatorMakeCredential operation creates a public key  
0665 credential source bound to a managing authenticator and returns  
0666 the credential public key associated with its credential private  
0667 key. The Relying Party can use this credential public key to  
0668 verify the authentication assertions created by this public key  
0669 credential source.  
0670  
0671 Public Key Credential  
0672 Generically, a credential is data one entity presents to another  
0673 in order to authenticate the former to the latter [RFC4949]. The  
0674 term public key credential refers to one of: a public key  
0675 credential source, the possibly-attested credential public key  
0676 corresponding to a public key credential source, or an  
0677 authentication assertion. Which one is generally determined by  
0678 context.  
0679  
0680 Note: This is a willful violation of [RFC4949]. In English, a  
0681 "credential" is both a) the thing presented to prove a statement  
0682 and b) intended to be used multiple times. It's impossible to  
0683 achieve both criteria securely with a single piece of data in a  
0684 public key system. [RFC4949] chooses to define a credential as  
0685 the thing that can be used multiple times (the public key),  
0686 while this specification gives "credential" the English term's  
0687 flexibility. This specification uses more specific terms to  
0688 identify the data related to an [RFC4949] credential:

0689 "Authentication information" (possibly including a private key)  
0690 Public key credential source  
0691  
0692 "Signed value"  
0693 Authentication assertion  
0694  
0695 [RFC4949] "credential"  
0696 Credential public key or attestation object  
0697  
0698  
0699

0630 Human Palatability  
0631 An identifier that is human-palatable is intended to be  
0632 rememberable and reproducible by typical human users, in  
0633 contrast to identifiers that are, for example, randomly  
0634 generated sequences of bits [EduPersonObjectClassSpec].  
0635

0636 Public Key Credential Source  
0637 A credential source ([CREDENTIAL-MANAGEMENT-1]) used by an  
0638 authenticator to generate authentication assertions. A public  
0639 key credential source consists of a struct with the following  
0640 items:

0641 type  
0642 whose value is of PublicKeyCredentialType, defaulting to  
0643 public-key.  
0644  
0645 id  
0646 A Credential ID.  
0647  
0648 privateKey  
0649 The credential private key.  
0650  
0651 rpId  
0652 The Relying Party Identifier, for the Relying Party this  
0653 public key credential source is associated with.  
0654  
0655 userHandle  
0656 The user handle associated when this public key credential  
0657 source was created. This item is nullable.  
0658  
0659 otherUI  
0660 Optional other information used by the authenticator to  
0661 inform its UI. For example, this might include the user's  
0662 displayName.  
0663  
0664 The authenticatorMakeCredential operation creates a public key  
0665 credential source bound to a managing authenticator and returns  
0666 the credential public key associated with its credential private  
0667 key. The Relying Party can use this credential public key to  
0668 verify the authentication assertions created by this public key  
0669 credential source.  
0670  
0671 Public Key Credential  
0672 Generically, a credential is data one entity presents to another  
0673 in order to authenticate the former to the latter [RFC4949]. The  
0674 term public key credential refers to one of: a public key  
0675 credential source, the possibly-attested credential public key  
0676 corresponding to a public key credential source, or an  
0677 authentication assertion. Which one is generally determined by  
0678 context.  
0679  
0680 Note: This is a willful violation of [RFC4949]. In English, a  
0681 "credential" is both a) the thing presented to prove a statement  
0682 and b) intended to be used multiple times. It's impossible to  
0683 achieve both criteria securely with a single piece of data in a  
0684 public key system. [RFC4949] chooses to define a credential as  
0685 the thing that can be used multiple times (the public key),  
0686 while this specification gives "credential" the English term's  
0687 flexibility. This specification uses more specific terms to  
0688 identify the data related to an [RFC4949] credential:

0689 "Authentication information" (possibly including a private key)  
0690 Public key credential source  
0691  
0692 "Signed value"  
0693 Authentication assertion  
0694  
0695 [RFC4949] "credential"  
0696 Credential public key or attestation object  
0697  
0698  
0699

0700 At registration time, the authenticator creates an asymmetric  
 0701 key pair, and stores its private key portion and information  
 0702 from the Relying Party into a public key credential source. The  
 0703 public key portion is returned to the Relying Party, who then  
 0704 stores it in conjunction with the present user's account.  
 0705 Subsequently, only that Relying Party, as identified by its RP  
 0706 ID, is able to employ the public key credential in  
 0707 authentication ceremonies, via the get() method. The Relying  
 0708 Party uses its stored copy of the credential public key to  
 0709 verify the resultant authentication assertion.

0710 **Rate Limiting**

0711 The process (also known as throttling) by which an authenticator  
 0712 implements controls against brute force attacks by limiting the  
 0713 number of consecutive failed authentication attempts within a  
 0714 given period of time. If the limit is reached, the authenticator  
 0715 should impose a delay that increases exponentially with each  
 0716 successive attempt, or disable the current authentication  
 0717 modality and offer a different authentication factor if  
 0718 available. Rate limiting is often implemented as an aspect of  
 0719 user verification.

0720 **Registration**

0721 The ceremony where a user, a Relying Party, and the user's  
 0722 computing device(s) (containing at least one authenticator) work  
 0723 in concert to create a public key credential and associate it  
 0724 with the user's Relying Party account. Note that this includes  
 0725 employing a test of user presence or user verification.

0726 **Relying Party**

0727 The entity whose web application utilizes the Web Authentication  
 0728 API to register and authenticate users. See Registration and  
 0729 Authentication, respectively.

0730 Note: While the term Relying Party is used in other contexts  
 0731 (e.g., X.509 and OAuth), an entity acting as a Relying Party in  
 0732 one context is not necessarily a Relying Party in other  
 0733 contexts.

0734 **Relying Party Identifier**

0735 **RP ID**

0736 A valid domain string that identifies the Relying Party on whose  
 0737 behalf a given registration or authentication ceremony is being  
 0738 performed. A public key credential can only be used for  
 0739 authentication with the same entity (as identified by RP ID) it  
 0740 was registered with. By default, the RP ID for a WebAuthn  
 0741 operation is set to the caller's origin's effective domain. This  
 0742 default MAY be overridden by the caller, as long as the  
 0743 caller-specified RP ID value is a registrable domain suffix of  
 0744 or is equal to the caller's origin's effective domain. See also  
 0745 5.1.3 Create a new credential - PublicKeyCredential's  
 0746 [[Create]](origin, options, sameOriginWithAncestors) method and  
 0747 5.1.4 Use an existing credential to make an assertion -  
 0748 PublicKeyCredential's [[Get]](options) method.

0749 Note: A Public key credential's scope is for a Relying Party's  
 0750 origin, with the following restrictions and relaxations:

- 0751 + The scheme is always https (i.e., a restriction), and,
- 0752 + the host may be equal to the Relying Party's origin's  
 0753 effective domain, or it may be equal to a registrable domain  
 0754 suffix of the Relying Party's origin's effective domain (i.e.,  
 0755 an available relaxation), and,
- 0756 + all (TCP) ports on that host (i.e., a relaxation).

0757 This is done in order to match the behavior of pervasively  
 0758 deployed ambient credentials (e.g., cookies, [RFC6265]). Please  
 0759 note that this is a greater relaxation of "same-origin"  
 0760 restrictions than what document.domain's setter provides.

0761

0700 At registration time, the authenticator creates an asymmetric  
 0701 key pair, and stores its private key portion and information  
 0702 from the Relying Party into a public key credential source. The  
 0703 public key portion is returned to the Relying Party, who then  
 0704 stores it in conjunction with the present user's account.  
 0705 Subsequently, only that Relying Party, as identified by its RP  
 0706 ID, is able to employ the public key credential in  
 0707 authentication ceremonies, via the get() method. The Relying  
 0708 Party uses its stored copy of the credential public key to  
 0709 verify the resultant authentication assertion.

0710 **Rate Limiting**

0711 The process (also known as throttling) by which an authenticator  
 0712 implements controls against brute force attacks by limiting the  
 0713 number of consecutive failed authentication attempts within a  
 0714 given period of time. If the limit is reached, the authenticator  
 0715 should impose a delay that increases exponentially with each  
 0716 successive attempt, or disable the current authentication  
 0717 modality and offer a different authentication factor if  
 0718 available. Rate limiting is often implemented as an aspect of  
 0719 user verification.

0720 **Registration**

0721 The ceremony where a user, a Relying Party, and the user's  
 0722 computing device(s) (containing at least one authenticator) work  
 0723 in concert to create a public key credential and associate it  
 0724 with the user's Relying Party account. Note that this includes  
 0725 employing a test of user presence or user verification.

0726 **Relying Party**

0727 The entity whose web application utilizes the Web Authentication  
 0728 API to register and authenticate users. See Registration and  
 0729 Authentication, respectively.

0730 Note: While the term Relying Party is used in other contexts  
 0731 (e.g., X.509 and OAuth), an entity acting as a Relying Party in  
 0732 one context is not necessarily a Relying Party in other  
 0733 contexts.

0734 **Relying Party Identifier**

0735 **RP ID**

0736 A valid domain string that identifies the Relying Party on whose  
 0737 behalf a given registration or authentication ceremony is being  
 0738 performed. A public key credential can only be used for  
 0739 authentication with the same entity (as identified by RP ID) it  
 0740 was registered with. By default, the RP ID for a WebAuthn  
 0741 operation is set to the caller's origin's effective domain. This  
 0742 default MAY be overridden by the caller, as long as the  
 0743 caller-specified RP ID value is a registrable domain suffix of  
 0744 or is equal to the caller's origin's effective domain. See also  
 0745 5.1.3 Create a new credential - PublicKeyCredential's  
 0746 [[Create]](origin, options, sameOriginWithAncestors) method and  
 0747 5.1.4 Use an existing credential to make an assertion -  
 0748 PublicKeyCredential's [[Get]](options) method.

0749 Note: A Public key credential's scope is for a Relying Party's  
 0750 origin, with the following restrictions and relaxations:

- 0751 + The scheme is always https (i.e., a restriction), and,
- 0752 + the host may be equal to the Relying Party's origin's  
 0753 effective domain, or it may be equal to a registrable domain  
 0754 suffix of the Relying Party's origin's effective domain (i.e.,  
 0755 an available relaxation), and,
- 0756 + all (TCP) ports on that host (i.e., a relaxation).

0757 This is done in order to match the behavior of pervasively  
 0758 deployed ambient credentials (e.g., cookies, [RFC6265]). Please  
 0759 note that this is a greater relaxation of "same-origin"  
 0760 restrictions than what document.domain's setter provides.

0761

077C **Test of User Presence**  
0771 A test of user presence is a simple form of authorization  
0772 gesture and technical process where a user interacts with an  
0773 authenticator by (typically) simply touching it (other  
0774 modalities may also exist), yielding a boolean result. Note that  
0775 this does not constitute user verification because a user  
0776 presence test, by definition, is not capable of biometric  
0777 recognition, nor does it involve the presentation of a shared  
0778 secret such as a password or PIN.

078C **User Consent**  
0781 User consent means the user agrees with what they are being  
0782 asked, i.e., it encompasses reading and understanding prompts.  
0783 An authorization gesture is a ceremony component often employed  
0784 to indicate user consent.

078E **User Handle**  
0787 The user handle is specified by a Relying Party and is a unique  
0788 identifier for a user account with that Relying Party. A user  
0789 handle is an opaque byte sequence with a maximum size of 64  
079C bytes.

0792 The user handle is not meant to be displayed to the user, but is  
0793 used by the Relying Party to control the number of credentials -  
0794 an authenticator will never contain more than one credential for  
0795 a given Relying Party under the same user handle.

079E **User Verification**  
0799 The technical process by which an authenticator locally  
0799 authorizes the invocation of the authenticatorMakeCredential and  
0800 authenticatorGetAssertion operations. User verification MAY be  
0801 instigated through various authorization gesture modalities; for  
0802 example, through a touch plus pin code, password entry, or  
0803 biometric recognition (e.g., presenting a fingerprint)  
0804 [ISOBiometricVocabulary]. The intent is to be able to  
0805 distinguish individual users. Note that invocation of the  
0806 authenticatorMakeCredential and authenticatorGetAssertion  
0807 operations implies use of key material managed by the  
0808 authenticator. Note that for security, user verification and use  
0809 of credential private keys must occur within a single logical  
0810 security boundary defining the authenticator.

0811 **User Present**  
0813 UP  
0814 Upon successful completion of a user presence test, the user is  
0815 said to be "present".

081E **User Verified**  
0817 UV  
0818 Upon successful completion of a user verification process, the  
0819 user is said to be "verified".

082C **WebAuthn Client**  
0822 Also referred to herein as simply a client. See also Conforming  
0823 User Agent. A WebAuthn Client is an intermediary entity  
0824 typically implemented in the user agent (in whole, or in part).  
0825 Conceptually, it underlies the Web Authentication API and  
0826 embodies the implementation of the [[Create]](origin, options,  
0827 sameOriginWithAncestors) and  
0828 [[DiscoverFromExternalSource]](origin, options,  
0829 sameOriginWithAncestors) internal methods. It is responsible for  
0830 both marshalling the inputs for the underlying authenticator  
0831 operations, and for returning the results of the latter  
0832 operations to the Web Authentication API's callers.

0833 **5. Web Authentication API**  
0834  
0835 This section normatively specifies the API for creating and using  
0836 public key credentials. The basic idea is that the credentials belong  
0837 to the user and are managed by an authenticator, with which the Relying

077C **Test of User Presence**  
0771 A test of user presence is a simple form of authorization  
0772 gesture and technical process where a user interacts with an  
0773 authenticator by (typically) simply touching it (other  
0774 modalities may also exist), yielding a boolean result. Note that  
0775 this does not constitute user verification because a user  
0776 presence test, by definition, is not capable of biometric  
0777 recognition, nor does it involve the presentation of a shared  
0778 secret such as a password or PIN.

078C **User Consent**  
0781 User consent means the user agrees with what they are being  
0782 asked, i.e., it encompasses reading and understanding prompts.  
0783 An authorization gesture is a ceremony component often employed  
0784 to indicate user consent.

078E **User Handle**  
0787 The user handle is specified by a Relying Party and is a unique  
0788 identifier for a user account with that Relying Party. A user  
0789 handle is an opaque byte sequence with a maximum size of 64  
079C bytes.

0792 The user handle is not meant to be displayed to the user, but is  
0793 used by the Relying Party to control the number of credentials -  
0794 an authenticator will never contain more than one credential for  
0795 a given Relying Party under the same user handle.

079E **User Verification**  
0799 The technical process by which an authenticator locally  
0799 authorizes the invocation of the authenticatorMakeCredential and  
0800 authenticatorGetAssertion operations. User verification MAY be  
0801 instigated through various authorization gesture modalities; for  
0802 example, through a touch plus pin code, password entry, or  
0803 biometric recognition (e.g., presenting a fingerprint)  
0804 [ISOBiometricVocabulary]. The intent is to be able to  
0805 distinguish individual users. Note that invocation of the  
0806 authenticatorMakeCredential and authenticatorGetAssertion  
0807 operations implies use of key material managed by the  
0808 authenticator. Note that for security, user verification and use  
0809 of credential private keys must occur within a single logical  
0810 security boundary defining the authenticator.

0811 **User Present**  
0813 UP  
0814 Upon successful completion of a user presence test, the user is  
0815 said to be "present".

081E **User Verified**  
0817 UV  
0818 Upon successful completion of a user verification process, the  
0819 user is said to be "verified".

082C **WebAuthn Client**  
0822 Also referred to herein as simply a client. See also Conforming  
0823 User Agent. A WebAuthn Client is an intermediary entity  
0824 typically implemented in the user agent (in whole, or in part).  
0825 Conceptually, it underlies the Web Authentication API and  
0826 embodies the implementation of the [[Create]](origin, options,  
0827 sameOriginWithAncestors) and  
0828 [[DiscoverFromExternalSource]](origin, options,  
0829 sameOriginWithAncestors) internal methods. It is responsible for  
0830 both marshalling the inputs for the underlying authenticator  
0831 operations, and for returning the results of the latter  
0832 operations to the Web Authentication API's callers.

0833 **5. Web Authentication API**  
0834  
0835 This section normatively specifies the API for creating and using  
0836 public key credentials. The basic idea is that the credentials belong  
0837 to the user and are managed by an authenticator, with which the Relying

0840 Party interacts through the client (consisting of the browser and  
 0841 underlying OS platform). Scripts can (with the user's consent) request  
 0842 the browser to create a new credential for future use by the Relying  
 0843 Party. Scripts can also request the user's permission to perform  
 0844 authentication operations with an existing credential. All such  
 0845 operations are performed in the authenticator and are mediated by the  
 0846 browser and/or platform on the user's behalf. At no point does the  
 0847 script get access to the credentials themselves; it only gets  
 0848 information about the credentials in the form of objects.  
 0849  
 0850 In addition to the above script interface, the authenticator MAY  
 0851 implement (or come with client software that implements) a user  
 0852 interface for management. Such an interface MAY be used, for example,  
 0853 to reset the authenticator to a clean state or to inspect the current  
 0854 state of the authenticator. In other words, such an interface is  
 0855 similar to the user interfaces provided by browsers for managing user  
 0856 state such as history, saved passwords, and cookies. Authenticator  
 0857 management actions such as credential deletion are considered to be the  
 0858 responsibility of such a user interface and are deliberately omitted  
 0859 from the API exposed to scripts.  
 0860  
 0861 The security properties of this API are provided by the client and the  
 0862 authenticator working together. The authenticator, which holds and  
 0863 manages credentials, ensures that all operations are scoped to a  
 0864 particular origin, and cannot be replayed against a different origin,  
 0865 by incorporating the origin in its responses. Specifically, as defined  
 0866 in 6.2 Authenticator operations, the full origin of the requester is  
 0867 included, and signed over, in the attestation object produced when a  
 0868 new credential is created as well as in all assertions produced by  
 0869 WebAuthn credentials.  
 0870  
 0871 Additionally, to maintain user privacy and prevent malicious Relying  
 0872 Parties from probing for the presence of public key credentials  
 0873 belonging to other Relying Parties, each credential is also associated  
 0874 with a Relying Party Identifier, or RP ID. This RP ID is provided by  
 0875 the client to the authenticator for all operations, and the  
 0876 authenticator ensures that credentials created by a Relying Party can  
 0877 only be used in operations requested by the same RP ID. Separating the  
 0878 origin from the RP ID in this way allows the API to be used in cases  
 0879 where a single Relying Party maintains multiple origins.  
 0880  
 0881 The client facilitates these security measures by providing the Relying  
 0882 Party's origin and RP ID to the authenticator for each operation. Since  
 0883 this is an integral part of the WebAuthn security model, user agents  
 0884 only expose this API to callers in secure contexts.  
 0885  
 0886 The Web Authentication API is defined by the union of the Web IDL  
 0887 fragments presented in the following sections. A combined IDL listing  
 0888 is given in the IDL Index.  
 0889  
 0890 **5.1. PublicKeyCredential Interface**  
 0891  
 0892 The PublicKeyCredential interface inherits from Credential  
 0893 [CREDENTIAL-MANAGEMENT-1], and contains the attributes that are  
 0894 returned to the caller when a new credential is created, or a new  
 0895 assertion is requested.  
 0896 [SecureContext, Exposed=Window]  
 0897 interface PublicKeyCredential : Credential {  
 0898 [SameObject] readonly attribute ArrayBuffer rawId;  
 0899 [SameObject] readonly attribute AuthenticatorResponse response;  
 0900 AuthenticationExtensionsClientOutputs getClientExtensionResults();  
 0901 };  
 0902  
 0903 id  
 0904 This attribute is inherited from Credential, though  
 0905 PublicKeyCredential overrides Credential's getter, instead  
 0906 returning the base64url encoding of the data contained in the  
 0907 object's [[identifier]] internal slot.  
 0908  
 0909 rawId

0840 Party interacts through the client (consisting of the browser and  
 0841 underlying OS platform). Scripts can (with the user's consent) request  
 0842 the browser to create a new credential for future use by the Relying  
 0843 Party. Scripts can also request the user's permission to perform  
 0844 authentication operations with an existing credential. All such  
 0845 operations are performed in the authenticator and are mediated by the  
 0846 browser and/or platform on the user's behalf. At no point does the  
 0847 script get access to the credentials themselves; it only gets  
 0848 information about the credentials in the form of objects.  
 0849  
 0850 In addition to the above script interface, the authenticator MAY  
 0851 implement (or come with client software that implements) a user  
 0852 interface for management. Such an interface MAY be used, for example,  
 0853 to reset the authenticator to a clean state or to inspect the current  
 0854 state of the authenticator. In other words, such an interface is  
 0855 similar to the user interfaces provided by browsers for managing user  
 0856 state such as history, saved passwords, and cookies. Authenticator  
 0857 management actions such as credential deletion are considered to be the  
 0858 responsibility of such a user interface and are deliberately omitted  
 0859 from the API exposed to scripts.  
 0860  
 0861 The security properties of this API are provided by the client and the  
 0862 authenticator working together. The authenticator, which holds and  
 0863 manages credentials, ensures that all operations are scoped to a  
 0864 particular origin, and cannot be replayed against a different origin,  
 0865 by incorporating the origin in its responses. Specifically, as defined  
 0866 in 6.2 Authenticator operations, the full origin of the requester is  
 0867 included, and signed over, in the attestation object produced when a  
 0868 new credential is created as well as in all assertions produced by  
 0869 WebAuthn credentials.  
 0870  
 0871 Additionally, to maintain user privacy and prevent malicious Relying  
 0872 Parties from probing for the presence of public key credentials  
 0873 belonging to other Relying Parties, each credential is also associated  
 0874 with a Relying Party Identifier, or RP ID. This RP ID is provided by  
 0875 the client to the authenticator for all operations, and the  
 0876 authenticator ensures that credentials created by a Relying Party can  
 0877 only be used in operations requested by the same RP ID. Separating the  
 0878 origin from the RP ID in this way allows the API to be used in cases  
 0879 where a single Relying Party maintains multiple origins.  
 0880  
 0881 The client facilitates these security measures by providing the Relying  
 0882 Party's origin and RP ID to the authenticator for each operation. Since  
 0883 this is an integral part of the WebAuthn security model, user agents  
 0884 only expose this API to callers in secure contexts.  
 0885  
 0886 The Web Authentication API is defined by the union of the Web IDL  
 0887 fragments presented in the following sections. A combined IDL listing  
 0888 is given in the IDL Index.  
 0889  
 0890 **5.1. PublicKeyCredential Interface**  
 0891  
 0892 The PublicKeyCredential interface inherits from Credential  
 0893 [CREDENTIAL-MANAGEMENT-1], and contains the attributes that are  
 0894 returned to the caller when a new credential is created, or a new  
 0895 assertion is requested.  
 0896 [SecureContext, Exposed=Window]  
 0897 interface PublicKeyCredential : Credential {  
 0898 [SameObject] readonly attribute ArrayBuffer rawId;  
 0899 [SameObject] readonly attribute AuthenticatorResponse response;  
 0900 AuthenticationExtensionsClientOutputs getClientExtensionResults();  
 0901 };  
 0902  
 0903 id  
 0904 This attribute is inherited from Credential, though  
 0905 PublicKeyCredential overrides Credential's getter, instead  
 0906 returning the base64url encoding of the data contained in the  
 0907 object's [[identifier]] internal slot.  
 0908  
 0909 rawId

0910 This attribute returns the ArrayBuffer contained in the  
0911 [[identifier]] internal slot.  
0912  
0913 response, of type AuthenticatorResponse, readonly  
0914 This attribute contains the authenticator's response to the  
0915 client's request to either create a public key credential, or  
0916 generate an authentication assertion. If the PublicKeyCredential  
0917 is created in response to create(), this attribute's value will  
0918 be an AuthenticatorAttestationResponse, otherwise, the  
0919 PublicKeyCredential was created in response to get(), and this  
0920 attribute's value will be an AuthenticatorAssertionResponse.  
0921  
0922 getClientExtensionResults()  
0923 This operation returns the value of [[clientExtensionsResults]],  
0924 which is a map containing extension identifier -> client  
0925 extension output entries produced by the extension's client  
0926 extension processing.  
0927  
0928 [[type]]  
0929 The PublicKeyCredential interface object's [[type]] internal  
0930 slot's value is the string "public-key".  
0931  
0932 Note: This is reflected via the type attribute getter inherited  
0933 from Credential.  
0934  
0935 [[discovery]]  
0936 The PublicKeyCredential interface object's [[discovery]]  
0937 internal slot's value is "remote".  
0938  
0939 [[identifier]]  
0940 This internal slot contains the credential ID, chosen by the  
0941 platform with help from the authenticator. The credential ID is  
0942 used to look up credentials for use, and is therefore expected  
0943 to be globally unique with high probability across all  
0944 credentials of the same type, across all authenticators.  
0945  
0946 Note: This API does not constrain the format or length of this  
0947 identifier, except that it MUST be sufficient for the platform  
0948 to uniquely select a key. For example, an authenticator without  
0949 on-board storage may create identifiers containing a credential  
0950 private key wrapped with a symmetric key that is burned into the  
0951 authenticator.  
0952  
0953 [[clientExtensionsResults]]  
0954 This internal slot contains the results of processing client  
0955 extensions requested by the Relying Party upon the Relying  
0956 Party's invocation of either navigator.credentials.create() or  
0957 navigator.credentials.get().  
0958  
0959 PublicKeyCredential's interface object inherits Credential's  
0960 implementation of [[CollectFromCredentialStore]](origin, options,  
0961 sameOriginWithAncestors), and defines its own implementation of  
0962 [[Create]](origin, options, sameOriginWithAncestors),  
0963 [[DiscoverFromExternalSource]](origin, options,  
0964 sameOriginWithAncestors), and [[Store]](credential,  
0965 sameOriginWithAncestors).  
0966  
0967 5.1.1. CredentialCreationOptions Dictionary Extension  
0968  
0969 To support registration via navigator.credentials.create(), this  
0970 document extends the CredentialCreationOptions dictionary as follows:  
0971 partial dictionary CredentialCreationOptions {  
0972 PublicKeyCredentialCreationOptions publicKey;  
0973 };  
0974  
0975 5.1.2. CredentialRequestOptions Dictionary Extension  
0976  
0977 To support obtaining assertions via navigator.credentials.get(), this  
0978 document extends the CredentialRequestOptions dictionary as follows:  
0979 partial dictionary CredentialRequestOptions {

0910 This attribute returns the ArrayBuffer contained in the  
0911 [[identifier]] internal slot.  
0912  
0913 response, of type AuthenticatorResponse, readonly  
0914 This attribute contains the authenticator's response to the  
0915 client's request to either create a public key credential, or  
0916 generate an authentication assertion. If the PublicKeyCredential  
0917 is created in response to create(), this attribute's value will  
0918 be an AuthenticatorAttestationResponse, otherwise, the  
0919 PublicKeyCredential was created in response to get(), and this  
0920 attribute's value will be an AuthenticatorAssertionResponse.  
0921  
0922 getClientExtensionResults()  
0923 This operation returns the value of [[clientExtensionsResults]],  
0924 which is a map containing extension identifier -> client  
0925 extension output entries produced by the extension's client  
0926 extension processing.  
0927  
0928 [[type]]  
0929 The PublicKeyCredential interface object's [[type]] internal  
0930 slot's value is the string "public-key".  
0931  
0932 Note: This is reflected via the type attribute getter inherited  
0933 from Credential.  
0934  
0935 [[discovery]]  
0936 The PublicKeyCredential interface object's [[discovery]]  
0937 internal slot's value is "remote".  
0938  
0939 [[identifier]]  
0940 This internal slot contains the credential ID, chosen by the  
0941 platform with help from the authenticator. The credential ID is  
0942 used to look up credentials for use, and is therefore expected  
0943 to be globally unique with high probability across all  
0944 credentials of the same type, across all authenticators.  
0945  
0946 Note: This API does not constrain the format or length of this  
0947 identifier, except that it MUST be sufficient for the platform  
0948 to uniquely select a key. For example, an authenticator without  
0949 on-board storage may create identifiers containing a credential  
0950 private key wrapped with a symmetric key that is burned into the  
0951 authenticator.  
0952  
0953 [[clientExtensionsResults]]  
0954 This internal slot contains the results of processing client  
0955 extensions requested by the Relying Party upon the Relying  
0956 Party's invocation of either navigator.credentials.create() or  
0957 navigator.credentials.get().  
0958  
0959 PublicKeyCredential's interface object inherits Credential's  
0960 implementation of [[CollectFromCredentialStore]](origin, options,  
0961 sameOriginWithAncestors), and defines its own implementation of  
0962 [[Create]](origin, options, sameOriginWithAncestors),  
0963 [[DiscoverFromExternalSource]](origin, options,  
0964 sameOriginWithAncestors), and [[Store]](credential,  
0965 sameOriginWithAncestors).  
0966  
0967 5.1.1. CredentialCreationOptions Dictionary Extension  
0968  
0969 To support registration via navigator.credentials.create(), this  
0970 document extends the CredentialCreationOptions dictionary as follows:  
0971 partial dictionary CredentialCreationOptions {  
0972 PublicKeyCredentialCreationOptions publicKey;  
0973 };  
0974  
0975 5.1.2. CredentialRequestOptions Dictionary Extension  
0976  
0977 To support obtaining assertions via navigator.credentials.get(), this  
0978 document extends the CredentialRequestOptions dictionary as follows:  
0979 partial dictionary CredentialRequestOptions {

```

0980 PublicKeyCredentialRequestOptions publicKey;
0981 };
0982
0983 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin,
0984 options, sameOriginWithAncestors) method
0985
0986 PublicKeyCredential's interface object's implementation of the
0987 [[Create]](origin, options, sameOriginWithAncestors) internal method
0988 [CREDENTIAL-MANAGEMENT-1] allows Relying Party scripts to call
0989 navigator.credentials.create() to request the creation of a new public
0990 key credential source, bound to an authenticator. This
0991 navigator.credentials.create() operation can be aborted by leveraging
0992 the AbortController; see DOM 3.3 Using AbortController and AbortSignal
0993 objects in APIs for detailed instructions.
0994
0995 This internal method accepts three arguments:
0996
0997 origin
0998 This argument is the relevant settings object's origin, as
0999 determined by the calling create() implementation.
1000
1001 options
1002 This argument is a CredentialCreationOptions object whose
1003 options.publicKey member contains a
1004 PublicKeyCredentialCreationOptions object specifying the desired
1005 attributes of the to-be-created public key credential.
1006
1007 sameOriginWithAncestors
1008 This argument is a boolean which is true if and only if the
1009 caller's environment settings object is same-origin with its
1010 ancestors.
1011
1012 Note: This algorithm is synchronous: the Promise resolution/rejection
1013 is handled by navigator.credentials.create().
1014
1015 When this method is invoked, the user agent MUST execute the following
1016 algorithm:
1017 1. Assert: options.publicKey is present.
1018 2. If sameOriginWithAncestors is false, return a "NotAllowedError"
1019 DOMException.
1020 Note: This "sameOriginWithAncestors" restriction aims to address
1021 the concern raised in the Origin Confusion section of
1022 [CREDENTIAL-MANAGEMENT-1], while allowing Relying Party script
1023 access to Web Authentication functionality, e.g., when running in a
1024 secure context framed document that is same-origin with its
1025 ancestors. However, in the future, this specification (in
1026 conjunction with [CREDENTIAL-MANAGEMENT-1]) may provide Relying
1027 Parties with more fine-grained control--e.g., ranging from allowing
1028 only top-level access to Web Authentication functionality, to
1029 allowing cross-origin embedded cases--by leveraging
1030 [Feature-Policy] once the latter specification becomes stably
1031 implemented in user agents.
1032 3. Let options be the value of options.publicKey.
1033 4. If the timeout member of options is present, check if its value
1034 lies within a reasonable range as defined by the platform and if
1035 not, correct it to the closest value lying within that range. Set a
1036 timer lifetimeTimer to this adjusted value. If the timeout member
1037 of options is not present, then set lifetimeTimer to a
1038 platform-specific default.
1039 5. Let callerOrigin be origin. If callerOrigin is an opaque origin,
1040 return a DOMException whose name is "NotAllowedError", and
1041 terminate this algorithm.
1042 6. Let effectiveDomain be the callerOrigin's effective domain. If
1043 effective domain is not a valid domain, then return a DOMException
1044 whose name is "SecurityError" and terminate this algorithm.
1045 Note: An effective domain may resolve to a host, which can be
1046 represented in various manners, such as domain, ipv4 address, ipv6
1047 address, opaque host, or empty host. Only the domain format of host
1048 is allowed here.
1049 7. If options.rp.id

```

```

0980 PublicKeyCredentialRequestOptions publicKey;
0981 };
0982
0983 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin,
0984 options, sameOriginWithAncestors) method
0985
0986 PublicKeyCredential's interface object's implementation of the
0987 [[Create]](origin, options, sameOriginWithAncestors) internal method
0988 [CREDENTIAL-MANAGEMENT-1] allows Relying Party scripts to call
0989 navigator.credentials.create() to request the creation of a new public
0990 key credential source, bound to an authenticator. This
0991 navigator.credentials.create() operation can be aborted by leveraging
0992 the AbortController; see DOM 3.3 Using AbortController and AbortSignal
0993 objects in APIs for detailed instructions.
0994
0995 This internal method accepts three arguments:
0996
0997 origin
0998 This argument is the relevant settings object's origin, as
0999 determined by the calling create() implementation.
1000
1001 options
1002 This argument is a CredentialCreationOptions object whose
1003 options.publicKey member contains a
1004 PublicKeyCredentialCreationOptions object specifying the desired
1005 attributes of the to-be-created public key credential.
1006
1007 sameOriginWithAncestors
1008 This argument is a boolean which is true if and only if the
1009 caller's environment settings object is same-origin with its
1010 ancestors.
1011
1012 Note: This algorithm is synchronous: the Promise resolution/rejection
1013 is handled by navigator.credentials.create().
1014
1015 When this method is invoked, the user agent MUST execute the following
1016 algorithm:
1017 1. Assert: options.publicKey is present.
1018 2. If sameOriginWithAncestors is false, return a "NotAllowedError"
1019 DOMException.
1020 Note: This "sameOriginWithAncestors" restriction aims to address
1021 the concern raised in the Origin Confusion section of
1022 [CREDENTIAL-MANAGEMENT-1], while allowing Relying Party script
1023 access to Web Authentication functionality, e.g., when running in a
1024 secure context framed document that is same-origin with its
1025 ancestors. However, in the future, this specification (in
1026 conjunction with [CREDENTIAL-MANAGEMENT-1]) may provide Relying
1027 Parties with more fine-grained control--e.g., ranging from allowing
1028 only top-level access to Web Authentication functionality, to
1029 allowing cross-origin embedded cases--by leveraging
1030 [Feature-Policy] once the latter specification becomes stably
1031 implemented in user agents.
1032 3. Let options be the value of options.publicKey.
1033 4. If the timeout member of options is present, check if its value
1034 lies within a reasonable range as defined by the platform and if
1035 not, correct it to the closest value lying within that range. Set a
1036 timer lifetimeTimer to this adjusted value. If the timeout member
1037 of options is not present, then set lifetimeTimer to a
1038 platform-specific default.
1039 5. Let callerOrigin be origin. If callerOrigin is an opaque origin,
1040 return a DOMException whose name is "NotAllowedError", and
1041 terminate this algorithm.
1042 6. Let effectiveDomain be the callerOrigin's effective domain. If
1043 effective domain is not a valid domain, then return a DOMException
1044 whose name is "SecurityError" and terminate this algorithm.
1045 Note: An effective domain may resolve to a host, which can be
1046 represented in various manners, such as domain, ipv4 address, ipv6
1047 address, opaque host, or empty host. Only the domain format of host
1048 is allowed here.
1049 7. If options.rp.id

```

1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119

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

Is not present  
Set options.rp.id to effectiveDomain.

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

8. Let credTypesAndPubKeyAlgs be a new list whose items are pairs of PublicKeyCredentialType and a COSEAlgorithmIdentifier.
9. For each current of options.pubKeyCredParams:
  1. If current.type does not contain a PublicKeyCredentialType supported by this implementation, then continue.
  2. Let alg be current.alg.
  3. Append the pair of current.type and alg to credTypesAndPubKeyAlgs.
10. If credTypesAndPubKeyAlgs is empty and options.pubKeyCredParams is not empty, return a DOMException whose name is "NotSupportedError", and terminate this algorithm.
11. Let clientExtensions be a new map and let authenticatorExtensions be a new map.
12. If the extensions member of options is present, then for each extensionId -> clientExtensionInput of options.extensions:
  1. If extensionId is not supported by this client platform or is not a registration extension, then continue.
  2. Set clientExtensions[extensionId] to clientExtensionInput.
  3. If extensionId is not an authenticator extension, then continue.
  4. Let authenticatorExtensionInput be the (CBOR) result of running extensionId's client extension processing algorithm on clientExtensionInput. If the algorithm returned an error, continue.
  5. Set authenticatorExtensions[extensionId] to the base64url encoding of authenticatorExtensionInput.
13. Let collectedClientData be a new CollectedClientData instance whose fields are:
  - type  
The string "webauthn.create".
  - challenge  
The base64url encoding of options.challenge.
  - origin  
The serialization of callerOrigin.
  - tokenBinding  
The status of Token Binding between the client and the callerOrigin, as well as the Token Binding ID associated with callerOrigin, if one is available.
14. Let clientDataJSON be the JSON-serialized client data constructed from collectedClientData.
15. Let clientDataHash be the hash of the serialized client data represented by clientDataJSON.
16. If the options.signal is present and its aborted flag is set to true, return a DOMException whose name is "AbortError" and terminate this algorithm.
17. **Start lifetimeTimer.**
18. **Let issuedRequests be a new ordered set.**
19. **For each authenticator that becomes available on this platform during the lifetime of lifetimeTimer, do the following:**  
The definitions of "lifetime of" and "becomes available" are intended to represent how devices are hot-plugged into (USB) or discovered by (NFC) browsers, and are underspecified. Resolving

1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119

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

Is not present  
Set options.rp.id to effectiveDomain.

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

8. Let credTypesAndPubKeyAlgs be a new list whose items are pairs of PublicKeyCredentialType and a COSEAlgorithmIdentifier.
9. For each current of options.pubKeyCredParams:
  1. If current.type does not contain a PublicKeyCredentialType supported by this implementation, then continue.
  2. Let alg be current.alg.
  3. Append the pair of current.type and alg to credTypesAndPubKeyAlgs.
10. If credTypesAndPubKeyAlgs is empty and options.pubKeyCredParams is not empty, return a DOMException whose name is "NotSupportedError", and terminate this algorithm.
11. Let clientExtensions be a new map and let authenticatorExtensions be a new map.
12. If the extensions member of options is present, then for each extensionId -> clientExtensionInput of options.extensions:
  1. If extensionId is not supported by this client platform or is not a registration extension, then continue.
  2. Set clientExtensions[extensionId] to clientExtensionInput.
  3. If extensionId is not an authenticator extension, then continue.
  4. Let authenticatorExtensionInput be the (CBOR) result of running extensionId's client extension processing algorithm on clientExtensionInput. If the algorithm returned an error, continue.
  5. Set authenticatorExtensions[extensionId] to the base64url encoding of authenticatorExtensionInput.
13. Let collectedClientData be a new CollectedClientData instance whose fields are:
  - type  
The string "webauthn.create".
  - challenge  
The base64url encoding of options.challenge.
  - origin  
The serialization of callerOrigin.
  - tokenBinding  
The status of Token Binding between the client and the callerOrigin, as well as the Token Binding ID associated with callerOrigin, if one is available.
14. Let clientDataJSON be the JSON-serialized client data constructed from collectedClientData.
15. Let clientDataHash be the hash of the serialized client data represented by clientDataJSON.
16. If the options.signal is present and its aborted flag is set to true, return a DOMException whose name is "AbortError" and terminate this algorithm.
17. **Let issuedRequests be a new ordered set.**
18. **Let authenticators represent a set of platform-specific handles, where each value identifies an authenticator presently available on this platform at a given instant.**  
Note: What qualifies an authenticator as "available" is intentionally unspecified; this is meant to represent how authenticators can be hot-plugged into (e.g., via USB) or



1120 this with good definitions or some other means will be addressed by  
1121 resolving Issue #613.

- 1122 1. If options.authenticatorSelection is present:
- 1123 1. If options.authenticatorSelection.authenticatorAttachment
- 1124 is present and its value is not equal to authenticator's
- 1125 attachment modality, continue.
- 1126 2. If options.authenticatorSelection.requireResidentKey is
- 1127 set to true and the authenticator is not capable of
- 1128 storing a Client-Side-Resident Credential Private Key,
- 1129 continue.
- 1130 3. If options.authenticatorSelection.userVerification is set
- 1131 to required and the authenticator is not capable of
- 1132 performing user verification, continue.
- 1133 2. Let userVerification be the effective user verification
- 1134 requirement for credential creation, a Boolean value, as
- 1135 follows. If options.authenticatorSelection.userVerification
- 1136 is set to required
- 1137 Let userVerification be true.
- 1138
- 1139 is set to preferred
- 1140 If the authenticator
- 1141
- 1142 is capable of user verification
- 1143 Let userVerification be true.
- 1144
- 1145 is not capable of user verification
- 1146 Let userVerification be false.
- 1147
- 1148 is set to discouraged
- 1149 Let userVerification be false.
- 1150
- 1151
- 1152 3. Let userPresence be a Boolean value set to the inverse of
- 1153 userVerification.
- 1154 4. Let excludeCredentialDescriptorList be a new list.
- 1155 5. For each credential descriptor C in
- 1156 options.excludeCredentials:
- 1157 1. If C.transports is not empty, and authenticator is
- 1158 connected over a transport not mentioned in C.transports,
- 1159 the client MAY continue.
- 1160 2. Otherwise, Append C to excludeCredentialDescriptorList.
- 1161
- 1162 6. Invoke the authenticatorMakeCredential operation on
- 1163 authenticator with clientDataHash, options.rp, options.user,
- 1164 options.authenticatorSelection.requireResidentKey,
- 1165 userPresence, userVerification, credTypesAndPubKeyAlgs,
- 1166 excludeCredentialDescriptorList, and authenticatorExtensions

1120 discovered (e.g., via NFC or Bluetooth) by the client by various  
1121 mechanisms.  
1122 19. Start lifetimeTimer.  
1123 20. While lifetimeTimer has not expired, perform the following actions  
1124 depending upon lifetimeTimer and the state and response for each  
1125 authenticator in authenticators:

1126  
1127 If lifetimeTimer expires,  
1128 For each authenticator in issuedRequests invoke the  
1129 authenticatorCancel operation on authenticator and remove  
1130 authenticator from issuedRequests.

1131  
1132 If the options.signal is present and its aborted flag is set to  
1133 true,  
1134 For each authenticator in issuedRequests invoke the  
1135 authenticatorCancel operation on authenticator and remove  
1136 authenticator from issuedRequests. Then return a  
1137 DOMException whose name is "AbortError" and terminate this  
1138 algorithm.

1139  
1140 If an authenticator becomes available on this platform,

- 1141 1. If options.authenticatorSelection is present:
- 1142 1. If
- 1143 options.authenticatorSelection.authenticatorAttachme
- 1144 nt is present and its value is not equal to
- 1145 authenticator's attachment modality, continue.
- 1146 2. If options.authenticatorSelection.requireResidentKey
- 1147 is set to true and the authenticator is not capable
- 1148 of storing a Client-Side-Resident Credential Private
- 1149 Key, continue.
- 1150 3. If options.authenticatorSelection.userVerification
- 1151 is set to required and the authenticator is not
- 1152 capable of performing user verification, continue.
- 1153 2. Let userVerification be the effective user verification
- 1154 requirement for credential creation, a Boolean value, as
- 1155 follows. If
- 1156 options.authenticatorSelection.userVerification
- 1157 is set to required
- 1158 Let userVerification be true.
- 1159
- 1160 is set to preferred
- 1161 If the authenticator
- 1162
- 1163 is capable of user verification
- 1164 Let userVerification be true.
- 1165
- 1166 is not capable of user verification
- 1167 Let userVerification be false.
- 1168
- 1169 is set to discouraged
- 1170 Let userVerification be false.
- 1171
- 1172
- 1173 3. Let userPresence be a Boolean value set to the inverse of
- 1174 userVerification.
- 1175 4. Let excludeCredentialDescriptorList be a new list.
- 1176 5. For each credential descriptor C in
- 1177 options.excludeCredentials:
- 1178 1. If C.transports is not empty, and authenticator is
- 1179 connected over a transport not mentioned in
- 1180 C.transports, the client MAY continue.
- 1181 2. Otherwise, Append C to
- 1182 excludeCredentialDescriptorList.
- 1183 6. Invoke the authenticatorMakeCredential operation on
- 1184 authenticator with clientDataHash, options.rp,
- 1185 options.user,
- 1186 options.authenticatorSelection.requireResidentKey,
- 1187 userPresence, userVerification, credTypesAndPubKeyAlgs,
- 1188 excludeCredentialDescriptorList, and
- 1189

1166 as parameters.  
 1167 7. Append authenticator to issuedRequests.  
 1168 20. While lifetimeTimer has not expired, perform the following actions  
 1169 depending upon lifetimeTimer and responses from the authenticators:  
 1170  
 1171 If lifetimeTimer expires,  
 1172 For each authenticator in issuedRequests invoke the  
 1173 authenticatorCancel operation on authenticator and remove  
 1174 authenticator from issuedRequests.  
 1175  
 1176 If the options.signal is present and its aborted flag is set to  
 1177 true,  
 1178 For each authenticator in issuedRequests invoke the  
 1179 authenticatorCancel operation on authenticator and remove  
 1180 authenticator from issuedRequests. Then return a  
 1181 DOMException whose name is "AbortError" and terminate this  
 1182 algorithm.  
 1183  
 1184 If any authenticator returns a status indicating that the user  
 1185 cancelled the operation,  
 1186  
 1187 1. Remove authenticator from issuedRequests.  
 1188 2. For each remaining authenticator in issuedRequests invoke  
 1189 the authenticatorCancel operation on authenticator and  
 1190 remove it from issuedRequests.  
 1191 Note: Authenticators may return an indication of "the  
 1192 user cancelled the entire operation". How a user agent  
 1193 manifests this state to users is unspecified.  
 1194  
 1195 If any authenticator returns an error status equivalent to  
 1196 "InvalidStateError",  
 1197  
 1198 1. Remove authenticator from issuedRequests.  
 1199 2. For each remaining authenticator in issuedRequests invoke  
 1200 the authenticatorCancel operation on authenticator and  
 1201 remove it from issuedRequests.  
 1202 3. Return a DOMException whose name is "InvalidStateError"  
 1203 and terminate this algorithm.  
 1204  
 1205 Note: This error status is handled separately because the  
 1206 authenticator returns it only if  
 1207 excludeCredentialDescriptorList identifies a credential  
 1208 bound to the authenticator and the user has consented to  
 1209 the operation. Given this explicit consent, it is  
 1210 acceptable for this case to be distinguishable to the  
 1211 Relying Party.  
 1212  
 1213 If any authenticator returns an error status not equivalent to  
 1214 "InvalidStateError",  
 1215 Remove authenticator from issuedRequests.  
 1216  
 1217 Note: This case does not imply user consent for the  
 1218 operation, so details about the error must be hidden from  
 1219 the Relying Party in order to prevent leak of potentially  
 1220 identifying information. See 14.2 Registration Ceremony  
 1221 Privacy for details.  
 1222  
 1223 If any authenticator indicates success,  
 1224  
 1225 1. Remove authenticator from issuedRequests.  
 1226 2. Let credentialCreationData be a struct whose items are:  
 1227  
 1228 attestationObjectResult  
 1229 whose value is the bytes returned from the  
 1230 successful authenticatorMakeCredential  
 1231 operation.  
 1232  
 1233 Note: this value is attObj, as defined in  
 1234 6.3.4 Generating an Attestation Object.  
 1235

1190 authenticatorExtensions as parameters.  
 1191 7. Append authenticator to issuedRequests.  
 1192  
 1193 If an authenticator ceases to be available on this platform,  
 1194 Remove authenticator from issuedRequests.  
 1195  
 1196 If any authenticator returns a status indicating that the user  
 1197 cancelled the operation,  
 1198  
 1199 1. Remove authenticator from issuedRequests.  
 1200 2. For each remaining authenticator in issuedRequests invoke  
 1201 the authenticatorCancel operation on authenticator and  
 1202 remove it from issuedRequests.  
 1203 Note: Authenticators may return an indication of "the  
 1204 user cancelled the entire operation". How a user agent  
 1205 manifests this state to users is unspecified.  
 1206  
 1207 If any authenticator returns an error status equivalent to  
 1208 "InvalidStateError",  
 1209  
 1210 1. Remove authenticator from issuedRequests.  
 1211 2. For each remaining authenticator in issuedRequests invoke  
 1212 the authenticatorCancel operation on authenticator and  
 1213 remove it from issuedRequests.  
 1214 3. Return a DOMException whose name is "InvalidStateError"  
 1215 and terminate this algorithm.  
 1216  
 1217 Note: This error status is handled separately because the  
 1218 authenticator returns it only if  
 1219 excludeCredentialDescriptorList identifies a credential  
 1220 bound to the authenticator and the user has consented to  
 1221 the operation. Given this explicit consent, it is  
 1222 acceptable for this case to be distinguishable to the  
 1223 Relying Party.  
 1224  
 1225 If any authenticator returns an error status not equivalent to  
 1226 "InvalidStateError",  
 1227 Remove authenticator from issuedRequests.  
 1228  
 1229 Note: This case does not imply user consent for the  
 1230 operation, so details about the error must be hidden from  
 1231 the Relying Party in order to prevent leak of potentially  
 1232 identifying information. See 14.2 Registration Ceremony  
 1233 Privacy for details.  
 1234  
 1235 If any authenticator indicates success,  
 1236  
 1237 1. Remove authenticator from issuedRequests.  
 1238 2. Let credentialCreationData be a struct whose items are:  
 1239  
 1240 attestationObjectResult  
 1241 whose value is the bytes returned from the  
 1242 successful authenticatorMakeCredential  
 1243 operation.  
 1244  
 1245 Note: this value is attObj, as defined in  
 1246 6.3.4 Generating an Attestation Object.  
 1247

1236 clientDataJSONResult  
 1237 whose value is the bytes of clientDataJSON.  
 1238  
 1239 attestationConveyancePreferenceOption  
 1240 whose value is the value of  
 1241 options.attestation.  
 1242  
 1243 clientExtensionResults  
 1244 whose value is an  
 1245 AuthenticationExtensionsClientOutputs object  
 1246 containing extension identifier -> client  
 1247 extension output entries. The entries are  
 1248 created by running each extension's client  
 1249 extension processing algorithm to create the  
 1250 client extension outputs, for each client  
 1251 extension in clientDataJSON.clientExtensions.  
 1252  
 1253 3. Let constructCredentialAlg be an algorithm that takes a  
 1254 global object global, and whose steps are:  
 1255 1. If  
 1256 credentialCreationData.attestationConveyancePreferen  
 1257 ceOption's value is  
 1258 "none"  
 1259 Replace potentially uniquely  
 1260 identifying information with  
 1261 non-identifying versions of the  
 1262 same:  
 1263  
 1264 1. If the AAGUID in the attested  
 1265 credential data is 16 zero bytes,  
 1266 credentialCreationData.attestationObj  
 1267 ectResult.fmt is "packed", and "x5c"  
 1268 & "ecdaaKeyld" are both absent from  
 1269 credentialCreationData.attestationObj  
 1270 ectResult, then self attestation is  
 1271 being used and no further action is  
 1272 needed.  
 1273 2. Otherwise  
 1274 1. Replace the AAGUID in the attested  
 1275 credential data with 16 zero bytes.  
 1276 2. Set the value of  
 1277 credentialCreationData.attestationObj  
 1278 ectResult.fmt to "none", and set the  
 1279 value of  
 1280 credentialCreationData.attestationObj  
 1281 ectResult.attStmt to be an empty CBOR  
 1282 map. (See 8.7 None Attestation  
 1283 Statement Format and 6.3.4  
 1284 Generating an Attestation Object).  
 1285  
 1286 "indirect"  
 1287 The client MAY replace the AAGUID  
 1288 and attestation statement with a  
 1289 more privacy-friendly and/or more  
 1290 easily verifiable version of the  
 1291 same data (for example, by  
 1292 employing an Anonymization CA).  
 1293  
 1294 "direct"  
 1295 Convey the authenticator's AAGUID  
 1296 and attestation statement,  
 1297 unaltered, to the RP.  
 1298  
 1299 @balfanz wishes to add to the "direct"  
 1300 case: If the authenticator violates the  
 1301 privacy requirements of the attestation  
 1302 type it is using, the client SHOULD  
 1303 terminate this algorithm with an  
 1304 "AttestationNotPrivateError".  
 1305

1248 clientDataJSONResult  
 1249 whose value is the bytes of clientDataJSON.  
 1250  
 1251 attestationConveyancePreferenceOption  
 1252 whose value is the value of  
 1253 options.attestation.  
 1254  
 1255 clientExtensionResults  
 1256 whose value is an  
 1257 AuthenticationExtensionsClientOutputs object  
 1258 containing extension identifier -> client  
 1259 extension output entries. The entries are  
 1260 created by running each extension's client  
 1261 extension processing algorithm to create the  
 1262 client extension outputs, for each client  
 1263 extension in clientDataJSON.clientExtensions.  
 1264  
 1265 3. Let constructCredentialAlg be an algorithm that takes a  
 1266 global object global, and whose steps are:  
 1267 1. If  
 1268 credentialCreationData.attestationConveyancePreferen  
 1269 ceOption's value is  
 1270 "none"  
 1271 Replace potentially uniquely  
 1272 identifying information with  
 1273 non-identifying versions of the  
 1274 same:  
 1275  
 1276 1. If the AAGUID in the attested  
 1277 credential data is 16 zero bytes,  
 1278 credentialCreationData.attestationObj  
 1279 ectResult.fmt is "packed", and "x5c"  
 1280 & "ecdaaKeyld" are both absent from  
 1281 credentialCreationData.attestationObj  
 1282 ectResult, then self attestation is  
 1283 being used and no further action is  
 1284 needed.  
 1285 2. Otherwise  
 1286 1. Replace the AAGUID in the attested  
 1287 credential data with 16 zero bytes.  
 1288 2. Set the value of  
 1289 credentialCreationData.attestationObj  
 1290 ectResult.fmt to "none", and set the  
 1291 value of  
 1292 credentialCreationData.attestationObj  
 1293 ectResult.attStmt to be an empty CBOR  
 1294 map. (See 8.7 None Attestation  
 1295 Statement Format and 6.3.4  
 1296 Generating an Attestation Object).  
 1297  
 1298 "indirect"  
 1299 The client MAY replace the AAGUID  
 1300 and attestation statement with a  
 1301 more privacy-friendly and/or more  
 1302 easily verifiable version of the  
 1303 same data (for example, by  
 1304 employing an Anonymization CA).  
 1305  
 1306 "direct"  
 1307 Convey the authenticator's AAGUID  
 1308 and attestation statement,  
 1309 unaltered, to the RP.  
 1310  
 1311 @balfanz wishes to add to the "direct"  
 1312 case: If the authenticator violates the  
 1313 privacy requirements of the attestation  
 1314 type it is using, the client SHOULD  
 1315 terminate this algorithm with an  
 1316 "AttestationNotPrivateError".  
 1317

1306  
1307 2. Let attestationObject be a new ArrayBuffer, created  
1308 using global's %ArrayBuffer%, containing the bytes  
1309 of credentialCreationData.attestationObjectResult's  
1310 value.  
1311 3. Let id be  
1312 attestationObject.authData.attestedCredentialData.cred  
1313 entialId.  
1314 4. Let publicKeyCred be a new PublicKeyCredential object  
1315 associated with global whose fields are:  
1316  
1317     [[identifier]]  
1318         id  
1319  
1320     response  
1321         A new AuthenticatorAttestationResponse  
1322         object associated with global whose  
1323         fields are:  
1324  
1325             clientDataJSON  
1326                 A new ArrayBuffer, created using  
1327                 global's %ArrayBuffer%, containing  
1328                 the bytes of  
1329                 credentialCreationData.clientDataJ  
1330                 SONResult.  
1331  
1332             attestationObject  
1333                 attestationObject  
1334  
1335             [[clientExtensionsResults]]  
1336                 A new ArrayBuffer, created using  
1337                 global's %ArrayBuffer%, containing the  
1338                 bytes of  
1339                 credentialCreationData.clientExtensionRe  
1340                 sults.  
1341  
1342     5. Return publicKeyCred.  
1343 4. For each remaining authenticator in issuedRequests invoke  
1344 the authenticatorCancel operation on authenticator and  
1345 remove it from issuedRequests.  
1346 5. Return constructCredentialAlg and terminate this  
1347 algorithm.  
1348  
1349 21. Return a DOMException whose name is "NotAllowedError". In order to  
1350 prevent information leak that could identify the user without  
1351 consent, this step MUST NOT be executed before lifetimeTimer has  
1352 expired. See 14.3 Authentication Ceremony Privacy for details.  
1353  
1354 During the above process, the user agent SHOULD show some UI to the  
1355 user to guide them in the process of selecting and authorizing an  
1356 authenticator.  
1357  
1358 5.1.4. Use an existing credential to make an assertion -  
1359 PublicKeyCredential's [[Get]](options) method  
1360  
1361 Relying Parties call navigator.credentials.get({publicKey:..., ...}) to  
1362 discover and use an existing public key credential, with the user's  
1363 consent. Relying Party script optionally specifies some criteria to  
1364 indicate what credential sources are acceptable to it. The user agent  
1365 and/or platform locates credential sources matching the specified  
1366 criteria, and guides the user to pick one that the script will be  
1367 allowed to use. The user may choose to decline the entire interaction  
1368 even if a credential source is present, for example to maintain  
1369 privacy. If the user picks a credential source, the user agent then  
1370 uses 6.2.3 The authenticatorGetAssertion operation to sign a Relying  
1371 Party-provided challenge and other collected data into an assertion,  
1372 which is used as a credential.  
1373  
1374 The get() implementation [CREDENTIAL-MANAGEMENT-1] calls  
1375 PublicKeyCredential.[[CollectFromCredentialStore]]() to collect any

1318  
1319 2. Let attestationObject be a new ArrayBuffer, created  
1320 using global's %ArrayBuffer%, containing the bytes  
1321 of credentialCreationData.attestationObjectResult's  
1322 value.  
1323 3. Let id be  
1324 attestationObject.authData.attestedCredentialData.cred  
1325 entialId.  
1326 4. Let publicKeyCred be a new PublicKeyCredential object  
1327 associated with global whose fields are:  
1328  
1329     [[identifier]]  
1330         id  
1331  
1332     response  
1333         A new AuthenticatorAttestationResponse  
1334         object associated with global whose  
1335         fields are:  
1336  
1337             clientDataJSON  
1338                 A new ArrayBuffer, created using  
1339                 global's %ArrayBuffer%, containing  
1340                 the bytes of  
1341                 credentialCreationData.clientDataJ  
1342                 SONResult.  
1343  
1344             attestationObject  
1345                 attestationObject  
1346  
1347             [[clientExtensionsResults]]  
1348                 A new ArrayBuffer, created using  
1349                 global's %ArrayBuffer%, containing the  
1350                 bytes of  
1351                 credentialCreationData.clientExtensionRe  
1352                 sults.  
1353  
1354     5. Return publicKeyCred.  
1355 4. For each remaining authenticator in issuedRequests invoke  
1356 the authenticatorCancel operation on authenticator and  
1357 remove it from issuedRequests.  
1358 5. Return constructCredentialAlg and terminate this  
1359 algorithm.  
1360  
1361 21. Return a DOMException whose name is "NotAllowedError". In order to  
1362 prevent information leak that could identify the user without  
1363 consent, this step MUST NOT be executed before lifetimeTimer has  
1364 expired. See 14.3 Authentication Ceremony Privacy for details.  
1365  
1366 During the above process, the user agent SHOULD show some UI to the  
1367 user to guide them in the process of selecting and authorizing an  
1368 authenticator.  
1369  
1370 5.1.4. Use an existing credential to make an assertion -  
1371 PublicKeyCredential's [[Get]](options) method  
1372  
1373 Relying Parties call navigator.credentials.get({publicKey:..., ...}) to  
1374 discover and use an existing public key credential, with the user's  
1375 consent. Relying Party script optionally specifies some criteria to  
1376 indicate what credential sources are acceptable to it. The user agent  
1377 and/or platform locates credential sources matching the specified  
1378 criteria, and guides the user to pick one that the script will be  
1379 allowed to use. The user may choose to decline the entire interaction  
1380 even if a credential source is present, for example to maintain  
1381 privacy. If the user picks a credential source, the user agent then  
1382 uses 6.2.3 The authenticatorGetAssertion operation to sign a Relying  
1383 Party-provided challenge and other collected data into an assertion,  
1384 which is used as a credential.  
1385  
1386 The get() implementation [CREDENTIAL-MANAGEMENT-1] calls  
1387 PublicKeyCredential.[[CollectFromCredentialStore]]() to collect any

1376 credentials that should be available without user mediation (roughly,  
 1377 this specification's authorization gesture), and if it does not find  
 1378 exactly one of those, it then calls  
 1379 PublicKeyCredential.[[DiscoverFromExternalSource]]() to have the user  
 1380 select a credential source.  
 1381  
 1382 Since this specification requires an authorization gesture to create  
 1383 any credentials, the  
 1384 PublicKeyCredential.[[CollectFromCredentialStore]](origin, options,  
 1385 sameOriginWithAncestors) internal method inherits the default behavior  
 1386 of Credential.[[CollectFromCredentialStore]](), of returning an empty  
 1387 set.  
 1388  
 1389 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin,  
 1390 options, sameOriginWithAncestors) method  
 1391  
 1392 This internal method accepts three arguments:  
 1393  
 1394 origin  
 1395 This argument is the relevant settings object's origin, as  
 1396 determined by the calling get() implementation, i.e.,  
 1397 CredentialsContainer's Request a Credential abstract operation.  
 1398  
 1399 options  
 1400 This argument is a CredentialRequestOptions object whose  
 1401 options.publicKey member contains a  
 1402 PublicKeyCredentialRequestOptions object specifying the desired  
 1403 attributes of the public key credential to discover.  
 1404  
 1405 sameOriginWithAncestors  
 1406 This argument is a boolean which is true if and only if the  
 1407 caller's environment settings object is same-origin with its  
 1408 ancestors.  
 1409  
 1410 Note: This algorithm is synchronous: the Promise resolution/rejection  
 1411 is handled by navigator.credentials.get().  
 1412  
 1413 When this method is invoked, the user agent MUST execute the following  
 1414 algorithm:  
 1415 1. Assert: options.publicKey is present.  
 1416 2. If sameOriginWithAncestors is false, return a "NotAllowedError"  
 1417 DOMException.  
 1418 Note: This "sameOriginWithAncestors" restriction aims to address  
 1419 the concern raised in the Origin Confusion section of  
 1420 [CREDENTIAL-MANAGEMENT-1], while allowing Relying Party script  
 1421 access to Web Authentication functionality, e.g., when running in a  
 1422 secure context framed document that is same-origin with its  
 1423 ancestors. However, in the future, this specification (in  
 1424 conjunction with [CREDENTIAL-MANAGEMENT-1]) may provide Relying  
 1425 Parties with more fine-grained control--e.g., ranging from allowing  
 1426 only top-level access to Web Authentication functionality, to  
 1427 allowing cross-origin embedded cases--by leveraging  
 1428 [Feature-Policy] once the latter specification becomes stably  
 1429 implemented in user agents.  
 1430 3. Let options be the value of options.publicKey.  
 1431 4. If the timeout member of options is present, check if its value  
 1432 lies within a reasonable range as defined by the platform and if  
 1433 not, correct it to the closest value lying within that range. Set a  
 1434 timer lifetimeTimer to this adjusted value. If the timeout member  
 1435 of options is not present, then set lifetimeTimer to a  
 1436 platform-specific default.  
 1437 5. Let callerOrigin be origin. If callerOrigin is an opaque origin,  
 1438 return a DOMException whose name is "NotAllowedError", and  
 1439 terminate this algorithm.  
 1440 6. Let effectiveDomain be the callerOrigin's effective domain. If  
 1441 effective domain is not a valid domain, then return a DOMException  
 1442 whose name is "SecurityError" and terminate this algorithm.  
 1443 Note: An effective domain may resolve to a host, which can be  
 1444 represented in various manners, such as domain, ipv4 address, ipv6  
 1445 address, opaque host, or empty host. Only the domain format of host

1388 credentials that should be available without user mediation (roughly,  
 1389 this specification's authorization gesture), and if it does not find  
 1390 exactly one of those, it then calls  
 1391 PublicKeyCredential.[[DiscoverFromExternalSource]]() to have the user  
 1392 select a credential source.  
 1393  
 1394 Since this specification requires an authorization gesture to create  
 1395 any credentials, the  
 1396 PublicKeyCredential.[[CollectFromCredentialStore]](origin, options,  
 1397 sameOriginWithAncestors) internal method inherits the default behavior  
 1398 of Credential.[[CollectFromCredentialStore]](), of returning an empty  
 1399 set.  
 1400  
 1401 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin,  
 1402 options, sameOriginWithAncestors) method  
 1403  
 1404 This internal method accepts three arguments:  
 1405  
 1406 origin  
 1407 This argument is the relevant settings object's origin, as  
 1408 determined by the calling get() implementation, i.e.,  
 1409 CredentialsContainer's Request a Credential abstract operation.  
 1410  
 1411 options  
 1412 This argument is a CredentialRequestOptions object whose  
 1413 options.publicKey member contains a  
 1414 PublicKeyCredentialRequestOptions object specifying the desired  
 1415 attributes of the public key credential to discover.  
 1416  
 1417 sameOriginWithAncestors  
 1418 This argument is a boolean which is true if and only if the  
 1419 caller's environment settings object is same-origin with its  
 1420 ancestors.  
 1421  
 1422 Note: This algorithm is synchronous: the Promise resolution/rejection  
 1423 is handled by navigator.credentials.get().  
 1424  
 1425 When this method is invoked, the user agent MUST execute the following  
 1426 algorithm:  
 1427 1. Assert: options.publicKey is present.  
 1428 2. If sameOriginWithAncestors is false, return a "NotAllowedError"  
 1429 DOMException.  
 1430 Note: This "sameOriginWithAncestors" restriction aims to address  
 1431 the concern raised in the Origin Confusion section of  
 1432 [CREDENTIAL-MANAGEMENT-1], while allowing Relying Party script  
 1433 access to Web Authentication functionality, e.g., when running in a  
 1434 secure context framed document that is same-origin with its  
 1435 ancestors. However, in the future, this specification (in  
 1436 conjunction with [CREDENTIAL-MANAGEMENT-1]) may provide Relying  
 1437 Parties with more fine-grained control--e.g., ranging from allowing  
 1438 only top-level access to Web Authentication functionality, to  
 1439 allowing cross-origin embedded cases--by leveraging  
 1440 [Feature-Policy] once the latter specification becomes stably  
 1441 implemented in user agents.  
 1442 3. Let options be the value of options.publicKey.  
 1443 4. If the timeout member of options is present, check if its value  
 1444 lies within a reasonable range as defined by the platform and if  
 1445 not, correct it to the closest value lying within that range. Set a  
 1446 timer lifetimeTimer to this adjusted value. If the timeout member  
 1447 of options is not present, then set lifetimeTimer to a  
 1448 platform-specific default.  
 1449 5. Let callerOrigin be origin. If callerOrigin is an opaque origin,  
 1450 return a DOMException whose name is "NotAllowedError", and  
 1451 terminate this algorithm.  
 1452 6. Let effectiveDomain be the callerOrigin's effective domain. If  
 1453 effective domain is not a valid domain, then return a DOMException  
 1454 whose name is "SecurityError" and terminate this algorithm.  
 1455 Note: An effective domain may resolve to a host, which can be  
 1456 represented in various manners, such as domain, ipv4 address, ipv6  
 1457 address, opaque host, or empty host. Only the domain format of host

1446 is allowed here.  
 1447 7. If options.rpld is not present, then set rpld to effectiveDomain.  
 1448 Otherwise:  
 1449 1. If options.rpld is not a registrable domain suffix of and is  
 1450 not equal to effectiveDomain, return a DOMException whose name  
 1451 is "SecurityError", and terminate this algorithm.  
 1452 2. Set rpld to options.rpld.  
 1453 Note: rpld represents the caller's RP ID. The RP ID defaults  
 1454 to being the caller's origin's effective domain unless the  
 1455 caller has explicitly set options.rpld when calling get().  
 1456 8. Let clientExtensions be a new map and let authenticatorExtensions  
 1457 be a new map.  
 1458 9. If the extensions member of options is present, then for each  
 1459 extensionId -> clientExtensionInput of options.extensions:  
 1460 1. If extensionId is not supported by this client platform or is  
 1461 not an authentication extension, then continue.  
 1462 2. Set clientExtensions[extensionId] to clientExtensionInput.  
 1463 3. If extensionId is not an authenticator extension, then  
 1464 continue.  
 1465 4. Let authenticatorExtensionInput be the (CBOR) result of  
 1466 running extensionId's client extension processing algorithm on  
 1467 clientExtensionInput. If the algorithm returned an error,  
 1468 continue.  
 1469 5. Set authenticatorExtensions[extensionId] to the base64url  
 1470 encoding of authenticatorExtensionInput.  
 1471 10. Let collectedClientData be a new CollectedClientData instance whose  
 1472 fields are:  
 1473 type  
 1474 The string "webauthn.get".  
 1475 challenge  
 1476 The base64url encoding of options.challenge  
 1477 origin  
 1478 The serialization of callerOrigin.  
 1479 tokenBinding  
 1480 The status of Token Binding between the client and the  
 1481 callerOrigin, as well as the Token Binding ID associated  
 1482 with callerOrigin, if one is available.  
 1483 11. Let clientDataJSON be the JSON-serialized client data constructed  
 1484 from collectedClientData.  
 1485 12. Let clientDataHash be the hash of the serialized client data  
 1486 represented by clientDataJSON.  
 1487 13. If the options.signal is present and its aborted flag is set to  
 1488 true, return a DOMException whose name is "AbortError" and  
 1489 terminate this algorithm.  
 1490 14. Let issuedRequests be a new ordered set.  
 1491 15. Let authenticator be a platform-specific handle whose value  
 1492 identifies an authenticator.  
 1493 16. Start lifetimeTimer.  
 1494 17. For each authenticator that becomes available on this platform  
 1495 during the lifetime of lifetimeTimer, perform the following steps:  
 1496 The definitions of "lifetime of" and "becomes available" are  
 1497 intended to represent how devices are hot-plugged into (USB) or  
 1498 discovered by (NFC) browsers, and are underspecified. Resolving  
 1499 this with good definitions or some other means will be addressed by  
 1500 resolving Issue #613.  
 1501  
 1502  
 1503  
 1504  
 1505

1458 is allowed here.  
 1459 7. If options.rpld is not present, then set rpld to effectiveDomain.  
 1460 Otherwise:  
 1461 1. If options.rpld is not a registrable domain suffix of and is  
 1462 not equal to effectiveDomain, return a DOMException whose name  
 1463 is "SecurityError", and terminate this algorithm.  
 1464 2. Set rpld to options.rpld.  
 1465 Note: rpld represents the caller's RP ID. The RP ID defaults  
 1466 to being the caller's origin's effective domain unless the  
 1467 caller has explicitly set options.rpld when calling get().  
 1468 8. Let clientExtensions be a new map and let authenticatorExtensions  
 1469 be a new map.  
 1470 9. If the extensions member of options is present, then for each  
 1471 extensionId -> clientExtensionInput of options.extensions:  
 1472 1. If extensionId is not supported by this client platform or is  
 1473 not an authentication extension, then continue.  
 1474 2. Set clientExtensions[extensionId] to clientExtensionInput.  
 1475 3. If extensionId is not an authenticator extension, then  
 1476 continue.  
 1477 4. Let authenticatorExtensionInput be the (CBOR) result of  
 1478 running extensionId's client extension processing algorithm on  
 1479 clientExtensionInput. If the algorithm returned an error,  
 1480 continue.  
 1481 5. Set authenticatorExtensions[extensionId] to the base64url  
 1482 encoding of authenticatorExtensionInput.  
 1483 10. Let collectedClientData be a new CollectedClientData instance whose  
 1484 fields are:  
 1485 type  
 1486 The string "webauthn.get".  
 1487 challenge  
 1488 The base64url encoding of options.challenge  
 1489 origin  
 1490 The serialization of callerOrigin.  
 1491 tokenBinding  
 1492 The status of Token Binding between the client and the  
 1493 callerOrigin, as well as the Token Binding ID associated  
 1494 with callerOrigin, if one is available.  
 1495 11. Let clientDataJSON be the JSON-serialized client data constructed  
 1496 from collectedClientData.  
 1497 12. Let clientDataHash be the hash of the serialized client data  
 1498 represented by clientDataJSON.  
 1499 13. If the options.signal is present and its aborted flag is set to  
 1500 true, return a DOMException whose name is "AbortError" and  
 1501 terminate this algorithm.  
 1502 14. Let issuedRequests be a new ordered set.  
 1503 15. Let savedCredentialIDs be a new map.  
 1504 16. Let authenticators represent a set of platform-specific handles,  
 1505 where each value identifies an authenticator presently available on  
 1506 this platform at a given instant.  
 1507 Note: What qualifies an authenticator as "available" is  
 1508 intentionally unspecified; this is meant to represent how  
 1509 authenticators can be hot-plugged into (e.g., via USB) or  
 1510 discovered (e.g., via NFC or Bluetooth) by the client by various  
 1511 mechanisms.  
 1512 17. Start lifetimeTimer.  
 1513 18. While lifetimeTimer has not expired, perform the following actions  
 1514 depending upon lifetimeTimer and the state and response for each  
 1515 authenticator in authenticators:  
 1516  
 1517 If lifetimeTimer expires,  
 1518 For each authenticator in issuedRequests invoke the  
 1519 authenticatorCancel operation on authenticator and remove  
 1520 authenticator from issuedRequests.  
 1521  
 1522 If the user exercises a user-interface option to cancel the

1506 1. If options.userVerification is set to required and the  
1507 authenticator is not capable of performing user verification,  
1508 continue.  
1509 2. Let userVerification be the effective user verification  
1510 requirement for assertion, a Boolean value, as follows. If  
1511 options.userVerification  
1512 is set to required  
1513 Let userVerification be true.  
1514  
1515 is set to preferred  
1516 If the authenticator  
1517 is capable of user verification  
1518 Let userVerification be true.  
1519 is not capable of user verification  
1520 Let userVerification be false.  
1521  
1522 is set to discouraged  
1523 Let userVerification be false.  
1524  
1525 3. Let userPresence be a Boolean value set to the inverse of  
1526 userVerification.  
1527 4. If options.allowCredentials  
1528 is not empty  
1529  
1530 1. Let allowCredentialDescriptorList be a new list.  
1531 2. Execute a platform-specific procedure to determine  
1532 which, if any, public key credentials described by  
1533 options.allowCredentials are bound to this  
1534 authenticator, by matching with rpId,  
1535 options.allowCredentials.id, and  
1536 options.allowCredentials.type. Set  
1537 allowCredentialDescriptorList to this filtered list.  
1538  
1539 options.allowCredentials.id, and  
1540 options.allowCredentials.type. Set  
1541 allowCredentialDescriptorList to this filtered list.

1528 process,  
1529 For each authenticator in issuedRequests invoke the  
1530 authenticatorCancel operation on authenticator and remove  
1531 authenticator from issuedRequests. Return a DOMException  
1532 whose name is "NotAllowedError".  
1533  
1534 If the signal member is present and the aborted flag is set to  
1535 true,  
1536 For each authenticator in issuedRequests invoke the  
1537 authenticatorCancel operation on authenticator and remove  
1538 authenticator from issuedRequests. Then return a  
1539 DOMException whose name is "AbortError" and terminate this  
1540 algorithm.  
1541  
1542 If issuedRequests is empty, options.allowCredentials is not empty,  
1543 and no authenticator will become available for any public  
1544 key credentials therein,  
1545 Indicate to the user that the credential could not be  
1546 found. When the user acknowledges the dialog, or once  
1547 lifetimeTimer expires, return a DOMException whose name is  
1548 "NotAllowedError".  
1549  
1550 Note: One way in which the platform may determine that no  
1551 authenticator will become available is by using the  
1552 transports members of options.allowCredentials. For  
1553 example, if all credentials only list internal, but all  
1554 internal authenticators have been tried, then there is no  
1555 possibility of satisfying the request. Alternatively, all  
1556 credentials may require transports that the platform does  
1557 not support.  
1558  
1559 If an authenticator becomes available on this platform,  
1560  
1561 1. If options.userVerification is set to required and the  
1562 authenticator is not capable of performing user  
1563 verification, continue.  
1564 2. Let userVerification be the effective user verification  
1565 requirement for assertion, a Boolean value, as follows.  
1566 If options.userVerification  
1567 is set to required  
1568 Let userVerification be true.  
1569  
1570 is set to preferred  
1571 If the authenticator  
1572 is capable of user verification  
1573 Let userVerification be true.  
1574 is not capable of user verification  
1575 Let userVerification be false.  
1576  
1577 is set to discouraged  
1578 Let userVerification be false.  
1579  
1580 3. Let userPresence be a Boolean value set to the inverse of  
1581 userVerification.  
1582 4. If options.allowCredentials  
1583 is not empty  
1584  
1585 1. Let allowCredentialDescriptorList be a new  
1586 list.  
1587 2. Execute a platform-specific procedure to  
1588 determine which, if any, public key credentials  
1589 described by options.allowCredentials are bound  
1590 to this authenticator, by matching with rpId,  
1591 options.allowCredentials.id, and  
1592 options.allowCredentials.type. Set  
1593 allowCredentialDescriptorList to this filtered  
1594 list.  
1595 options.allowCredentials.id, and  
1596 options.allowCredentials.type. Set  
1597 allowCredentialDescriptorList to this filtered

1542 3. If allowCredentialDescriptorList is empty, continue.

1543 4. Let distinctTransports be a new ordered set.

1544 5. If allowCredentialDescriptorList has exactly one

1545 value, let savedCredentialId be a new

1546 PublicKeyCredentialDescriptor.id and set its value

1547 to allowCredentialDescriptorList[0].id's value (see

1548 here in 6.2.3 The authenticatorGetAssertion

1549 operation for more information).

1550

1551 The foregoing step may be incorrect, in that we

1552 are attempting to create savedCredentialId here and

1553 use it later below, and we do not have a global in

1554 which to allocate a place for it. Perhaps this is

1555 good enough? addendum: @jcjones feels the above step

1556 is likely good enough.

1557

1558 1. For each credential descriptor C in

1559 allowCredentialDescriptorList, append each value, if

1560 any, of C.transports to distinctTransports.

1561 Note: This will aggregate only distinct values of

1562 transports (for this authenticator) in

1563 distinctTransports due to the properties of ordered

1564 sets.

1565 2. If distinctTransports

1566 is not empty

1567 The client selects one transport value

1568 from distinctTransports, possibly

1569 incorporating local configuration

1570 knowledge of the appropriate transport

1571 to use with authenticator in making its

1572 selection.

1573

1574 Then, using transport, invoke the

1575 authenticatorGetAssertion operation on

1576 authenticator, with rpId,

1577 clientDataHash,

1578 allowCredentialDescriptorList,

1579 userPresence, userVerification, and

1580 authenticatorExtensions as parameters.

1581

1582 is empty

1583 Using local configuration knowledge of the

1584 appropriate transport to use with

1585 authenticator, invoke the

1586 authenticatorGetAssertion operation on

1587 authenticator with rpId, clientDataHash,

1588 allowCredentialDescriptorList,

1589 userPresence, userVerification, and

1590 clientExtensions as parameters.

1591

1592 is empty

1593 Using local configuration knowledge of the

1594 appropriate transport to use with authenticator,

1595 invoke the authenticatorGetAssertion operation on

1596 authenticator with rpId, clientDataHash,

1597 userPresence, userVerification and clientExtensions

1598 as parameters.

1599

1600 Note: In this case, the Relying Party did not supply

1601 a list of acceptable credential descriptors. Thus,

1602 the authenticator is being asked to exercise any

1603

1598 list.

1599 3. If allowCredentialDescriptorList is empty,

1600 continue.

1601 4. Let distinctTransports be a new ordered set.

1602 5. If allowCredentialDescriptorList has exactly

1603 one value, set

1604 savedCredentialIds[authenticator] to

1605 allowCredentialDescriptorList[0].id's value

1606 (see here in 6.2.3 The

1607 authenticatorGetAssertion operation for more

1608 information).

1609 6. For each credential descriptor C in

1610 allowCredentialDescriptorList, append each

1611 value, if any, of C.transports to

1612 distinctTransports.

1613 Note: This will aggregate only distinct values

1614 of transports (for this authenticator) in

1615 distinctTransports due to the properties of

1616 ordered sets.

1617 7. If distinctTransports

1618 is not empty

1619 The client selects one transport

1620 value from distinctTransports,

1621 possibly incorporating local

1622 configuration knowledge of the

1623 appropriate transport to use with

1624 authenticator in making its

1625 selection.

1626

1627 Then, using transport, invoke the

1628 authenticatorGetAssertion

1629 operation on authenticator, with

1630 rpId, clientDataHash,

1631 allowCredentialDescriptorList,

1632 userPresence, userVerification,

1633 and authenticatorExtensions as

1634 parameters.

1635

1636 is empty

1637 Using local configuration

1638 knowledge of the appropriate

1639 transport to use with

1640 authenticator, invoke the

1641 authenticatorGetAssertion

1642 operation on authenticator with

1643 rpId, clientDataHash,

1644 allowCredentialDescriptorList,

1645 userPresence, userVerification,

1646 and clientExtensions as

1647 parameters.

1648

1649 is empty

1650 Using local configuration knowledge of the

1651 appropriate transport to use with

1652 authenticator, invoke the

1653 authenticatorGetAssertion operation on

1654 authenticator with rpId, clientDataHash,

1655 userPresence, userVerification and

1656 clientExtensions as parameters.

1657

1658 Note: In this case, the Relying Party did not

1659 supply a list of acceptable credential

1660 descriptors. Thus, the authenticator is being

1661



1604 credential it may possess that is bound to the  
1605 Relying Party, as identified by rpId.

1606  
1607 5. Append authenticator to issuedRequests.

1608 18. While lifetimeTimer has not expired, perform the following actions  
1609 depending upon lifetimeTimer and responses from the authenticators:  
1610

1611 If lifetimeTimer expires,  
1612 For each authenticator in issuedRequests invoke the  
1613 authenticatorCancel operation on authenticator and remove  
1614 authenticator from issuedRequests.

1615

1616 If the signal member is present and the aborted flag is set to  
1617 true,  
1618 For each authenticator in issuedRequests invoke the  
1619 authenticatorCancel operation on authenticator and remove  
1620 authenticator from issuedRequests. Then return a  
1621 DOMException whose name is "AbortError" and terminate this  
1622 algorithm.

1623

1624 If any authenticator returns a status indicating that the user  
1625 cancelled the operation,  
1626

1627 1. Remove authenticator from issuedRequests.  
1628 2. For each remaining authenticator in issuedRequests invoke  
1629 the authenticatorCancel operation on authenticator and  
1630 remove it from issuedRequests.  
1631 Note: Authenticators may return an indication of "the  
1632 user cancelled the entire operation". How a user agent  
1633 manifests this state to users is unspecified.

1634

1635 If any authenticator returns an error status,  
1636 Remove authenticator from issuedRequests.

1637

1638 If any authenticator indicates success,  
1639

1640 1. Remove authenticator from issuedRequests.  
1641 2. Let assertionCreationData be a struct whose items are:

1642

1643 credentialIdResult  
1644 If savedCredentialId exists, set the value of  
1645 credentialIdResult to be the bytes of  
1646 savedCredentialId. Otherwise, set the value of  
1647 credentialIdResult to be the bytes of the  
1648 credential ID returned from the successful  
1649 authenticatorGetAssertion operation, as  
1650 defined in 6.2.3 The  
1651 authenticatorGetAssertion operation.

1652

1653 clientDataJSONResult  
1654 whose value is the bytes of clientDataJSON.

1655

1656 authenticatorDataResult  
1657 whose value is the bytes of the authenticator  
1658 data returned by the authenticator.

1659

1660 signatureResult  
1661 whose value is the bytes of the signature  
1662 value returned by the authenticator.

1663

1664 userHandleResult  
1665 If the authenticator returned a user handle,  
1666 set the value of userHandleResult to be the  
1667 bytes of the returned user handle. Otherwise,  
1668 set the value of userHandleResult to null.

1669

1670 clientExtensionResults  
1671 whose value is an  
1672 AuthenticationExtensionsClientOutputs object

1662 asked to exercise any credential it may  
1663 possess that is bound to the Relying Party, as  
1664 identified by rpId.

1665  
1666 5. Append authenticator to issuedRequests.

1667

1668 If an authenticator ceases to be available on this platform,  
1669 Remove authenticator from issuedRequests.

1670

1671 If any authenticator returns a status indicating that the user  
1672 cancelled the operation,  
1673

1674 1. Remove authenticator from issuedRequests.  
1675 2. For each remaining authenticator in issuedRequests invoke  
1676 the authenticatorCancel operation on authenticator and  
1677 remove it from issuedRequests.  
1678 Note: Authenticators may return an indication of "the  
1679 user cancelled the entire operation". How a user agent  
1680 manifests this state to users is unspecified.

1681

1682 If any authenticator returns an error status,  
1683 Remove authenticator from issuedRequests.

1684

1685 If any authenticator indicates success,  
1686

1687 1. Remove authenticator from issuedRequests.  
1688 2. Let assertionCreationData be a struct whose items are:

1689

1690 credentialIdResult  
1691 If savedCredentialIds[authenticator] exists,  
1692 set the value of credentialIdResult to be the  
1693 bytes of savedCredentialIds[authenticator].  
1694 Otherwise, set the value of credentialIdResult  
1695 to be the bytes of the credential ID returned  
1696 from the successful authenticatorGetAssertion  
1697 operation, as defined in 6.2.3 The  
1698 authenticatorGetAssertion operation.

1699

1700 clientDataJSONResult  
1701 whose value is the bytes of clientDataJSON.

1702

1703 authenticatorDataResult  
1704 whose value is the bytes of the authenticator  
1705 data returned by the authenticator.

1706

1707 signatureResult  
1708 whose value is the bytes of the signature  
1709 value returned by the authenticator.

1710

1711 userHandleResult  
1712 If the authenticator returned a user handle,  
1713 set the value of userHandleResult to be the  
1714 bytes of the returned user handle. Otherwise,  
1715 set the value of userHandleResult to null.

1716

1717 clientExtensionResults  
1718 whose value is an  
1719 AuthenticationExtensionsClientOutputs object

1673 containing extension identifier -> client  
1674 extension output entries. The entries are  
1675 created by running each extension's client  
1676 extension processing algorithm to create the  
1677 client extension outputs, for each client  
1678 extension in clientDataJSON.clientExtensions.  
1679  
1680 3. Let constructAssertionAlg be an algorithm that takes a  
1681 global object global, and whose steps are:  
1682 1. Let pubKeyCred be a new PublicKeyCredential object  
1683 associated with global whose fields are:  
1684  
1685 [[identifier]]  
1686 A new ArrayBuffer, created using  
1687 global's %ArrayBuffer%, containing the  
1688 bytes of  
1689 assertionCreationData.credentialIdResult  
1690  
1691  
1692 response  
1693 A new AuthenticatorAssertionResponse  
1694 object associated with global whose  
1695 fields are:  
1696  
1697 clientDataJSON  
1698 A new ArrayBuffer, created using  
1699 global's %ArrayBuffer%, containing  
1700 the bytes of  
1701 assertionCreationData.clientDataJS  
1702 ONResult.  
1703  
1704 authenticatorData  
1705 A new ArrayBuffer, created using  
1706 global's %ArrayBuffer%, containing  
1707 the bytes of  
1708 assertionCreationData.authenticato  
1709 rDataResult.  
1710  
1711 signature  
1712 A new ArrayBuffer, created using  
1713 global's %ArrayBuffer%, containing  
1714 the bytes of  
1715 assertionCreationData.signatureRes  
1716 ult.  
1717  
1718 userHandle  
1719 If  
1720 assertionCreationData.userHandleRe  
1721 sult is null, set this field to  
1722 null. Otherwise, set this field to  
1723 a new ArrayBuffer, created using  
1724 global's %ArrayBuffer%, containing  
1725 the bytes of  
1726 assertionCreationData.userHandleRe  
1727 sult.  
1728  
1729 [[clientExtensionsResults]]  
1730 A new ArrayBuffer, created using  
1731 global's %ArrayBuffer%, containing the  
1732 bytes of  
1733 assertionCreationData.clientExtensionRes  
1734 ults.  
1735  
1736 2. Return pubKeyCred.  
1737 4. For each remaining authenticator in issuedRequests invoke  
1738 the authenticatorCancel operation on authenticator and  
1739 remove it from issuedRequests.  
1740 5. Return constructAssertionAlg and terminate this  
1741 algorithm.  
1742

1720 containing extension identifier -> client  
1721 extension output entries. The entries are  
1722 created by running each extension's client  
1723 extension processing algorithm to create the  
1724 client extension outputs, for each client  
1725 extension in clientDataJSON.clientExtensions.  
1726  
1727 3. Let constructAssertionAlg be an algorithm that takes a  
1728 global object global, and whose steps are:  
1729 1. Let pubKeyCred be a new PublicKeyCredential object  
1730 associated with global whose fields are:  
1731  
1732 [[identifier]]  
1733 A new ArrayBuffer, created using  
1734 global's %ArrayBuffer%, containing the  
1735 bytes of  
1736 assertionCreationData.credentialIdResult  
1737  
1738  
1739 response  
1740 A new AuthenticatorAssertionResponse  
1741 object associated with global whose  
1742 fields are:  
1743  
1744 clientDataJSON  
1745 A new ArrayBuffer, created using  
1746 global's %ArrayBuffer%, containing  
1747 the bytes of  
1748 assertionCreationData.clientDataJS  
1749 ONResult.  
1750  
1751 authenticatorData  
1752 A new ArrayBuffer, created using  
1753 global's %ArrayBuffer%, containing  
1754 the bytes of  
1755 assertionCreationData.authenticato  
1756 rDataResult.  
1757  
1758 signature  
1759 A new ArrayBuffer, created using  
1760 global's %ArrayBuffer%, containing  
1761 the bytes of  
1762 assertionCreationData.signatureRes  
1763 ult.  
1764  
1765 userHandle  
1766 If  
1767 assertionCreationData.userHandleRe  
1768 sult is null, set this field to  
1769 null. Otherwise, set this field to  
1770 a new ArrayBuffer, created using  
1771 global's %ArrayBuffer%, containing  
1772 the bytes of  
1773 assertionCreationData.userHandleRe  
1774 sult.  
1775  
1776 [[clientExtensionsResults]]  
1777 A new ArrayBuffer, created using  
1778 global's %ArrayBuffer%, containing the  
1779 bytes of  
1780 assertionCreationData.clientExtensionRes  
1781 ults.  
1782  
1783 2. Return pubKeyCred.  
1784 4. For each remaining authenticator in issuedRequests invoke  
1785 the authenticatorCancel operation on authenticator and  
1786 remove it from issuedRequests.  
1787 5. Return constructAssertionAlg and terminate this  
1788 algorithm.  
1789

1743 19. Return a DOMException whose name is "NotAllowedError". In order to  
1744 prevent information leak that could identify the user without  
1745 consent, this step MUST NOT be executed before lifetimeTimer has  
1746 expired. See 14.3 Authentication Ceremony Privacy for details.  
1747  
1748 During the above process, the user agent SHOULD show some UI to the  
1749 user to guide them in the process of selecting and authorizing an  
1750 authenticator with which to complete the operation.  
1751  
1752 5.1.5. Store an existing credential - PublicKeyCredential's  
1753 [[Store]](credential, sameOriginWithAncestors) method  
1754  
1755 The [[Store]](credential, sameOriginWithAncestors) method is not  
1756 supported for Web Authentication's PublicKeyCredential type, so it  
1757 always returns an error.  
1758  
1759 Note: This algorithm is synchronous; the Promise resolution/rejection  
1760 is handled by navigator.credentials.store().  
1761  
1762 This internal method accepts two arguments:  
1763  
1764 credential  
1765 This argument is a PublicKeyCredential object.  
1766  
1767 sameOriginWithAncestors  
1768 This argument is a boolean which is true if and only if the  
1769 caller's environment settings object is same-origin with its  
1770 ancestors.  
1771  
1772 When this method is invoked, the user agent MUST execute the following  
1773 algorithm:  
1774 1. Return a DOMException whose name is "NotSupportedError", and  
1775 terminate this algorithm  
1776  
1777 5.1.6. Preventing silent access to an existing credential -  
1778 PublicKeyCredential's [[preventSilentAccess]](credential,  
1779 sameOriginWithAncestors) method  
1780  
1781 Calling the [[preventSilentAccess]](credential,  
1782 sameOriginWithAncestors) method will have no effect on authenticators  
1783 that require an authorization gesture, but setting that flag may  
1784 potentially exclude authenticators that can operate without user  
1785 intervention.  
1786  
1787 This internal method accepts no arguments.  
1788  
1789 5.1.7. Availability of User-Verifying Platform Authenticator -  
1790 PublicKeyCredential's isUserVerifyingPlatformAuthenticatorAvailable() method  
1791  
1792 Relying Parties use this method to determine whether they can create a  
1793 new credential using a user-verifying platform authenticator. Upon  
1794 invocation, the client employs a platform-specific procedure to  
1795 discover available user-verifying platform authenticators. If  
1796 successful, the client then assesses whether the user is willing to  
1797 create a credential using one of the available user-verifying platform  
1798 authenticators. This assessment may include various factors, such as:  
1799 \* Whether the user is running in private or incognito mode.  
1800 \* Whether the user has configured the client to not create such  
1801 credentials.  
1802 \* Whether the user has previously expressed an unwillingness to  
1803 create a new credential for this Relying Party, either through  
1804 configuration or by declining a user interface prompt.  
1805 \* The user's explicitly stated intentions, determined through user  
1806 interaction.  
1807  
1808 If this assessment is affirmative, the promise is resolved with the  
1809 value of True. Otherwise, the promise is resolved with the value of  
1810 False. Based on the result, the Relying Party can take further actions  
1811 to guide the user to create a credential.  
1812

1790 19. Return a DOMException whose name is "NotAllowedError". In order to  
1791 prevent information leak that could identify the user without  
1792 consent, this step MUST NOT be executed before lifetimeTimer has  
1793 expired. See 14.3 Authentication Ceremony Privacy for details.  
1794  
1795 During the above process, the user agent SHOULD show some UI to the  
1796 user to guide them in the process of selecting and authorizing an  
1797 authenticator with which to complete the operation.  
1798  
1799 5.1.5. Store an existing credential - PublicKeyCredential's  
1800 [[Store]](credential, sameOriginWithAncestors) method  
1801  
1802 The [[Store]](credential, sameOriginWithAncestors) method is not  
1803 supported for Web Authentication's PublicKeyCredential type, so it  
1804 always returns an error.  
1805  
1806 Note: This algorithm is synchronous; the Promise resolution/rejection  
1807 is handled by navigator.credentials.store().  
1808  
1809 This internal method accepts two arguments:  
1810  
1811 credential  
1812 This argument is a PublicKeyCredential object.  
1813  
1814 sameOriginWithAncestors  
1815 This argument is a boolean which is true if and only if the  
1816 caller's environment settings object is same-origin with its  
1817 ancestors.  
1818  
1819 When this method is invoked, the user agent MUST execute the following  
1820 algorithm:  
1821 1. Return a DOMException whose name is "NotSupportedError", and  
1822 terminate this algorithm  
1823  
1824 5.1.6. Preventing silent access to an existing credential -  
1825 PublicKeyCredential's [[preventSilentAccess]](credential,  
1826 sameOriginWithAncestors) method  
1827  
1828 Calling the [[preventSilentAccess]](credential,  
1829 sameOriginWithAncestors) method will have no effect on authenticators  
1830 that require an authorization gesture, but setting that flag may  
1831 potentially exclude authenticators that can operate without user  
1832 intervention.  
1833  
1834 This internal method accepts no arguments.  
1835  
1836 5.1.7. Availability of User-Verifying Platform Authenticator -  
1837 PublicKeyCredential's isUserVerifyingPlatformAuthenticatorAvailable() method  
1838  
1839 Relying Parties use this method to determine whether they can create a  
1840 new credential using a user-verifying platform authenticator. Upon  
1841 invocation, the client employs a platform-specific procedure to  
1842 discover available user-verifying platform authenticators. If  
1843 successful, the client then assesses whether the user is willing to  
1844 create a credential using one of the available user-verifying platform  
1845 authenticators. This assessment may include various factors, such as:  
1846 \* Whether the user is running in private or incognito mode.  
1847 \* Whether the user has configured the client to not create such  
1848 credentials.  
1849 \* Whether the user has previously expressed an unwillingness to  
1850 create a new credential for this Relying Party, either through  
1851 configuration or by declining a user interface prompt.  
1852 \* The user's explicitly stated intentions, determined through user  
1853 interaction.  
1854  
1855 If this assessment is affirmative, the promise is resolved with the  
1856 value of True. Otherwise, the promise is resolved with the value of  
1857 False. Based on the result, the Relying Party can take further actions  
1858 to guide the user to create a credential.  
1859

1813 This method has no arguments and returns a boolean value.  
 1814  
 1815  
 1816 If the promise will return False, the client SHOULD wait a fixed period  
 1817 of time from the invocation of the method before returning False. This  
 1818 is done so that callers cannot distinguish between the case where the  
 1819 user was unwilling to create a credential using one of the available  
 1820 user-verifying platform authenticators and the case where no  
 1821 user-verifying platform authenticator exists. Trying to make these  
 1822 cases indistinguishable is done in an attempt to not provide additional  
 1823 information that could be used for fingerprinting. A timeout value on  
 1824 the order of 10 minutes is recommended; this is enough time for  
 1825 successful user interactions to be performed but short enough that the  
 1826 dangling promise will still be resolved in a reasonably timely fashion.  
 1827 partial interface PublicKeyCredential {  
 1828 static Promise < boolean > isUserVerifyingPlatformAuthenticatorAvailable();  
 1829 };  
 1830  
 1831 **5.2. Authenticator Responses (interface AuthenticatorResponse)**  
 1832  
 1833 Authenticators respond to Relying Party requests by returning an object  
 1834 derived from the AuthenticatorResponse interface:  
 1835 [SecureContext, Exposed=Window]  
 1836 interface AuthenticatorResponse {  
 1837 [SameObject] readonly attribute ArrayBuffer clientDataJSON;  
 1838 };  
 1839  
 1840 clientDataJSON, of type ArrayBuffer, readonly  
 1841 This attribute contains a JSON serialization of the client data  
 1842 passed to the authenticator by the client in its call to either  
 1843 create() or get().  
 1844  
 1845 **5.2.1. Information about Public Key Credential (interface**  
 1846 **AuthenticatorAttestationResponse)**  
 1847  
 1848 The AuthenticatorAttestationResponse interface represents the  
 1849 authenticator's response to a client's request for the creation of a  
 1850 new public key credential. It contains information about the new  
 1851 credential that can be used to identify it for later use, and metadata  
 1852 that can be used by the Relying Party to assess the characteristics of  
 1853 the credential during registration.  
 1854 [SecureContext, Exposed=Window]  
 1855 interface AuthenticatorAttestationResponse : AuthenticatorResponse {  
 1856 [SameObject] readonly attribute ArrayBuffer attestationObject;  
 1857 };  
 1858  
 1859 clientDataJSON  
 1860 This attribute, inherited from AuthenticatorResponse, contains  
 1861 the JSON-serialized client data (see 6.3 Attestation) passed to  
 1862 the authenticator by the client in order to generate this  
 1863 credential. The exact JSON serialization must be preserved, as  
 1864 the hash of the serialized client data has been computed over  
 1865 it.  
 1866  
 1867 attestationObject, of type ArrayBuffer, readonly  
 1868 This attribute contains an attestation object, which is opaque  
 1869 to, and cryptographically protected against tampering by, the  
 1870 client. The attestation object contains both authenticator data  
 1871 and an attestation statement. The former contains the AAGUID, a  
 1872 unique credential ID, and the credential public key. The  
 1873 contents of the attestation statement are determined by the  
 1874 attestation statement format used by the authenticator. It also  
 1875 contains any additional information that the Relying Party's  
 1876 server requires to validate the attestation statement, as well  
 1877 as to decode and validate the authenticator data along with the  
 1878 JSON-serialized client data. For more details, see 6.3  
 1879 Attestation, 6.3.4 Generating an Attestation Object, and Figure  
 1880 3.  
 1881  
 1882 **5.2.2. Web Authentication Assertion (interface**  
**AuthenticatorAssertionResponse)**

1860 This method has no arguments and returns a boolean value.  
 1861  
 1862  
 1863 If the promise will return False, the client SHOULD wait a fixed period  
 1864 of time from the invocation of the method before returning False. This  
 1865 is done so that callers cannot distinguish between the case where the  
 1866 user was unwilling to create a credential using one of the available  
 1867 user-verifying platform authenticators and the case where no  
 1868 user-verifying platform authenticator exists. Trying to make these  
 1869 cases indistinguishable is done in an attempt to not provide additional  
 1870 information that could be used for fingerprinting. A timeout value on  
 1871 the order of 10 minutes is recommended; this is enough time for  
 1872 successful user interactions to be performed but short enough that the  
 1873 dangling promise will still be resolved in a reasonably timely fashion.  
 1874 partial interface PublicKeyCredential {  
 1875 static Promise < boolean > isUserVerifyingPlatformAuthenticatorAvailable();  
 1876 };  
 1877  
 1878 **5.2. Authenticator Responses (interface AuthenticatorResponse)**  
 1879  
 1880 Authenticators respond to Relying Party requests by returning an object  
 1881 derived from the AuthenticatorResponse interface:  
 1882 [SecureContext, Exposed=Window]  
 1883 interface AuthenticatorResponse {  
 1884 [SameObject] readonly attribute ArrayBuffer clientDataJSON;  
 1885 };  
 1886  
 1887 clientDataJSON, of type ArrayBuffer, readonly  
 1888 This attribute contains a JSON serialization of the client data  
 1889 passed to the authenticator by the client in its call to either  
 1890 create() or get().  
 1891  
 1892 **5.2.1. Information about Public Key Credential (interface**  
 1893 **AuthenticatorAttestationResponse)**  
 1894  
 1895 The AuthenticatorAttestationResponse interface represents the  
 1896 authenticator's response to a client's request for the creation of a  
 1897 new public key credential. It contains information about the new  
 1898 credential that can be used to identify it for later use, and metadata  
 1899 that can be used by the Relying Party to assess the characteristics of  
 1900 the credential during registration.  
 1901 [SecureContext, Exposed=Window]  
 1902 interface AuthenticatorAttestationResponse : AuthenticatorResponse {  
 1903 [SameObject] readonly attribute ArrayBuffer attestationObject;  
 1904 };  
 1905  
 1906 clientDataJSON  
 1907 This attribute, inherited from AuthenticatorResponse, contains  
 1908 the JSON-serialized client data (see 6.3 Attestation) passed to  
 1909 the authenticator by the client in order to generate this  
 1910 credential. The exact JSON serialization must be preserved, as  
 1911 the hash of the serialized client data has been computed over  
 1912 it.  
 1913  
 1914 attestationObject, of type ArrayBuffer, readonly  
 1915 This attribute contains an attestation object, which is opaque  
 1916 to, and cryptographically protected against tampering by, the  
 1917 client. The attestation object contains both authenticator data  
 1918 and an attestation statement. The former contains the AAGUID, a  
 1919 unique credential ID, and the credential public key. The  
 1920 contents of the attestation statement are determined by the  
 1921 attestation statement format used by the authenticator. It also  
 1922 contains any additional information that the Relying Party's  
 1923 server requires to validate the attestation statement, as well  
 1924 as to decode and validate the authenticator data along with the  
 1925 JSON-serialized client data. For more details, see 6.3  
 1926 Attestation, 6.3.4 Generating an Attestation Object, and Figure  
 1927 3.  
 1928  
 1929 **5.2.2. Web Authentication Assertion (interface**  
**AuthenticatorAssertionResponse)**

```

1883 The AuthenticatorAssertionResponse interface represents an
1884 authenticator's response to a client's request for generation of a new
1885 authentication assertion given the Relying Party's challenge and
1886 optional list of credentials it is aware of. This response contains a
1887 cryptographic signature proving possession of the credential private
1888 key, and optionally evidence of user consent to a specific transaction.
1889 [SecureContext, Exposed=Window]
1890 interface AuthenticatorAssertionResponse : AuthenticatorResponse {
1891   [SameObject] readonly attribute ArrayBuffer authenticatorData;
1892   [SameObject] readonly attribute ArrayBuffer signature;
1893   [SameObject] readonly attribute ArrayBuffer? userHandle;
1894 };
1895
1896 clientDataJSON
1897   This attribute, inherited from AuthenticatorResponse, contains
1898   the JSON-serialized client data (see 5.10.1 Client data used in
1899   WebAuthn signatures (dictionary CollectedClientData)) passed to
1900   the authenticator by the client in order to generate this
1901   assertion. The exact JSON serialization MUST be preserved, as
1902   the hash of the serialized client data has been computed over
1903   it.
1904
1905 authenticatorData, of type ArrayBuffer, readonly
1906   This attribute contains the authenticator data returned by the
1907   authenticator. See 6.1 Authenticator data.
1908
1909 signature, of type ArrayBuffer, readonly
1910   This attribute contains the raw signature returned from the
1911   authenticator. See 6.2.3 The authenticatorGetAssertion
1912   operation.
1913
1914 userHandle, of type ArrayBuffer, readonly, nullable
1915   This attribute contains the user handle returned from the
1916   authenticator, or null if the authenticator did not return a
1917   user handle. See 6.2.3 The authenticatorGetAssertion operation.
1918
1919 5.3. Parameters for Credential Generation (dictionary
1920   PublicKeyCredentialParameters)
1921
1922 dictionary PublicKeyCredentialParameters {
1923   required PublicKeyCredentialType type;
1924   required COSEAlgorithmIdentifier alg;
1925 };
1926
1927 This dictionary is used to supply additional parameters when creating a
1928 new credential.
1929
1930 The type member specifies the type of credential to be created.
1931
1932 The alg member specifies the cryptographic signature algorithm with
1933 which the newly generated credential will be used, and thus also the
1934 type of asymmetric key pair to be generated, e.g., RSA or Elliptic
1935 Curve.
1936
1937 Note: we use "alg" as the latter member name, rather than spelling-out
1938 "algorithm", because it will be serialized into a message to the
1939 authenticator, which may be sent over a low-bandwidth link.
1940
1941 5.4. Options for Credential Creation (dictionary
1942   PublicKeyCredentialCreationOptions)
1943
1944 dictionary PublicKeyCredentialCreationOptions {
1945   required PublicKeyCredentialRpEntity rp;
1946   required PublicKeyCredentialUserEntity user;
1947
1948   required BufferSource challenge;
1949   required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
1950
1951   unsigned long timeout;
1952

```

```

1930 The AuthenticatorAssertionResponse interface represents an
1931 authenticator's response to a client's request for generation of a new
1932 authentication assertion given the Relying Party's challenge and
1933 optional list of credentials it is aware of. This response contains a
1934 cryptographic signature proving possession of the credential private
1935 key, and optionally evidence of user consent to a specific transaction.
1936 [SecureContext, Exposed=Window]
1937 interface AuthenticatorAssertionResponse : AuthenticatorResponse {
1938   [SameObject] readonly attribute ArrayBuffer authenticatorData;
1939   [SameObject] readonly attribute ArrayBuffer signature;
1940   [SameObject] readonly attribute ArrayBuffer? userHandle;
1941 };
1942
1943 clientDataJSON
1944   This attribute, inherited from AuthenticatorResponse, contains
1945   the JSON-serialized client data (see 5.10.1 Client data used in
1946   WebAuthn signatures (dictionary CollectedClientData)) passed to
1947   the authenticator by the client in order to generate this
1948   assertion. The exact JSON serialization MUST be preserved, as
1949   the hash of the serialized client data has been computed over
1950   it.
1951
1952 authenticatorData, of type ArrayBuffer, readonly
1953   This attribute contains the authenticator data returned by the
1954   authenticator. See 6.1 Authenticator data.
1955
1956 signature, of type ArrayBuffer, readonly
1957   This attribute contains the raw signature returned from the
1958   authenticator. See 6.2.3 The authenticatorGetAssertion
1959   operation.
1960
1961 userHandle, of type ArrayBuffer, readonly, nullable
1962   This attribute contains the user handle returned from the
1963   authenticator, or null if the authenticator did not return a
1964   user handle. See 6.2.3 The authenticatorGetAssertion operation.
1965
1966 5.3. Parameters for Credential Generation (dictionary
1967   PublicKeyCredentialParameters)
1968
1969 dictionary PublicKeyCredentialParameters {
1970   required PublicKeyCredentialType type;
1971   required COSEAlgorithmIdentifier alg;
1972 };
1973
1974 This dictionary is used to supply additional parameters when creating a
1975 new credential.
1976
1977 The type member specifies the type of credential to be created.
1978
1979 The alg member specifies the cryptographic signature algorithm with
1980 which the newly generated credential will be used, and thus also the
1981 type of asymmetric key pair to be generated, e.g., RSA or Elliptic
1982 Curve.
1983
1984 Note: we use "alg" as the latter member name, rather than spelling-out
1985 "algorithm", because it will be serialized into a message to the
1986 authenticator, which may be sent over a low-bandwidth link.
1987
1988 5.4. Options for Credential Creation (dictionary
1989   PublicKeyCredentialCreationOptions)
1990
1991 dictionary PublicKeyCredentialCreationOptions {
1992   required PublicKeyCredentialRpEntity rp;
1993   required PublicKeyCredentialUserEntity user;
1994
1995   required BufferSource challenge;
1996   required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
1997
1998   unsigned long timeout;
1999

```

```

1953 sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
1954 AuthenticatorSelectionCriteria authenticatorSelection;
1955 AttestationConveyancePreference attestation = "none";
1956 AuthenticationExtensionsClientInputs extensions;
1957 };
1958
1959 rp, of type PublicKeyCredentialRpEntity
1960 This member contains data about the Relying Party responsible
1961 for the request.
1962
1963 Its value's name member is required.
1964
1965 Its value's id member specifies the relying party identifier
1966 with which the credential should be associated. If omitted, its
1967 value will be the CredentialsContainer object's relevant
1968 settings object's origin's effective domain.
1969
1970 user, of type PublicKeyCredentialUserEntity
1971 This member contains data about the user account for which the
1972 Relying Party is requesting attestation.
1973
1974 Its value's name, displayName and id members are required.
1975
1976 challenge, of type BufferSource
1977 This member contains a challenge intended to be used for
1978 generating the newly created credential's attestation object.
1979 See the 13.1 Cryptographic Challenges security consideration.
1980
1981 pubKeyCredParams, of type sequence<PublicKeyCredentialParameters>
1982 This member contains information about the desired properties of
1983 the credential to be created. The sequence is ordered from most
1984 preferred to least preferred. The platform makes a best-effort
1985 to create the most preferred credential that it can.
1986
1987 timeout, of type unsigned long
1988 This member specifies a time, in milliseconds, that the caller
1989 is willing to wait for the call to complete. This is treated as
1990 a hint, and MAY be overridden by the platform.
1991
1992 excludeCredentials, of type sequence<PublicKeyCredentialDescriptor>,
1993 defaulting to None
1994 This member is intended for use by Relying Parties that wish to
1995 limit the creation of multiple credentials for the same account
1996 on a single authenticator. The platform is requested to return
1997 an error if the new credential would be created on an
1998 authenticator that also contains one of the credentials
1999 enumerated in this parameter.
2000
2001 authenticatorSelection, of type AuthenticatorSelectionCriteria
2002 This member is intended for use by Relying Parties that wish to
2003 select the appropriate authenticators to participate in the
2004 create() operation.
2005
2006 attestation, of type AttestationConveyancePreference, defaulting to
2007 "none"
2008 This member is intended for use by Relying Parties that wish to
2009 express their preference for attestation conveyance. The default
2010 is none.
2011
2012 extensions, of type AuthenticationExtensionsClientInputs
2013 This member contains additional parameters requesting additional
2014 processing by the client and authenticator. For example, the
2015 caller may request that only authenticators with certain
2016 capabilities be used to create the credential, or that
2017 particular information be returned in the attestation object.
2018 Some extensions are defined in 9 WebAuthn Extensions; consult
2019 the IANA "WebAuthn Extension Identifier" registry established by
2020 [WebAuthn-Registries] for an up-to-date list of registered
2021 WebAuthn Extensions.
2022

```

```

2000 sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
2001 AuthenticatorSelectionCriteria authenticatorSelection;
2002 AttestationConveyancePreference attestation = "none";
2003 AuthenticationExtensionsClientInputs extensions;
2004 };
2005
2006 rp, of type PublicKeyCredentialRpEntity
2007 This member contains data about the Relying Party responsible
2008 for the request.
2009
2010 Its value's name member is required.
2011
2012 Its value's id member specifies the relying party identifier
2013 with which the credential should be associated. If omitted, its
2014 value will be the CredentialsContainer object's relevant
2015 settings object's origin's effective domain.
2016
2017 user, of type PublicKeyCredentialUserEntity
2018 This member contains data about the user account for which the
2019 Relying Party is requesting attestation.
2020
2021 Its value's name, displayName and id members are required.
2022
2023 challenge, of type BufferSource
2024 This member contains a challenge intended to be used for
2025 generating the newly created credential's attestation object.
2026 See the 13.1 Cryptographic Challenges security consideration.
2027
2028 pubKeyCredParams, of type sequence<PublicKeyCredentialParameters>
2029 This member contains information about the desired properties of
2030 the credential to be created. The sequence is ordered from most
2031 preferred to least preferred. The platform makes a best-effort
2032 to create the most preferred credential that it can.
2033
2034 timeout, of type unsigned long
2035 This member specifies a time, in milliseconds, that the caller
2036 is willing to wait for the call to complete. This is treated as
2037 a hint, and MAY be overridden by the platform.
2038
2039 excludeCredentials, of type sequence<PublicKeyCredentialDescriptor>,
2040 defaulting to None
2041 This member is intended for use by Relying Parties that wish to
2042 limit the creation of multiple credentials for the same account
2043 on a single authenticator. The platform is requested to return
2044 an error if the new credential would be created on an
2045 authenticator that also contains one of the credentials
2046 enumerated in this parameter.
2047
2048 authenticatorSelection, of type AuthenticatorSelectionCriteria
2049 This member is intended for use by Relying Parties that wish to
2050 select the appropriate authenticators to participate in the
2051 create() operation.
2052
2053 attestation, of type AttestationConveyancePreference, defaulting to
2054 "none"
2055 This member is intended for use by Relying Parties that wish to
2056 express their preference for attestation conveyance. The default
2057 is none.
2058
2059 extensions, of type AuthenticationExtensionsClientInputs
2060 This member contains additional parameters requesting additional
2061 processing by the client and authenticator. For example, the
2062 caller may request that only authenticators with certain
2063 capabilities be used to create the credential, or that
2064 particular information be returned in the attestation object.
2065 Some extensions are defined in 9 WebAuthn Extensions; consult
2066 the IANA "WebAuthn Extension Identifier" registry established by
2067 [WebAuthn-Registries] for an up-to-date list of registered
2068 WebAuthn Extensions.
2069

```

2023 5.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)  
 2024  
 2025 The PublicKeyCredentialEntity dictionary describes a user account, or a  
 2026 Relying Party, with which a public key credential is associated.  
 2027 dictionary PublicKeyCredentialEntity {  
 2028 required DOMString name;  
 2029 USVString icon;  
 2030 };  
 2031  
 2032 name, of type DOMString  
 2033 A human-readable name for the entity. Its function depends on  
 2034 what the PublicKeyCredentialEntity represents:  
 2035  
 2036 + When inherited by PublicKeyCredentialRpEntity it is a  
 2037 human-friendly identifier for the Relying Party, intended only  
 2038 for display. For example, "ACME Corporation", "Wonderful  
 2039 Widgets, Inc." or "OAO Primerteh".  
 2040 + When inherited by PublicKeyCredentialUserEntity, it is a  
 2041 human-palatable identifier for a user account. It is intended  
 2042 only for display, and SHOULD allow the user to easily tell the  
 2043 difference between user accounts with similar displayNames.  
 2044 For example, "alexm", "alex.p.mueller@example.com" or  
 2045 "+14255551234". The Relying Party MAY let the user choose  
 2046 this, and MAY restrict the choice as needed or appropriate.  
 2047 For example, a Relying Party might choose to map  
 2048 human-palatable username account identifiers to the name  
 2049 member of PublicKeyCredentialUserEntity.  
 2050  
 2051 Authenticators MUST accept and store a 64-byte minimum length  
 2052 for a name member's value. Authenticators MAY truncate a name  
 2053 member's value to a length equal to or greater than 64 bytes.  
 2054  
 2055 icon, of type USVString  
 2056 A serialized URL which resolves to an image associated with the  
 2057 entity. For example, this could be a user's avatar or a Relying  
 2058 Party's logo. This URL MUST be an a priori authenticated URL.  
 2059 Authenticators MUST accept and store a 128-byte minimum length  
 2060 for an icon member's value. Authenticators MAY ignore an icon  
 2061 member's value if its length is greater than 128 bytes.  
 2062  
 2063 5.4.2. RP Parameters for Credential Generation (dictionary  
 2064 PublicKeyCredentialRpEntity)  
 2065  
 2066 The PublicKeyCredentialRpEntity dictionary is used to supply additional  
 2067 Relying Party attributes when creating a new credential.  
 2068 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {  
 2069 DOMString id;  
 2070 };  
 2071  
 2072 id, of type DOMString  
 2073 A unique identifier for the Relying Party entity, which sets the  
 2074 RP ID.  
 2075  
 2076 5.4.3. User Account Parameters for Credential Generation (dictionary  
 2077 PublicKeyCredentialUserEntity)  
 2078  
 2079 The PublicKeyCredentialUserEntity dictionary is used to supply  
 2080 additional user account attributes when creating a new credential.  
 2081 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {  
 2082 required BufferSource id;  
 2083 required DOMString displayName;  
 2084 };  
 2085  
 2086 id, of type BufferSource  
 2087 The user handle of the user account entity.  
 2088  
 2089 displayName, of type DOMString  
 2090 A human-friendly name for the user account, intended only for  
 2091 display. For example, "Alex P. Miller" or ". The Relying  
 2092 Party SHOULD let the user choose this, and SHOULD NOT restrict

2070 5.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)  
 2071  
 2072 The PublicKeyCredentialEntity dictionary describes a user account, or a  
 2073 Relying Party, with which a public key credential is associated.  
 2074 dictionary PublicKeyCredentialEntity {  
 2075 required DOMString name;  
 2076 USVString icon;  
 2077 };  
 2078  
 2079 name, of type DOMString  
 2080 A human-readable name for the entity. Its function depends on  
 2081 what the PublicKeyCredentialEntity represents:  
 2082  
 2083 + When inherited by PublicKeyCredentialRpEntity it is a  
 2084 human-friendly identifier for the Relying Party, intended only  
 2085 for display. For example, "ACME Corporation", "Wonderful  
 2086 Widgets, Inc." or "OAO Primerteh".  
 2087 + When inherited by PublicKeyCredentialUserEntity, it is a  
 2088 human-palatable identifier for a user account. It is intended  
 2089 only for display, and SHOULD allow the user to easily tell the  
 2090 difference between user accounts with similar displayNames.  
 2091 For example, "alexm", "alex.p.mueller@example.com" or  
 2092 "+14255551234". The Relying Party MAY let the user choose  
 2093 this, and MAY restrict the choice as needed or appropriate.  
 2094 For example, a Relying Party might choose to map  
 2095 human-palatable username account identifiers to the name  
 2096 member of PublicKeyCredentialUserEntity.  
 2097  
 2098 Authenticators MUST accept and store a 64-byte minimum length  
 2099 for a name member's value. Authenticators MAY truncate a name  
 2100 member's value to a length equal to or greater than 64 bytes.  
 2101  
 2102 icon, of type USVString  
 2103 A serialized URL which resolves to an image associated with the  
 2104 entity. For example, this could be a user's avatar or a Relying  
 2105 Party's logo. This URL MUST be an a priori authenticated URL.  
 2106 Authenticators MUST accept and store a 128-byte minimum length  
 2107 for an icon member's value. Authenticators MAY ignore an icon  
 2108 member's value if its length is greater than 128 bytes.  
 2109  
 2110 5.4.2. RP Parameters for Credential Generation (dictionary  
 2111 PublicKeyCredentialRpEntity)  
 2112  
 2113 The PublicKeyCredentialRpEntity dictionary is used to supply additional  
 2114 Relying Party attributes when creating a new credential.  
 2115 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {  
 2116 DOMString id;  
 2117 };  
 2118  
 2119 id, of type DOMString  
 2120 A unique identifier for the Relying Party entity, which sets the  
 2121 RP ID.  
 2122  
 2123 5.4.3. User Account Parameters for Credential Generation (dictionary  
 2124 PublicKeyCredentialUserEntity)  
 2125  
 2126 The PublicKeyCredentialUserEntity dictionary is used to supply  
 2127 additional user account attributes when creating a new credential.  
 2128 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {  
 2129 required BufferSource id;  
 2130 required DOMString displayName;  
 2131 };  
 2132  
 2133 id, of type BufferSource  
 2134 The user handle of the user account entity.  
 2135  
 2136 displayName, of type DOMString  
 2137 A human-friendly name for the user account, intended only for  
 2138 display. For example, "Alex P. Miller" or ". The Relying  
 2139 Party SHOULD let the user choose this, and SHOULD NOT restrict

2093 the choice more than necessary.  
 2094  
 2095 Authenticators MUST accept and store a 64-byte minimum length  
 2096 for a displayName member's value. Authenticators MAY truncate a  
 2097 displayName member's value to a length equal to or greater than  
 2098 64 bytes.  
 2099  
 2100 5.4.4. Authenticator Selection Criteria (dictionary  
 2101 AuthenticatorSelectionCriteria)  
 2102  
 2103 Relying Parties may use the AuthenticatorSelectionCriteria dictionary  
 2104 to specify their requirements regarding authenticator attributes.  
 2105 dictionary AuthenticatorSelectionCriteria {  
 2106 AuthenticatorAttachment authenticatorAttachment;  
 2107 boolean requireResidentKey = false;  
 2108 UserVerificationRequirement userVerification = "preferred";  
 2109 };  
 2110  
 2111 authenticatorAttachment, of type AuthenticatorAttachment  
 2112 If this member is present, eligible authenticators are filtered  
 2113 to only authenticators attached with the specified 5.4.5  
 2114 Authenticator Attachment enumeration (enum  
 2115 AuthenticatorAttachment).  
 2116  
 2117 requireResidentKey, of type boolean, defaulting to false  
 2118 This member describes the Relying Parties' requirements  
 2119 regarding availability of the Client-side-resident Credential  
 2120 Private Key. If the parameter is set to true, the authenticator  
 2121 MUST create a Client-side-resident Credential Private Key when  
 2122 creating a public key credential.  
 2123  
 2124 userVerification, of type UserVerificationRequirement, defaulting to  
 2125 "preferred"  
 2126 This member describes the Relying Party's requirements regarding  
 2127 user verification for the create() operation. Eligible  
 2128 authenticators are filtered to only those capable of satisfying  
 2129 this requirement.  
 2130  
 2131 5.4.5. Authenticator Attachment enumeration (enum AuthenticatorAttachment)  
 2132  
 2133 enum AuthenticatorAttachment {  
 2134 "platform" // Platform attachment  
 2135 "cross-platform" // Cross-platform attachment  
 2136 };  
 2137  
 2138 Clients can communicate with authenticators using a variety of  
 2139 mechanisms. For example, a client MAY use a platform-specific API to  
 2140 communicate with an authenticator which is physically bound to a  
 2141 platform. On the other hand, a client can use a variety of standardized  
 2142 cross-platform transport protocols such as Bluetooth (see 5.10.4  
 2143 Authenticator Transport enumeration (enum AuthenticatorTransport)) to  
 2144 discover and communicate with cross-platform attached authenticators.  
 2145 Therefore, we use AuthenticatorAttachment to describe an  
 2146 authenticator's attachment modality. We define authenticators that are  
 2147 part of the client's platform as having a platform attachment, and  
 2148 refer to them as platform authenticators. While those that are  
 2149 reachable via cross-platform transport protocols are defined as having  
 2150 cross-platform attachment, and refer to them as roaming authenticators.  
 2151 \* platform attachment - the respective authenticator is attached  
 2152 using platform-specific transports. Usually, authenticators of this  
 2153 class are non-removable from the platform. A public key credential  
 2154 bound to a platform authenticator is called a platform credential.  
 2155 \* cross-platform attachment - the respective authenticator is  
 2156 attached using cross-platform transports. Authenticators of this  
 2157 class are removable from, and can "roam" among, client platforms. A  
 2158 public key credential bound to a roaming authenticator is called a  
 2159 roaming credential.  
 2160  
 2161 This distinction is important because there are use-cases where only  
 2162 platform authenticators are acceptable to a Relying Party, and

2140 the choice more than necessary.  
 2141  
 2142 Authenticators MUST accept and store a 64-byte minimum length  
 2143 for a displayName member's value. Authenticators MAY truncate a  
 2144 displayName member's value to a length equal to or greater than  
 2145 64 bytes.  
 2146  
 2147 5.4.4. Authenticator Selection Criteria (dictionary  
 2148 AuthenticatorSelectionCriteria)  
 2149  
 2150 Relying Parties may use the AuthenticatorSelectionCriteria dictionary  
 2151 to specify their requirements regarding authenticator attributes.  
 2152 dictionary AuthenticatorSelectionCriteria {  
 2153 AuthenticatorAttachment authenticatorAttachment;  
 2154 boolean requireResidentKey = false;  
 2155 UserVerificationRequirement userVerification = "preferred";  
 2156 };  
 2157  
 2158 authenticatorAttachment, of type AuthenticatorAttachment  
 2159 If this member is present, eligible authenticators are filtered  
 2160 to only authenticators attached with the specified 5.4.5  
 2161 Authenticator Attachment enumeration (enum  
 2162 AuthenticatorAttachment).  
 2163  
 2164 requireResidentKey, of type boolean, defaulting to false  
 2165 This member describes the Relying Parties' requirements  
 2166 regarding availability of the Client-side-resident Credential  
 2167 Private Key. If the parameter is set to true, the authenticator  
 2168 MUST create a Client-side-resident Credential Private Key when  
 2169 creating a public key credential.  
 2170  
 2171 userVerification, of type UserVerificationRequirement, defaulting to  
 2172 "preferred"  
 2173 This member describes the Relying Party's requirements regarding  
 2174 user verification for the create() operation. Eligible  
 2175 authenticators are filtered to only those capable of satisfying  
 2176 this requirement.  
 2177  
 2178 5.4.5. Authenticator Attachment enumeration (enum AuthenticatorAttachment)  
 2179  
 2180 enum AuthenticatorAttachment {  
 2181 "platform" // Platform attachment  
 2182 "cross-platform" // Cross-platform attachment  
 2183 };  
 2184  
 2185 Clients can communicate with authenticators using a variety of  
 2186 mechanisms. For example, a client MAY use a platform-specific API to  
 2187 communicate with an authenticator which is physically bound to a  
 2188 platform. On the other hand, a client can use a variety of standardized  
 2189 cross-platform transport protocols such as Bluetooth (see 5.10.4  
 2190 Authenticator Transport enumeration (enum AuthenticatorTransport)) to  
 2191 discover and communicate with cross-platform attached authenticators.  
 2192 Therefore, we use AuthenticatorAttachment to describe an  
 2193 authenticator's attachment modality. We define authenticators that are  
 2194 part of the client's platform as having a platform attachment, and  
 2195 refer to them as platform authenticators. While those that are  
 2196 reachable via cross-platform transport protocols are defined as having  
 2197 cross-platform attachment, and refer to them as roaming authenticators.  
 2198 \* platform attachment - the respective authenticator is attached  
 2199 using platform-specific transports. Usually, authenticators of this  
 2200 class are non-removable from the platform. A public key credential  
 2201 bound to a platform authenticator is called a platform credential.  
 2202 \* cross-platform attachment - the respective authenticator is  
 2203 attached using cross-platform transports. Authenticators of this  
 2204 class are removable from, and can "roam" among, client platforms. A  
 2205 public key credential bound to a roaming authenticator is called a  
 2206 roaming credential.  
 2207  
 2208 This distinction is important because there are use-cases where only  
 2209 platform authenticators are acceptable to a Relying Party, and



conversely ones where only roaming authenticators are employed. As a concrete example of the former, a platform credential may be used by Relying Parties to quickly and conveniently reauthenticate the user with a minimum of friction, e.g., the user will not have to dig around in their pocket for their key fob or phone. As a concrete example of the latter, when the user is accessing the Relying Party from a given client for the first time, they may be asked to use a roaming credential which was originally registered with the Relying Party using a different client.

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

5.4.6. Attestation Conveyance Preference enumeration (enum AttestationConveyancePreference)

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

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

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

5.5. Options for Assertion Generation (dictionary PublicKeyCredentialRequestOptions)

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

```
dictionary PublicKeyCredentialRequestOptions {
  required BufferSource challenge;
  unsigned long timeout;
  USVString rpld;
  sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
  UserVerificationRequirement userVerification = "preferred";
  AuthenticationExtensionsClientInputs extensions;
};
```

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

conversely ones where only roaming authenticators are employed. As a concrete example of the former, a platform credential may be used by Relying Parties to quickly and conveniently reauthenticate the user with a minimum of friction, e.g., the user will not have to dig around in their pocket for their key fob or phone. As a concrete example of the latter, when the user is accessing the Relying Party from a given client for the first time, they may be asked to use a roaming credential which was originally registered with the Relying Party using a different client.

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

5.4.6. Attestation Conveyance Preference enumeration (enum AttestationConveyancePreference)

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

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

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

5.5. Options for Assertion Generation (dictionary PublicKeyCredentialRequestOptions)

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

```
dictionary PublicKeyCredentialRequestOptions {
  required BufferSource challenge;
  unsigned long timeout;
  USVString rpld;
  sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
  UserVerificationRequirement userVerification = "preferred";
  AuthenticationExtensionsClientInputs extensions;
};
```

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

2233 authenticator signs, along with other data, when producing an  
 2234 authentication assertion. See the 13.1 Cryptographic Challenges  
 2235 security consideration.

2236  
 2237 timeout, of type unsigned long  
 2238 This OPTIONAL member specifies a time, in milliseconds, that the  
 2239 caller is willing to wait for the call to complete. The value is  
 2240 treated as a hint, and MAY be overridden by the platform.

2241  
 2242 rpId, of type USVString  
 2243 This optional member specifies the relying party identifier  
 2244 claimed by the caller. If omitted, its value will be the  
 2245 CredentialsContainer object's relevant settings object's  
 2246 origin's effective domain.

2247  
 2248 allowCredentials, of type sequence<PublicKeyCredentialDescriptor>,  
 2249 defaulting to None  
 2250 This optional member contains a list of  
 2251 PublicKeyCredentialDescriptor objects representing public key  
 2252 credentials acceptable to the caller, in descending order of the  
 2253 caller's preference (the first item in the list is the most  
 2254 preferred credential, and so on down the list).

2255  
 2256 userVerification, of type UserVerificationRequirement, defaulting to  
 2257 "preferred"  
 2258 This member describes the Relying Party's requirements regarding  
 2259 user verification for the get() operation. Eligible  
 2260 authenticators are filtered to only those capable of satisfying  
 2261 this requirement.

2262  
 2263 extensions, of type AuthenticationExtensionsClientInputs  
 2264 This OPTIONAL member contains additional parameters requesting  
 2265 additional processing by the client and authenticator. For  
 2266 example, if transaction confirmation is sought from the user,  
 2267 then the prompt string might be included as an extension.

2268  
 2269 **5.6. Abort operations with AbortSignal**

2270  
 2271 Developers are encouraged to leverage the AbortController to manage the  
 2272 [[Create]](origin, options, sameOriginWithAncestors) and  
 2273 [[DiscoverFromExternalSource]](origin, options,  
 2274 sameOriginWithAncestors) operations. See DOM 3.3 Using AbortController  
 2275 and AbortSignal objects in APIs section for detailed instructions.

2276  
 2277 Note: DOM 3.3 Using AbortController and AbortSignal objects in APIs  
 2278 section specifies that web platform APIs integrating with the  
 2279 AbortController must reject the promise immediately once the aborted  
 2280 flag is set. Given the complex inheritance and parallelization  
 2281 structure of the [[Create]](origin, options, sameOriginWithAncestors)  
 2282 and [[DiscoverFromExternalSource]](origin, options,  
 2283 sameOriginWithAncestors) methods, the algorithms for the two APIs  
 2284 fulfills this requirement by checking the aborted flag in three places.  
 2285 In the case of [[Create]](origin, options, sameOriginWithAncestors),  
 2286 the aborted flag is checked first in Credential Management 1 2.5.4  
 2287 Create a Credential immediately before calling [[Create]](origin,  
 2288 options, sameOriginWithAncestors), then in 5.1.3 Create a new  
 2289 credential - PublicKeyCredential's [[Create]](origin, options,  
 2290 sameOriginWithAncestors) method right before authenticator sessions  
 2291 start, and finally during authenticator sessions. The same goes for  
 2292 [[DiscoverFromExternalSource]](origin, options,  
 2293 sameOriginWithAncestors).

2294  
 2295 The visibility and focus state of the Window object determines whether  
 2296 the [[Create]](origin, options, sameOriginWithAncestors) and  
 2297 [[DiscoverFromExternalSource]](origin, options,  
 2298 sameOriginWithAncestors) operations should continue. When the Window  
 2299 object associated with the [Document loses focus, [[Create]](origin,  
 2300 options, sameOriginWithAncestors) and  
 2301 [[DiscoverFromExternalSource]](origin, options,  
 2302 sameOriginWithAncestors) operations SHOULD be aborted.

2280 authenticator signs, along with other data, when producing an  
 2281 authentication assertion. See the 13.1 Cryptographic Challenges  
 2282 security consideration.

2283  
 2284 timeout, of type unsigned long  
 2285 This OPTIONAL member specifies a time, in milliseconds, that the  
 2286 caller is willing to wait for the call to complete. The value is  
 2287 treated as a hint, and MAY be overridden by the platform.

2288  
 2289 rpId, of type USVString  
 2290 This optional member specifies the relying party identifier  
 2291 claimed by the caller. If omitted, its value will be the  
 2292 CredentialsContainer object's relevant settings object's  
 2293 origin's effective domain.

2294  
 2295 allowCredentials, of type sequence<PublicKeyCredentialDescriptor>,  
 2296 defaulting to None  
 2297 This optional member contains a list of  
 2298 PublicKeyCredentialDescriptor objects representing public key  
 2299 credentials acceptable to the caller, in descending order of the  
 2300 caller's preference (the first item in the list is the most  
 2301 preferred credential, and so on down the list).

2302  
 2303 userVerification, of type UserVerificationRequirement, defaulting to  
 2304 "preferred"  
 2305 This member describes the Relying Party's requirements regarding  
 2306 user verification for the get() operation. Eligible  
 2307 authenticators are filtered to only those capable of satisfying  
 2308 this requirement.

2309  
 2310 extensions, of type AuthenticationExtensionsClientInputs  
 2311 This OPTIONAL member contains additional parameters requesting  
 2312 additional processing by the client and authenticator. For  
 2313 example, if transaction confirmation is sought from the user,  
 2314 then the prompt string might be included as an extension.

2315  
 2316 **5.6. Abort operations with AbortSignal**

2317  
 2318 Developers are encouraged to leverage the AbortController to manage the  
 2319 [[Create]](origin, options, sameOriginWithAncestors) and  
 2320 [[DiscoverFromExternalSource]](origin, options,  
 2321 sameOriginWithAncestors) operations. See DOM 3.3 Using AbortController  
 2322 and AbortSignal objects in APIs section for detailed instructions.

2323  
 2324 Note: DOM 3.3 Using AbortController and AbortSignal objects in APIs  
 2325 section specifies that web platform APIs integrating with the  
 2326 AbortController must reject the promise immediately once the aborted  
 2327 flag is set. Given the complex inheritance and parallelization  
 2328 structure of the [[Create]](origin, options, sameOriginWithAncestors)  
 2329 and [[DiscoverFromExternalSource]](origin, options,  
 2330 sameOriginWithAncestors) methods, the algorithms for the two APIs  
 2331 fulfills this requirement by checking the aborted flag in three places.  
 2332 In the case of [[Create]](origin, options, sameOriginWithAncestors),  
 2333 the aborted flag is checked first in Credential Management 1 2.5.4  
 2334 Create a Credential immediately before calling [[Create]](origin,  
 2335 options, sameOriginWithAncestors), then in 5.1.3 Create a new  
 2336 credential - PublicKeyCredential's [[Create]](origin, options,  
 2337 sameOriginWithAncestors) method right before authenticator sessions  
 2338 start, and finally during authenticator sessions. The same goes for  
 2339 [[DiscoverFromExternalSource]](origin, options,  
 2340 sameOriginWithAncestors).

2341  
 2342 The visibility and focus state of the Window object determines whether  
 2343 the [[Create]](origin, options, sameOriginWithAncestors) and  
 2344 [[DiscoverFromExternalSource]](origin, options,  
 2345 sameOriginWithAncestors) operations should continue. When the Window  
 2346 object associated with the [Document loses focus, [[Create]](origin,  
 2347 options, sameOriginWithAncestors) and  
 2348 [[DiscoverFromExternalSource]](origin, options,  
 2349 sameOriginWithAncestors) operations SHOULD be aborted.

2303 The WHATWG HTML WG is discussing whether to provide a hook when a  
2304 browsing context gains or loses focuses. If a hook is provided, the  
2305 above paragraph will be updated to include the hook. See WHATWG HTML WG  
2306 Issue #2711 for more details.  
2307  
2308  
2309 5.7. Authentication Extensions Client Inputs (typedef  
2310 AuthenticationExtensionsClientInputs)  
2311  
2312 dictionary AuthenticationExtensionsClientInputs {  
2313 };  
2314  
2315 This is a dictionary containing the client extension input values for  
2316 zero or more WebAuthn extensions, as defined in 9 WebAuthn Extensions.  
2317  
2318 5.8. Authentication Extensions Client Outputs (typedef  
2319 AuthenticationExtensionsClientOutputs)  
2320  
2321 dictionary AuthenticationExtensionsClientOutputs {  
2322 };  
2323  
2324 This is a dictionary containing the client extension output values for  
2325 zero or more WebAuthn extensions, as defined in 9 WebAuthn Extensions.  
2326  
2327 5.9. Authentication Extensions Authenticator Inputs (typedef  
2328 AuthenticationExtensionsAuthenticatorInputs)  
2329  
2330 typedef record<DOMString, DOMString> AuthenticationExtensionsAuthenticatorInputs  
2331 ;  
2332  
2333 This is a dictionary containing the authenticator extension input  
2334 values for zero or more WebAuthn extensions, as defined in 9 WebAuthn  
2335 Extensions.  
2336  
2337 5.10. Supporting Data Structures  
2338  
2339 The public key credential type uses certain data structures that are  
2340 specified in supporting specifications. These are as follows.  
2341  
2342 5.10.1. Client data used in WebAuthn signatures (dictionary  
2343 CollectedClientData)  
2344  
2345 The client data represents the contextual bindings of both the Relying  
2346 Party and the client platform. It is a key-value mapping whose keys are  
2347 strings. Values can be any type that has a valid encoding in JSON. Its  
2348 structure is defined by the following Web IDL.  
2349  
2350 Note: The CollectedClientData may be extended in the future. Therefore  
2351 it's critical when parsing to be tolerant of unknown keys and of any  
2352 reordering of the keys.  
2353 dictionary CollectedClientData {  
2354 required DOMString type;  
2355 required DOMString challenge;  
2356 required DOMString origin;  
2357 TokenBinding tokenBinding;  
2358 };  
2359  
2360 dictionary TokenBinding {  
2361 required TokenBindingStatus status;  
2362 DOMString id;  
2363 };  
2364  
2365 enum TokenBindingStatus { "present", "supported", "not-supported" };  
2366  
2367 The type member contains the string "webauthn.create" when creating new  
2368 credentials, and "webauthn.get" when getting an assertion from an  
2369 existing credential. The purpose of this member is to prevent certain  
2370 types of signature confusion attacks (where an attacker substitutes one  
2371 legitimate signature for another).  
2372

2350 The WHATWG HTML WG is discussing whether to provide a hook when a  
2351 browsing context gains or loses focuses. If a hook is provided, the  
2352 above paragraph will be updated to include the hook. See WHATWG HTML WG  
2353 Issue #2711 for more details.  
2354  
2355  
2356 5.7. Authentication Extensions Client Inputs (typedef  
2357 AuthenticationExtensionsClientInputs)  
2358  
2359 dictionary AuthenticationExtensionsClientInputs {  
2360 };  
2361  
2362 This is a dictionary containing the client extension input values for  
2363 zero or more WebAuthn extensions, as defined in 9 WebAuthn Extensions.  
2364  
2365 5.8. Authentication Extensions Client Outputs (typedef  
2366 AuthenticationExtensionsClientOutputs)  
2367  
2368 dictionary AuthenticationExtensionsClientOutputs {  
2369 };  
2370  
2371 This is a dictionary containing the client extension output values for  
2372 zero or more WebAuthn extensions, as defined in 9 WebAuthn Extensions.  
2373  
2374 5.9. Authentication Extensions Authenticator Inputs (typedef  
2375 AuthenticationExtensionsAuthenticatorInputs)  
2376  
2377 typedef record<DOMString, DOMString> AuthenticationExtensionsAuthenticatorInputs  
2378 ;  
2379  
2380 This is a dictionary containing the authenticator extension input  
2381 values for zero or more WebAuthn extensions, as defined in 9 WebAuthn  
2382 Extensions.  
2383  
2384 5.10. Supporting Data Structures  
2385  
2386 The public key credential type uses certain data structures that are  
2387 specified in supporting specifications. These are as follows.  
2388  
2389 5.10.1. Client data used in WebAuthn signatures (dictionary  
2390 CollectedClientData)  
2391  
2392 The client data represents the contextual bindings of both the Relying  
2393 Party and the client platform. It is a key-value mapping whose keys are  
2394 strings. Values can be any type that has a valid encoding in JSON. Its  
2395 structure is defined by the following Web IDL.  
2396  
2397 Note: The CollectedClientData may be extended in the future. Therefore  
2398 it's critical when parsing to be tolerant of unknown keys and of any  
2399 reordering of the keys.  
2400 dictionary CollectedClientData {  
2401 required DOMString type;  
2402 required DOMString challenge;  
2403 required DOMString origin;  
2404 TokenBinding tokenBinding;  
2405 };  
2406  
2407 dictionary TokenBinding {  
2408 required TokenBindingStatus status;  
2409 DOMString id;  
2410 };  
2411  
2412 enum TokenBindingStatus { "present", "supported", "not-supported" };  
2413  
2414 The type member contains the string "webauthn.create" when creating new  
2415 credentials, and "webauthn.get" when getting an assertion from an  
2416 existing credential. The purpose of this member is to prevent certain  
2417 types of signature confusion attacks (where an attacker substitutes one  
2418 legitimate signature for another).  
2419

2373 The challenge member contains the base64url encoding of the challenge  
 2374 provided by the RP. See the 13.1 Cryptographic Challenges security  
 2375 consideration.  
 2376

2377 The origin member contains the fully qualified origin of the requester,  
 2378 as provided to the authenticator by the client, in the syntax defined  
 2379 by [RFC6454].  
 2380

2381 The tokenBinding member contains information about the state of the  
 2382 Token Binding protocol used when communicating with the Relying Party.  
 2383 The status member is one of:  
 2384 \* not-supported: when the client does not support token binding.  
 2385 \* supported: the client supports token binding, but it was not  
 2386 negotiated when communicating with the Relying Party.  
 2387 \* present: token binding was used when communicating with the Relying  
 2388 Party. In this case, the id member MUST be present and MUST be a  
 2389 base64url encoding of the Token Binding ID that was used.  
 2390

2391 This structure is used by the client to compute the following  
 2392 quantities:  
 2393

2394 JSON-serialized client data  
 2395 This is the UTF-8 encoding of the result of calling the initial  
 2396 value of JSON.stringify on a CollectedClientData dictionary.  
 2397

2398 Hash of the serialized client data  
 2399 This is the hash (computed using SHA-256) of the JSON-serialized  
 2400 client data, as constructed by the client.  
 2401

2402 5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)

```
2403 enum PublicKeyCredentialType {
2404   "public-key"
2405 };
```

2406 This enumeration defines the valid credential types. It is an extension  
 2407 point; values can be added to it in the future, as more credential  
 2408 types are defined. The values of this enumeration are used for  
 2409 versioning the Authentication Assertion and attestation structures  
 2410 according to the type of the authenticator.  
 2411

2412 Currently one credential type is defined, namely "public-key".  
 2413

2414 5.10.3. Credential Descriptor (dictionary PublicKeyCredentialDescriptor)

```
2415 dictionary PublicKeyCredentialDescriptor {
2416   required PublicKeyCredentialType type;
2417   required BufferSource id;
2418   sequence<AuthenticatorTransport> transports;
2419 };
```

2420 This dictionary contains the attributes that are specified by a caller  
 2421 when referring to a public key credential as an input parameter to the  
 2422 create() or get() methods. It mirrors the fields of the  
 2423 PublicKeyCredential object returned by the latter methods.  
 2424

2425 The type member contains the type of the public key credential the  
 2426 caller is referring to.  
 2427

2428 The id member contains the credential ID of the public key credential  
 2429 the caller is referring to.  
 2430

2431 The OPTIONAL transports member contains a hint as to how the client  
 2432 might communicate with the managing authenticator of the public key  
 2433 credential the caller is referring to.  
 2434

2435 5.10.4. Authenticator Transport enumeration (enum AuthenticatorTransport)

```
2436 enum AuthenticatorTransport {
2437   "usb",
```

2420 The challenge member contains the base64url encoding of the challenge  
 2421 provided by the RP. See the 13.1 Cryptographic Challenges security  
 2422 consideration.  
 2423

2424 The origin member contains the fully qualified origin of the requester,  
 2425 as provided to the authenticator by the client, in the syntax defined  
 2426 by [RFC6454].  
 2427

2428 The tokenBinding member contains information about the state of the  
 2429 Token Binding protocol used when communicating with the Relying Party.  
 2430 The status member is one of:  
 2431 \* not-supported: when the client does not support token binding.  
 2432 \* supported: the client supports token binding, but it was not  
 2433 negotiated when communicating with the Relying Party.  
 2434 \* present: token binding was used when communicating with the Relying  
 2435 Party. In this case, the id member MUST be present and MUST be a  
 2436 base64url encoding of the Token Binding ID that was used.  
 2437

2438 This structure is used by the client to compute the following  
 2439 quantities:  
 2440

2441 JSON-serialized client data  
 2442 This is the UTF-8 encoding of the result of calling the initial  
 2443 value of JSON.stringify on a CollectedClientData dictionary.  
 2444

2445 Hash of the serialized client data  
 2446 This is the hash (computed using SHA-256) of the JSON-serialized  
 2447 client data, as constructed by the client.  
 2448

2449 5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)

```
2450 enum PublicKeyCredentialType {
2451   "public-key"
2452 };
```

2453 This enumeration defines the valid credential types. It is an extension  
 2454 point; values can be added to it in the future, as more credential  
 2455 types are defined. The values of this enumeration are used for  
 2456 versioning the Authentication Assertion and attestation structures  
 2457 according to the type of the authenticator.  
 2458

2459 Currently one credential type is defined, namely "public-key".  
 2460

2461 5.10.3. Credential Descriptor (dictionary PublicKeyCredentialDescriptor)

```
2462 dictionary PublicKeyCredentialDescriptor {
2463   required PublicKeyCredentialType type;
2464   required BufferSource id;
2465   sequence<AuthenticatorTransport> transports;
2466 };
```

2467 This dictionary contains the attributes that are specified by a caller  
 2468 when referring to a public key credential as an input parameter to the  
 2469 create() or get() methods. It mirrors the fields of the  
 2470 PublicKeyCredential object returned by the latter methods.  
 2471

2472 The type member contains the type of the public key credential the  
 2473 caller is referring to.  
 2474

2475 The id member contains the credential ID of the public key credential  
 2476 the caller is referring to.  
 2477

2478 The OPTIONAL transports member contains a hint as to how the client  
 2479 might communicate with the managing authenticator of the public key  
 2480 credential the caller is referring to.  
 2481

2482 5.10.4. Authenticator Transport enumeration (enum AuthenticatorTransport)

```
2483 enum AuthenticatorTransport {
2484   "usb",
```

```

2443     "nfc",
2444     "ble",
2445     "internal"
2446 };
2447
2448 Authenticators may implement various transports for communicating with
2449 clients. This enumeration defines hints as to how clients might
2450 communicate with a particular authenticator in order to obtain an
2451 assertion for a specific credential. Note that these hints represent
2452 the Relying Party's best belief as to how an authenticator may be
2453 reached. A Relying Party may obtain a list of transports hints from
2454 some attestation statement formats or via some out-of-band mechanism;
2455 it is outside the scope of this specification to define that mechanism.
2456 * usb - the respective authenticator can be contacted over removable
2457 USB.
2458 * nfc - the respective authenticator can be contacted over Near Field
2459 Communication (NFC).
2460 * ble - the respective authenticator can be contacted over Bluetooth
2461 Smart (Bluetooth Low Energy / BLE).
2462 * internal - the respective authenticator is contacted using a
2463 platform-specific transport. These authenticators are not removable
2464 from the platform.
2465
2466 5.10.5. Cryptographic Algorithm Identifier (typedef COSEAlgorithmIdentifier)
2467
2468 typedef long COSEAlgorithmIdentifier;
2469
2470 A COSEAlgorithmIdentifier's value is a number identifying a
2471 cryptographic algorithm. The algorithm identifiers SHOULD be values
2472 registered in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG],
2473 for instance, -7 for "ES256" and -257 for "RS256".
2474
2475 5.10.6. User Verification Requirement enumeration (enum
2476 UserVerificationRequirement)
2477
2478 enum UserVerificationRequirement {
2479     "required",
2480     "preferred",
2481     "discouraged"
2482 };
2483
2484 A Relying Party may require user verification for some of its
2485 operations but not for others, and may use this type to express its
2486 needs.
2487
2488 The value required indicates that the Relying Party requires user
2489 verification for the operation and will fail the operation if the
2490 response does not have the UV flag set.
2491
2492 The value preferred indicates that the Relying Party prefers user
2493 verification for the operation if possible, but will not fail the
2494 operation if the response does not have the UV flag set.
2495
2496 The value discouraged indicates that the Relying Party does not want
2497 user verification employed during the operation (e.g., in the interest
2498 of minimizing disruption to the user interaction flow).
2499
2500 6. WebAuthn Authenticator Model
2501
2502 The Web Authentication API implies a specific abstract functional model
2503 for an authenticator. This section describes that authenticator model.
2504
2505 Client platforms MAY implement and expose this abstract model in any
2506 way desired. However, the behavior of the client's Web Authentication
2507 API implementation, when operating on the authenticators supported by
2508 that platform, MUST be indistinguishable from the behavior specified in
2509 5 Web Authentication API.
2510
2511 For authenticators, this model defines the logical operations that they
2512 MUST support, and the data formats that they expose to the client and

```

```

2490     "nfc",
2491     "ble",
2492 };
2493
2494 Authenticators may communicate with clients using a variety of
2495 transports. This enumeration defines a hint as to how clients might
2496 communicate with a particular authenticator in order to obtain an
2497 assertion for a specific credential. Note that these hints represent
2498 the Relying Party's best belief as to how an authenticator may be
2499 reached. A Relying Party may obtain a list of transports hints from
2500 some attestation statement formats or via some out-of-band mechanism;
2501 it is outside the scope of this specification to define that mechanism.
2502 * usb - the respective authenticator can be contacted over USB.
2503
2504 * nfc - the respective authenticator can be contacted over Near Field
2505 Communication (NFC).
2506 * ble - the respective authenticator can be contacted over Bluetooth
2507 Smart (Bluetooth Low Energy / BLE).
2508
2509 5.10.5. Cryptographic Algorithm Identifier (typedef COSEAlgorithmIdentifier)
2510
2511 typedef long COSEAlgorithmIdentifier;
2512
2513 A COSEAlgorithmIdentifier's value is a number identifying a
2514 cryptographic algorithm. The algorithm identifiers SHOULD be values
2515 registered in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG],
2516 for instance, -7 for "ES256" and -257 for "RS256".
2517
2518 5.10.6. User Verification Requirement enumeration (enum
2519 UserVerificationRequirement)
2520
2521 enum UserVerificationRequirement {
2522     "required",
2523     "preferred",
2524     "discouraged"
2525 };
2526
2527 A Relying Party may require user verification for some of its
2528 operations but not for others, and may use this type to express its
2529 needs.
2530
2531 The value required indicates that the Relying Party requires user
2532 verification for the operation and will fail the operation if the
2533 response does not have the UV flag set.
2534
2535 The value preferred indicates that the Relying Party prefers user
2536 verification for the operation if possible, but will not fail the
2537 operation if the response does not have the UV flag set.
2538
2539 The value discouraged indicates that the Relying Party does not want
2540 user verification employed during the operation (e.g., in the interest
2541 of minimizing disruption to the user interaction flow).
2542
2543 6. WebAuthn Authenticator Model
2544
2545 The Web Authentication API implies a specific abstract functional model
2546 for an authenticator. This section describes that authenticator model.
2547
2548 Client platforms MAY implement and expose this abstract model in any
2549 way desired. However, the behavior of the client's Web Authentication
2550 API implementation, when operating on the authenticators supported by
2551 that platform, MUST be indistinguishable from the behavior specified in
2552 5 Web Authentication API.
2553
2554 For authenticators, this model defines the logical operations that they
2555 MUST support, and the data formats that they expose to the client and

```

2513 the Relying Party. However, it does not define the details of how  
2514 authenticators communicate with the client platform, unless they are  
2515 necessary for interoperability with Relying Parties. For instance, this  
2516 abstract model does not define protocols for connecting authenticators  
2517 to clients over transports such as USB or NFC. Similarly, this abstract  
2518 model does not define specific error codes or methods of returning  
2519 them; however, it does define error behavior in terms of the needs of  
2520 the client. Therefore, specific error codes are mentioned as a means of  
2521 showing which error conditions must be distinguishable (or not) from  
2522 each other in order to enable a compliant and secure client  
2523 implementation.  
2524

2525 Relying Parties may influence authenticator selection, if they deem  
2526 necessary, by stipulating various authenticator characteristics when  
2527 creating credentials and/or when generating assertions, through use of  
2528 credential creation options or assertion generation options,  
2529 respectively. The algorithms underlying the WebAuthn API marshal these  
2530 options and pass them to the applicable authenticator operations  
2531 defined below.  
2532

2533 In this abstract model, the authenticator provides key management and  
2534 cryptographic signatures. It can be embedded in the WebAuthn client or  
2535 housed in a separate device entirely. The authenticator itself can  
2536 contain a cryptographic module which operates at a higher security  
2537 level than the rest of the authenticator. This is particularly  
2538 important for authenticators that are embedded in the WebAuthn client,  
2539 as in those cases this cryptographic module (which may, for example, be  
2540 a TPM) could be considered more trustworthy than the rest of the  
2541 authenticator.  
2542

2543 Each authenticator stores a credentials map, a map from (rpId,  
2544 [userHandle]) to public key credential source.  
2545

2546 Additionally, each authenticator has an AAGUID, which is a 128-bit  
2547 identifier indicating the type (e.g. make and model) of the  
2548 authenticator. The AAGUID MUST be chosen by the manufacturer to be  
2549 identical across all substantially identical authenticators made by  
2550 that manufacturer, and different (with high probability) from the  
2551 AAGUIDs of all other types of authenticators. The AAGUID for a given  
2552 type of authenticator SHOULD be randomly generated to ensure this. The  
2553 RP MAY use the AAGUID to infer certain properties of the authenticator,  
2554 such as certification level and strength of key protection, using  
2555 information from other sources.  
2556

2557 The primary function of the authenticator is to provide WebAuthn  
2558 signatures, which are bound to various contextual data. These data are  
2559 observed and added at different levels of the stack as a signature  
2560 request passes from the server to the authenticator. In verifying a  
2561 signature, the server checks these bindings against expected values.  
2562 These contextual bindings are divided in two: Those added by the RP or  
2563 the client, referred to as client data; and those added by the  
2564 authenticator, referred to as the authenticator data. The authenticator  
2565 signs over the client data, but is otherwise not interested in its  
2566 contents. To save bandwidth and processing requirements on the  
2567 authenticator, the client hashes the client data and sends only the  
2568 result to the authenticator. The authenticator signs over the  
2569 combination of the hash of the serialized client data, and its own  
2570 authenticator data.  
2571

2572 The goals of this design can be summarized as follows.  
2573 \* The scheme for generating signatures should accommodate cases where  
2574 the link between the client platform and authenticator is very  
2575 limited, in bandwidth and/or latency. Examples include Bluetooth  
2576 Low Energy and Near-Field Communication.  
2577 \* The data processed by the authenticator should be small and easy to  
2578 interpret in low-level code. In particular, authenticators should  
2579 not have to parse high-level encodings such as JSON.  
2580 \* Both the client platform and the authenticator should have the  
2581 flexibility to add contextual bindings as needed.  
2582 \* The design aims to reuse as much as possible of existing encoding

2555 the Relying Party. However, it does not define the details of how  
2556 authenticators communicate with the client platform, unless they are  
2557 necessary for interoperability with Relying Parties. For instance, this  
2558 abstract model does not define protocols for connecting authenticators  
2559 to clients over transports such as USB or NFC. Similarly, this abstract  
2560 model does not define specific error codes or methods of returning  
2561 them; however, it does define error behavior in terms of the needs of  
2562 the client. Therefore, specific error codes are mentioned as a means of  
2563 showing which error conditions must be distinguishable (or not) from  
2564 each other in order to enable a compliant and secure client  
2565 implementation.  
2566

2567 Relying Parties may influence authenticator selection, if they deem  
2568 necessary, by stipulating various authenticator characteristics when  
2569 creating credentials and/or when generating assertions, through use of  
2570 credential creation options or assertion generation options,  
2571 respectively. The algorithms underlying the WebAuthn API marshal these  
2572 options and pass them to the applicable authenticator operations  
2573 defined below.  
2574

2575 In this abstract model, the authenticator provides key management and  
2576 cryptographic signatures. It can be embedded in the WebAuthn client or  
2577 housed in a separate device entirely. The authenticator itself can  
2578 contain a cryptographic module which operates at a higher security  
2579 level than the rest of the authenticator. This is particularly  
2580 important for authenticators that are embedded in the WebAuthn client,  
2581 as in those cases this cryptographic module (which may, for example, be  
2582 a TPM) could be considered more trustworthy than the rest of the  
2583 authenticator.  
2584

2585 Each authenticator stores a credentials map, a map from (rpId,  
2586 [userHandle]) to public key credential source.  
2587

2588 Additionally, each authenticator has an AAGUID, which is a 128-bit  
2589 identifier indicating the type (e.g. make and model) of the  
2590 authenticator. The AAGUID MUST be chosen by the manufacturer to be  
2591 identical across all substantially identical authenticators made by  
2592 that manufacturer, and different (with high probability) from the  
2593 AAGUIDs of all other types of authenticators. The AAGUID for a given  
2594 type of authenticator SHOULD be randomly generated to ensure this. The  
2595 RP MAY use the AAGUID to infer certain properties of the authenticator,  
2596 such as certification level and strength of key protection, using  
2597 information from other sources.  
2598

2599 The primary function of the authenticator is to provide WebAuthn  
2600 signatures, which are bound to various contextual data. These data are  
2601 observed and added at different levels of the stack as a signature  
2602 request passes from the server to the authenticator. In verifying a  
2603 signature, the server checks these bindings against expected values.  
2604 These contextual bindings are divided in two: Those added by the RP or  
2605 the client, referred to as client data; and those added by the  
2606 authenticator, referred to as the authenticator data. The authenticator  
2607 signs over the client data, but is otherwise not interested in its  
2608 contents. To save bandwidth and processing requirements on the  
2609 authenticator, the client hashes the client data and sends only the  
2610 result to the authenticator. The authenticator signs over the  
2611 combination of the hash of the serialized client data, and its own  
2612 authenticator data.  
2613

2614 The goals of this design can be summarized as follows.  
2615 \* The scheme for generating signatures should accommodate cases where  
2616 the link between the client platform and authenticator is very  
2617 limited, in bandwidth and/or latency. Examples include Bluetooth  
2618 Low Energy and Near-Field Communication.  
2619 \* The data processed by the authenticator should be small and easy to  
2620 interpret in low-level code. In particular, authenticators should  
2621 not have to parse high-level encodings such as JSON.  
2622 \* Both the client platform and the authenticator should have the  
2623 flexibility to add contextual bindings as needed.  
2624 \* The design aims to reuse as much as possible of existing encoding

2583 formats in order to aid adoption and implementation.  
 2584  
 2585 Authenticators produce cryptographic signatures for two distinct  
 2586 purposes:  
 2587 1. An attestation signature is produced when a new public key  
 2588 credential is created via an authenticatorMakeCredential operation.  
 2589 An attestation signature provides cryptographic proof of certain  
 2590 properties of the authenticator and the credential. For instance,  
 2591 an attestation signature asserts the authenticator type (as denoted  
 2592 by its AAGUID) and the credential public key. The attestation  
 2593 signature is signed by an attestation private key, which is chosen  
 2594 depending on the type of attestation desired. For more details on  
 2595 attestation, see 6.3 Attestation.  
 2596 2. An assertion signature is produced when the  
 2597 authenticatorGetAssertion method is invoked. It represents an  
 2598 assertion by the authenticator that the user has consented to a  
 2599 specific transaction, such as logging in, or completing a purchase.  
 2600 Thus, an assertion signature asserts that the authenticator  
 2601 possessing a particular credential private key has established, to  
 2602 the best of its ability, that the user requesting this transaction  
 2603 is the same user who consented to creating that particular public  
 2604 key credential. It also asserts additional information, termed  
 2605 client data, that may be useful to the caller, such as the means by  
 2606 which user consent was provided, and the prompt shown to the user  
 2607 by the authenticator. The assertion signature format is illustrated  
 2608 in Figure 2, below.  
 2609  
 2610 The formats of these signatures, as well as the procedures for  
 2611 generating them, are specified below.  
 2612  
 2613 6.1. Authenticator data  
 2614  
 2615 The authenticator data structure encodes contextual bindings made by  
 2616 the authenticator. These bindings are controlled by the authenticator  
 2617 itself, and derive their trust from the Relying Party's assessment of  
 2618 the security properties of the authenticator. In one extreme case, the  
 2619 authenticator may be embedded in the client, and its bindings may be no  
 2620 more trustworthy than the client data. At the other extreme, the  
 2621 authenticator may be a discrete entity with high-security hardware and  
 2622 software, connected to the client over a secure channel. In both cases,  
 2623 the Relying Party receives the authenticator data in the same format,  
 2624 and uses its knowledge of the authenticator to make trust decisions.  
 2625  
 2626 The authenticator data has a compact but extensible encoding. This is  
 2627 desired since authenticators can be devices with limited capabilities  
 2628 and low power requirements, with much simpler software stacks than the  
 2629 client platform components.  
 2630  
 2631 The authenticator data structure is a byte array of 37 bytes or more,  
 2632 as follows.  
 2633  
 2634 Name Length (in bytes) Description  
 2635 rpIdHash 32 SHA-256 hash of the RP ID associated with the credential.  
 2636 flags 1 Flags (bit 0 is the least significant bit):  
 2637 \* Bit 0: User Present (UP) result.  
 2638 + 1 means the user is present.  
 2639 + 0 means the user is not present.  
 2640 \* Bit 1: Reserved for future use (RFU1).  
 2641 \* Bit 2: User Verified (UV) result.  
 2642 + 1 means the user is verified.  
 2643 + 0 means the user is not verified.  
 2644 \* Bits 3-5: Reserved for future use (RFU2).  
 2645 \* Bit 6: Attested credential data included (AT).  
 2646 + Indicates whether the authenticator added attested credential  
 2647 data.  
 2648 \* Bit 7: Extension data included (ED).  
 2649 + Indicates if the authenticator data has extensions.  
 2650  
 2651 signCount 4 Signature counter, 32-bit unsigned big-endian integer.  
 2652 attestedCredentialData variable (if present) attested credential data

2625 formats in order to aid adoption and implementation.  
 2626  
 2627 Authenticators produce cryptographic signatures for two distinct  
 2628 purposes:  
 2629 1. An attestation signature is produced when a new public key  
 2630 credential is created via an authenticatorMakeCredential operation.  
 2631 An attestation signature provides cryptographic proof of certain  
 2632 properties of the authenticator and the credential. For instance,  
 2633 an attestation signature asserts the authenticator type (as denoted  
 2634 by its AAGUID) and the credential public key. The attestation  
 2635 signature is signed by an attestation private key, which is chosen  
 2636 depending on the type of attestation desired. For more details on  
 2637 attestation, see 6.3 Attestation.  
 2638 2. An assertion signature is produced when the  
 2639 authenticatorGetAssertion method is invoked. It represents an  
 2640 assertion by the authenticator that the user has consented to a  
 2641 specific transaction, such as logging in, or completing a purchase.  
 2642 Thus, an assertion signature asserts that the authenticator  
 2643 possessing a particular credential private key has established, to  
 2644 the best of its ability, that the user requesting this transaction  
 2645 is the same user who consented to creating that particular public  
 2646 key credential. It also asserts additional information, termed  
 2647 client data, that may be useful to the caller, such as the means by  
 2648 which user consent was provided, and the prompt shown to the user  
 2649 by the authenticator. The assertion signature format is illustrated  
 2650 in Figure 2, below.  
 2651  
 2652 The formats of these signatures, as well as the procedures for  
 2653 generating them, are specified below.  
 2654  
 2655 6.1. Authenticator data  
 2656  
 2657 The authenticator data structure encodes contextual bindings made by  
 2658 the authenticator. These bindings are controlled by the authenticator  
 2659 itself, and derive their trust from the Relying Party's assessment of  
 2660 the security properties of the authenticator. In one extreme case, the  
 2661 authenticator may be embedded in the client, and its bindings may be no  
 2662 more trustworthy than the client data. At the other extreme, the  
 2663 authenticator may be a discrete entity with high-security hardware and  
 2664 software, connected to the client over a secure channel. In both cases,  
 2665 the Relying Party receives the authenticator data in the same format,  
 2666 and uses its knowledge of the authenticator to make trust decisions.  
 2667  
 2668 The authenticator data has a compact but extensible encoding. This is  
 2669 desired since authenticators can be devices with limited capabilities  
 2670 and low power requirements, with much simpler software stacks than the  
 2671 client platform components.  
 2672  
 2673 The authenticator data structure is a byte array of 37 bytes or more,  
 2674 as follows.  
 2675  
 2676 Name Length (in bytes) Description  
 2677 rpIdHash 32 SHA-256 hash of the RP ID associated with the credential.  
 2678 flags 1 Flags (bit 0 is the least significant bit):  
 2679 \* Bit 0: User Present (UP) result.  
 2680 + 1 means the user is present.  
 2681 + 0 means the user is not present.  
 2682 \* Bit 1: Reserved for future use (RFU1).  
 2683 \* Bit 2: User Verified (UV) result.  
 2684 + 1 means the user is verified.  
 2685 + 0 means the user is not verified.  
 2686 \* Bits 3-5: Reserved for future use (RFU2).  
 2687 \* Bit 6: Attested credential data included (AT).  
 2688 + Indicates whether the authenticator added attested credential  
 2689 data.  
 2690 \* Bit 7: Extension data included (ED).  
 2691 + Indicates if the authenticator data has extensions.  
 2692  
 2693 signCount 4 Signature counter, 32-bit unsigned big-endian integer.  
 2694 attestedCredentialData variable (if present) attested credential data

2653 (if present). See 6.3.1 Attested credential data for details. Its  
2654 length depends on the length of the credential ID and credential public  
2655 key being attested.  
2656 extensions variable (if present) Extension-defined authenticator data.  
2657 This is a CBOR [RFC7049] map with extension identifiers as keys, and  
2658 authenticator extension outputs as values. See 9 WebAuthn Extensions  
2659 for details.

2660 NOTE: The names in the Name column in the above table are only for  
2661 reference within this document, and are not present in the actual  
2662 representation of the authenticator data.  
2663  
2664

2665 The RP ID is originally received from the client when the credential is  
2666 created, and again when an assertion is generated. However, it differs  
2667 from other client data in some important ways. First, unlike the client  
2668 data, the RP ID of a credential does not change between operations but  
2669 instead remains the same for the lifetime of that credential. Secondly,  
2670 it is validated by the authenticator during the  
2671 authenticatorGetAssertion operation, by verifying that the RP ID  
2672 associated with the requested credential exactly matches the RP ID  
2673 supplied by the client, and that the RP ID is a registrable domain  
2674 suffix of or is equal to the effective domain of the RP's origin's  
2675 effective domain.  
2676

2677 The UP flag SHALL be set if and only if the authenticator detected a  
2678 user through an authenticator specific gesture. The RFU bits SHALL be  
2679 set to zero.  
2680

2681 For attestation signatures, the authenticator MUST set the AT flag and  
2682 include the attestedCredentialData. For authentication signatures, the  
2683 AT flag MUST NOT be set and the attestedCredentialData MUST NOT be  
2684 included.  
2685

2686 If the authenticator does not include any extension data, it MUST set  
2687 the ED flag to zero, and to one if extension data is included.  
2688

2689 The figure below shows a visual representation of the authenticator  
2690 data structure.  
2691 [fido-signature-formats-figure1.svg] Authenticator data layout.  
2692

2693 Note that the authenticator data describes its own length: If the AT  
2694 and ED flags are not set, it is always 37 bytes long. The attested  
2695 credential data (which is only present if the AT flag is set) describes  
2696 its own length. If the ED flag is set, then the total length is 37  
2697 bytes plus the length of the attested credential data, plus the length  
2698 of the CBOR map that follows.  
2699

### 2700 6.1.1. Signature Counter Considerations

2701 Authenticators MUST implement a signature counter feature. The  
2702 signature counter is incremented for each successful  
2703 authenticatorGetAssertion operation by some positive value, and its  
2704 value is returned to the Relying Party within the authenticator data.  
2705 The signature counter's purpose is to aid Relying Parties in detecting  
2706 cloned authenticators. Clone detection is more important for  
2707 authenticators with limited protection measures.  
2708  
2709

2710 An Relying Party stores the signature counter of the most recent  
2711 authenticatorGetAssertion operation. Upon a new  
2712 authenticatorGetAssertion operation, the Relying Party compares the  
2713 stored signature counter value with the new signCount value returned in  
2714 the assertion's authenticator data. If this new signCount value is less  
2715 than or equal to the stored value, a cloned authenticator may exist, or  
2716 the authenticator may be malfunctioning.  
2717

2718 Detecting a signature counter mismatch does not indicate whether the  
2719 current operation was performed by a cloned authenticator or the  
2720 original authenticator. Relying Parties should address this situation  
2721 appropriately relative to their individual situations, i.e., their risk  
2722 tolerance.

2695 (if present). See 6.3.1 Attested credential data for details. Its  
2696 length depends on the length of the credential ID and credential public  
2697 key being attested.  
2698 extensions variable (if present) Extension-defined authenticator data.  
2699 This is a CBOR [RFC7049] map with extension identifiers as keys, and  
2700 authenticator extension outputs as values. See 9 WebAuthn Extensions  
2701 for details.

2702 NOTE: The names in the Name column in the above table are only for  
2703 reference within this document, and are not present in the actual  
2704 representation of the authenticator data.  
2705  
2706

2707 The RP ID is originally received from the client when the credential is  
2708 created, and again when an assertion is generated. However, it differs  
2709 from other client data in some important ways. First, unlike the client  
2710 data, the RP ID of a credential does not change between operations but  
2711 instead remains the same for the lifetime of that credential. Secondly,  
2712 it is validated by the authenticator during the  
2713 authenticatorGetAssertion operation, by verifying that the RP ID  
2714 associated with the requested credential exactly matches the RP ID  
2715 supplied by the client, and that the RP ID is a registrable domain  
2716 suffix of or is equal to the effective domain of the RP's origin's  
2717 effective domain.  
2718

2719 The UP flag SHALL be set if and only if the authenticator detected a  
2720 user through an authenticator specific gesture. The RFU bits SHALL be  
2721 set to zero.  
2722

2723 For attestation signatures, the authenticator MUST set the AT flag and  
2724 include the attestedCredentialData. For authentication signatures, the  
2725 AT flag MUST NOT be set and the attestedCredentialData MUST NOT be  
2726 included.  
2727

2728 If the authenticator does not include any extension data, it MUST set  
2729 the ED flag to zero, and to one if extension data is included.  
2730

2731 The figure below shows a visual representation of the authenticator  
2732 data structure.  
2733 [fido-signature-formats-figure1.svg] Authenticator data layout.  
2734

2735 Note that the authenticator data describes its own length: If the AT  
2736 and ED flags are not set, it is always 37 bytes long. The attested  
2737 credential data (which is only present if the AT flag is set) describes  
2738 its own length. If the ED flag is set, then the total length is 37  
2739 bytes plus the length of the attested credential data, plus the length  
2740 of the CBOR map that follows.  
2741

### 2742 6.1.1. Signature Counter Considerations

2743 Authenticators MUST implement a signature counter feature. The  
2744 signature counter is incremented for each successful  
2745 authenticatorGetAssertion operation by some positive value, and its  
2746 value is returned to the Relying Party within the authenticator data.  
2747 The signature counter's purpose is to aid Relying Parties in detecting  
2748 cloned authenticators. Clone detection is more important for  
2749 authenticators with limited protection measures.  
2750  
2751

2752 An Relying Party stores the signature counter of the most recent  
2753 authenticatorGetAssertion operation. Upon a new  
2754 authenticatorGetAssertion operation, the Relying Party compares the  
2755 stored signature counter value with the new signCount value returned in  
2756 the assertion's authenticator data. If this new signCount value is less  
2757 than or equal to the stored value, a cloned authenticator may exist, or  
2758 the authenticator may be malfunctioning.  
2759

2760 Detecting a signature counter mismatch does not indicate whether the  
2761 current operation was performed by a cloned authenticator or the  
2762 original authenticator. Relying Parties should address this situation  
2763 appropriately relative to their individual situations, i.e., their risk  
2764 tolerance.



2723  
2724  
2725  
2726  
2727  
2728  
2729  
2730  
2731  
2732  
2733  
2734  
2735  
2736  
2737  
2738  
2739  
2740  
2741  
2742  
2743  
2744  
2745  
2746  
2747  
2748  
2749  
2750  
2751  
2752  
2753  
2754  
2755  
2756  
2757  
2758  
2759  
2760  
2761  
2762  
2763  
2764  
2765  
2766  
2767  
2768  
2769  
2770  
2771  
2772  
2773  
2774  
2775  
2776  
2777  
2778  
2779  
2780  
2781  
2782  
2783  
2784  
2785  
2786  
2787  
2788  
2789  
2790  
2791  
2792

**Authenticators:**

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

**6.1.2. FIDO U2F signature format compatibility**

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

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

**6.2. Authenticator operations**

A WebAuthn Client MUST connect to an authenticator in order to invoke any of the operations of that authenticator. This connection defines an authenticator session. An authenticator must maintain isolation between sessions. It may do this by only allowing one session to exist at any particular time, or by providing more complicated session management.

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

**6.2.1. Lookup Credential Source by Credential ID algorithm**

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

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

**6.2.2. The authenticatorMakeCredential operation**

It takes the following input parameters:

- hash  
The hash of the serialized client data, provided by the client.
- rpEntity  
The Relying Party's PublicKeyCredentialRpEntity.
- userEntity  
The user account's PublicKeyCredentialUserEntity, containing the user handle given by the Relying Party.
- requireResidentKey  
The authenticatorSelection.requireResidentKey value given by the Relying Party.

2765  
2766  
2767  
2768  
2769  
2770  
2771  
2772  
2773  
2774  
2775  
2776  
2777  
2778  
2779  
2780  
2781  
2782  
2783  
2784  
2785  
2786  
2787  
2788  
2789  
2790  
2791  
2792  
2793  
2794  
2795  
2796  
2797  
2798  
2799  
2800  
2801  
2802  
2803  
2804  
2805  
2806  
2807  
2808  
2809  
2810  
2811  
2812  
2813  
2814  
2815  
2816  
2817  
2818  
2819  
2820  
2821  
2822  
2823  
2824  
2825  
2826  
2827  
2828  
2829  
2830  
2831  
2832  
2833  
2834

**Authenticators:**

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

**6.1.2. FIDO U2F signature format compatibility**

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

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

**6.2. Authenticator operations**

A WebAuthn Client MUST connect to an authenticator in order to invoke any of the operations of that authenticator. This connection defines an authenticator session. An authenticator must maintain isolation between sessions. It may do this by only allowing one session to exist at any particular time, or by providing more complicated session management.

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

**6.2.1. Lookup Credential Source by Credential ID algorithm**

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

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

**6.2.2. The authenticatorMakeCredential operation**

It takes the following input parameters:

- hash  
The hash of the serialized client data, provided by the client.
- rpEntity  
The Relying Party's PublicKeyCredentialRpEntity.
- userEntity  
The user account's PublicKeyCredentialUserEntity, containing the user handle given by the Relying Party.
- requireResidentKey  
The authenticatorSelection.requireResidentKey value given by the Relying Party.

2793 requireUserPresence  
 2794 A Boolean value provided by the client, which in invocations  
 2795 from a WebAuthn Client's [[Create]](origin, options,  
 2796 sameOriginWithAncestors) method is always set to the inverse of  
 2797 requireUserVerification.  
 2798  
 2799 requireUserVerification  
 2800 The effective user verification requirement for credential  
 2801 creation, a Boolean value provided by the client.  
 2802  
 2803 credTypesAndPubKeyAlgs  
 2804 A sequence of pairs of PublicKeyCredentialType and public key  
 2805 algorithms (COSEAlgorithmIdentifier) requested by the Relying  
 2806 Party. This sequence is ordered from most preferred to least  
 2807 preferred. The platform makes a best-effort to create the most  
 2808 preferred credential that it can.  
 2809  
 2810 excludeCredentialDescriptorList  
 2811 An optional list of PublicKeyCredentialDescriptor objects  
 2812 provided by the Relying Party with the intention that, if any of  
 2813 these are known to the authenticator, it should not create a new  
 2814 credential. excludeCredentialDescriptorList contains a list of  
 2815 known credentials.  
 2816  
 2817 extensions  
 2818 A CBOR map from extension identifiers to their authenticator  
 2819 extension inputs, created by the client based on the extensions  
 2820 requested by the Relying Party, if any.  
 2821  
 2822 Note: Before performing this operation, all other operations in  
 2823 progress in the authenticator session MUST be aborted by running the  
 2824 authenticatorCancel operation.  
 2825  
 2826 When this operation is invoked, the authenticator MUST perform the  
 2827 following procedure:  
 2828 1. Check if all the supplied parameters are syntactically well-formed  
 2829 and of the correct length. If not, return an error code equivalent  
 2830 to "UnknownError" and terminate the operation.  
 2831 2. Check if at least one of the specified combinations of  
 2832 PublicKeyCredentialType and cryptographic parameters in  
 2833 credTypesAndPubKeyAlgs is supported. If not, return an error code  
 2834 equivalent to "NotSupportedError" and terminate the operation.  
 2835 3. For each descriptor of excludeCredentialDescriptorList:  
 2836 1. If looking up descriptor.id in this authenticator returns  
 2837 non-null, and the returned item's RP ID and type match  
 2838 rpEntity.id and excludeCredentialDescriptorList.type  
 2839 respectively, then obtain user consent for creating a new  
 2840 credential. The method of obtaining user consent MUST include  
 2841 a test of user presence. If the user  
 2842  
 2843 confirms consent to create a new credential  
 2844 return an error code equivalent to  
 2845 "InvalidStateError" and terminate the operation.  
 2846  
 2847 does not consent to create a new credential  
 2848 return an error code equivalent to "NotAllowedError"  
 2849 and terminate the operation.  
 2850  
 2851 4. If requireResidentKey is true and the authenticator cannot store a  
 2852 Client-side-resident Credential Private Key, return an error code  
 2853 equivalent to "ConstraintError" and terminate the operation.  
 2854 5. If requireUserVerification is true and the authenticator cannot  
 2855 perform user verification, return an error code equivalent to  
 2856 "ConstraintError" and terminate the operation.  
 2857 6. Obtain user consent for creating a new credential. The prompt for  
 2858 obtaining this consent is shown by the authenticator if it has its  
 2859 own output capability, or by the user agent otherwise. The prompt  
 2860 SHOULD display rpEntity.id, rpEntity.name, userEntity.name and  
 2861 userEntity.displayName, if possible.  
 2862 If requireUserVerification is true, the method of obtaining user

2835 requireUserPresence  
 2836 A Boolean value provided by the client, which in invocations  
 2837 from a WebAuthn Client's [[Create]](origin, options,  
 2838 sameOriginWithAncestors) method is always set to the inverse of  
 2839 requireUserVerification.  
 2840  
 2841 requireUserVerification  
 2842 The effective user verification requirement for credential  
 2843 creation, a Boolean value provided by the client.  
 2844  
 2845 credTypesAndPubKeyAlgs  
 2846 A sequence of pairs of PublicKeyCredentialType and public key  
 2847 algorithms (COSEAlgorithmIdentifier) requested by the Relying  
 2848 Party. This sequence is ordered from most preferred to least  
 2849 preferred. The platform makes a best-effort to create the most  
 2850 preferred credential that it can.  
 2851  
 2852 excludeCredentialDescriptorList  
 2853 An optional list of PublicKeyCredentialDescriptor objects  
 2854 provided by the Relying Party with the intention that, if any of  
 2855 these are known to the authenticator, it should not create a new  
 2856 credential. excludeCredentialDescriptorList contains a list of  
 2857 known credentials.  
 2858  
 2859 extensions  
 2860 A CBOR map from extension identifiers to their authenticator  
 2861 extension inputs, created by the client based on the extensions  
 2862 requested by the Relying Party, if any.  
 2863  
 2864 Note: Before performing this operation, all other operations in  
 2865 progress in the authenticator session MUST be aborted by running the  
 2866 authenticatorCancel operation.  
 2867  
 2868 When this operation is invoked, the authenticator MUST perform the  
 2869 following procedure:  
 2870 1. Check if all the supplied parameters are syntactically well-formed  
 2871 and of the correct length. If not, return an error code equivalent  
 2872 to "UnknownError" and terminate the operation.  
 2873 2. Check if at least one of the specified combinations of  
 2874 PublicKeyCredentialType and cryptographic parameters in  
 2875 credTypesAndPubKeyAlgs is supported. If not, return an error code  
 2876 equivalent to "NotSupportedError" and terminate the operation.  
 2877 3. For each descriptor of excludeCredentialDescriptorList:  
 2878 1. If looking up descriptor.id in this authenticator returns  
 2879 non-null, and the returned item's RP ID and type match  
 2880 rpEntity.id and excludeCredentialDescriptorList.type  
 2881 respectively, then obtain user consent for creating a new  
 2882 credential. The method of obtaining user consent MUST include  
 2883 a test of user presence. If the user  
 2884  
 2885 confirms consent to create a new credential  
 2886 return an error code equivalent to  
 2887 "InvalidStateError" and terminate the operation.  
 2888  
 2889 does not consent to create a new credential  
 2890 return an error code equivalent to "NotAllowedError"  
 2891 and terminate the operation.  
 2892  
 2893 4. If requireResidentKey is true and the authenticator cannot store a  
 2894 Client-side-resident Credential Private Key, return an error code  
 2895 equivalent to "ConstraintError" and terminate the operation.  
 2896 5. If requireUserVerification is true and the authenticator cannot  
 2897 perform user verification, return an error code equivalent to  
 2898 "ConstraintError" and terminate the operation.  
 2899 6. Obtain user consent for creating a new credential. The prompt for  
 2900 obtaining this consent is shown by the authenticator if it has its  
 2901 own output capability, or by the user agent otherwise. The prompt  
 2902 SHOULD display rpEntity.id, rpEntity.name, userEntity.name and  
 2903 userEntity.displayName, if possible.  
 2904 If requireUserVerification is true, the method of obtaining user

2863 consent MUST include user verification.  
 2864 If requireUserPresence is true, the method of obtaining user  
 2865 consent MUST include a test of user presence.  
 2866 If the user does not consent or if user verification fails, return  
 2867 an error code equivalent to "NotAllowedError" and terminate the  
 2868 operation.  
 2869 7. Once user consent has been obtained, generate a new credential  
 2870 object:  
 2871 1. Let (publicKey, privateKey) be a new pair of cryptographic  
 2872 keys using the combination of PublicKeyCredentialType and  
 2873 cryptographic parameters represented by the first item in  
 2874 credTypesAndPubKeyAlgs that is supported by this  
 2875 authenticator.  
 2876 2. Let userHandle be userEntity.id.  
 2877 3. Let credentialSource be a new public key credential source  
 2878 with the fields:  
 2879     type  
 2880         public-key.  
 2881     privateKey  
 2882         privateKey  
 2883     rpId  
 2884         rpEntity.id  
 2885     userHandle  
 2886         userHandle  
 2887     otherUI  
 2888         Any other information the authenticator chooses to  
 2889         include.  
 2890 4. If requireResidentKey is true or the authenticator chooses to  
 2891 create a Client-side-resident Credential Private Key:  
 2892 1. Let credentialId be a new credential id.  
 2893 2. Set credentialSource.id to credentialId.  
 2894 3. Let credentials be this authenticator's credentials map.  
 2895 4. Set credentials[(rpEntity.id, userHandle)] to  
 2896 credentialSource.  
 2897 5. Otherwise:  
 2898 1. Let credentialId be the result of serializing and  
 2899 encrypting credentialSource so that only this  
 2900 authenticator can decrypt it.  
 2901 8. If any error occurred while creating the new credential object,  
 2902 return an error code equivalent to "UnknownError" and terminate the  
 2903 operation.  
 2904 9. Let processedExtensions be the result of authenticator extension  
 2905 processing for each supported extension identifier -> authenticator  
 2906 extension input in extensions.  
 2907 10. If the authenticator supports:  
 2908     a per-RP ID signature counter  
 2909         allocate the counter, associate it with the RP ID, and  
 2910         initialize the counter value as zero.  
 2911     a global signature counter  
 2912         Use the global signature counter's actual value when  
 2913         generating authenticator data.  
 2914     a per credential signature counter  
 2915         allocate the counter, associate it with the new  
 2916         credential, and initialize the counter value as zero.  
 2917 11. Let attestedCredentialData be the attested credential data byte  
 2918 array including the credentialId and publicKey.  
 2919 12. Let authenticatorData be the byte array specified in 6.1  
 2920 Authenticator data, including attestedCredentialData as the  
 2921 attestedCredentialData and processedExtensions, if any, as the  
 2922 extensions.  
 2923  
 2924  
 2925  
 2926  
 2927  
 2928  
 2929  
 2930  
 2931  
 2932

2905 consent MUST include user verification.  
 2906 If requireUserPresence is true, the method of obtaining user  
 2907 consent MUST include a test of user presence.  
 2908 If the user does not consent or if user verification fails, return  
 2909 an error code equivalent to "NotAllowedError" and terminate the  
 2910 operation.  
 2911 7. Once user consent has been obtained, generate a new credential  
 2912 object:  
 2913 1. Let (publicKey, privateKey) be a new pair of cryptographic  
 2914 keys using the combination of PublicKeyCredentialType and  
 2915 cryptographic parameters represented by the first item in  
 2916 credTypesAndPubKeyAlgs that is supported by this  
 2917 authenticator.  
 2918 2. Let userHandle be userEntity.id.  
 2919 3. Let credentialSource be a new public key credential source  
 2920 with the fields:  
 2921     type  
 2922         public-key.  
 2923     privateKey  
 2924         privateKey  
 2925     rpId  
 2926         rpEntity.id  
 2927     userHandle  
 2928         userHandle  
 2929     otherUI  
 2930         Any other information the authenticator chooses to  
 2931         include.  
 2932 4. If requireResidentKey is true or the authenticator chooses to  
 2933 create a Client-side-resident Credential Private Key:  
 2934 1. Let credentialId be a new credential id.  
 2935 2. Set credentialSource.id to credentialId.  
 2936 3. Let credentials be this authenticator's credentials map.  
 2937 4. Set credentials[(rpEntity.id, userHandle)] to  
 2938 credentialSource.  
 2939 5. Otherwise:  
 2940 1. Let credentialId be the result of serializing and  
 2941 encrypting credentialSource so that only this  
 2942 authenticator can decrypt it.  
 2943 8. If any error occurred while creating the new credential object,  
 2944 return an error code equivalent to "UnknownError" and terminate the  
 2945 operation.  
 2946 9. Let processedExtensions be the result of authenticator extension  
 2947 processing for each supported extension identifier -> authenticator  
 2948 extension input in extensions.  
 2949 10. If the authenticator supports:  
 2950     a per-RP ID signature counter  
 2951         allocate the counter, associate it with the RP ID, and  
 2952         initialize the counter value as zero.  
 2953     a global signature counter  
 2954         Use the global signature counter's actual value when  
 2955         generating authenticator data.  
 2956     a per credential signature counter  
 2957         allocate the counter, associate it with the new  
 2958         credential, and initialize the counter value as zero.  
 2959 11. Let attestedCredentialData be the attested credential data byte  
 2960 array including the credentialId and publicKey.  
 2961 12. Let authenticatorData be the byte array specified in 6.1  
 2962 Authenticator data, including attestedCredentialData as the  
 2963 attestedCredentialData and processedExtensions, if any, as the  
 2964 extensions.  
 2965  
 2966  
 2967  
 2968  
 2969  
 2970  
 2971  
 2972  
 2973  
 2974

2933 13. Return the attestation object for the new credential created by the  
 2934 procedure specified in 6.3.4 Generating an Attestation Object  
 2935 using an authenticator-chosen attestation statement format,  
 2936 authenticatorData, and hash. For more details on attestation, see  
 2937 6.3 Attestation.  
 2938  
 2939 On successful completion of this operation, the authenticator returns  
 2940 the attestation object to the client.  
 2941  
 2942 6.2.3. The authenticatorGetAssertion operation  
 2943  
 2944 It takes the following input parameters:  
 2945  
 2946 rpld  
 2947 The caller's RP ID, as determined by the user agent and the  
 2948 client.  
 2949  
 2950 hash  
 2951 The hash of the serialized client data, provided by the client.  
 2952  
 2953 allowCredentialDescriptorList  
 2954 An optional list of PublicKeyCredentialDescriptors describing  
 2955 credentials acceptable to the Relying Party (possibly filtered  
 2956 by the client), if any.  
 2957  
 2958 requireUserPresence  
 2959 A Boolean value provided by the client, which in invocations  
 2960 from a WebAuthn Client's [[DiscoverFromExternalSource]](origin,  
 2961 options, sameOriginWithAncestors) method is always set to the  
 2962 inverse of requireUserVerification.  
 2963  
 2964 requireUserVerification  
 2965 The effective user verification requirement for assertion, a  
 2966 Boolean value provided by the client.  
 2967  
 2968 extensions  
 2969 A CBOR map from extension identifiers to their authenticator  
 2970 extension inputs, created by the client based on the extensions  
 2971 requested by the Relying Party, if any.  
 2972  
 2973 Note: Before performing this operation, all other operations in  
 2974 progress in the authenticator session must be aborted by running the  
 2975 authenticatorCancel operation.  
 2976  
 2977 When this method is invoked, the authenticator MUST perform the  
 2978 following procedure:  
 2979 1. Check if all the supplied parameters are syntactically well-formed  
 2980 and of the correct length. If not, return an error code equivalent  
 2981 to "UnknownError" and terminate the operation.  
 2982 2. Let credentialOptions be a new empty set of public key credential  
 2983 sources.  
 2984 3. If allowCredentialDescriptorList was supplied, then for each  
 2985 descriptor of allowCredentialDescriptorList:  
 2986 1. Let credSource be the result of looking up descriptor.id in  
 2987 this authenticator.  
 2988 2. If credSource is not null, append it to credentialOptions.  
 2989 4. Otherwise (allowCredentialDescriptorList was not supplied), for  
 2990 each key -> credSource of this authenticator's credentials map,  
 2991 append credSource to credentialOptions.  
 2992 5. Remove any items from credentialOptions whose rpld is not equal to  
 2993 rpld.  
 2994 6. If credentialOptions is now empty, return an error code equivalent  
 2995 to "NotAllowedError" and terminate the operation.  
 2996 7. Prompt the user to select a public key credential source  
 2997 selectedCredential from credentialOptions. Obtain user consent for  
 2998 using selectedCredential. The prompt for obtaining this consent may  
 2999 be shown by the authenticator if it has its own output capability,  
 3000 or by the user agent otherwise.  
 3001 If requireUserVerification is true, the method of obtaining user  
 3002 consent MUST include user verification.

2975 13. Return the attestation object for the new credential created by the  
 2976 procedure specified in 6.3.4 Generating an Attestation Object  
 2977 using an authenticator-chosen attestation statement format,  
 2978 authenticatorData, and hash. For more details on attestation, see  
 2979 6.3 Attestation.  
 2980  
 2981 On successful completion of this operation, the authenticator returns  
 2982 the attestation object to the client.  
 2983  
 2984 6.2.3. The authenticatorGetAssertion operation  
 2985  
 2986 It takes the following input parameters:  
 2987  
 2988 rpld  
 2989 The caller's RP ID, as determined by the user agent and the  
 2990 client.  
 2991  
 2992 hash  
 2993 The hash of the serialized client data, provided by the client.  
 2994  
 2995 allowCredentialDescriptorList  
 2996 An optional list of PublicKeyCredentialDescriptors describing  
 2997 credentials acceptable to the Relying Party (possibly filtered  
 2998 by the client), if any.  
 2999  
 3000 requireUserPresence  
 3001 A Boolean value provided by the client, which in invocations  
 3002 from a WebAuthn Client's [[DiscoverFromExternalSource]](origin,  
 3003 options, sameOriginWithAncestors) method is always set to the  
 3004 inverse of requireUserVerification.  
 3005  
 3006 requireUserVerification  
 3007 The effective user verification requirement for assertion, a  
 3008 Boolean value provided by the client.  
 3009  
 3010 extensions  
 3011 A CBOR map from extension identifiers to their authenticator  
 3012 extension inputs, created by the client based on the extensions  
 3013 requested by the Relying Party, if any.  
 3014  
 3015 Note: Before performing this operation, all other operations in  
 3016 progress in the authenticator session must be aborted by running the  
 3017 authenticatorCancel operation.  
 3018  
 3019 When this method is invoked, the authenticator MUST perform the  
 3020 following procedure:  
 3021 1. Check if all the supplied parameters are syntactically well-formed  
 3022 and of the correct length. If not, return an error code equivalent  
 3023 to "UnknownError" and terminate the operation.  
 3024 2. Let credentialOptions be a new empty set of public key credential  
 3025 sources.  
 3026 3. If allowCredentialDescriptorList was supplied, then for each  
 3027 descriptor of allowCredentialDescriptorList:  
 3028 1. Let credSource be the result of looking up descriptor.id in  
 3029 this authenticator.  
 3030 2. If credSource is not null, append it to credentialOptions.  
 3031 4. Otherwise (allowCredentialDescriptorList was not supplied), for  
 3032 each key -> credSource of this authenticator's credentials map,  
 3033 append credSource to credentialOptions.  
 3034 5. Remove any items from credentialOptions whose rpld is not equal to  
 3035 rpld.  
 3036 6. If credentialOptions is now empty, return an error code equivalent  
 3037 to "NotAllowedError" and terminate the operation.  
 3038 7. Prompt the user to select a public key credential source  
 3039 selectedCredential from credentialOptions. Obtain user consent for  
 3040 using selectedCredential. The prompt for obtaining this consent may  
 3041 be shown by the authenticator if it has its own output capability,  
 3042 or by the user agent otherwise.  
 3043 If requireUserVerification is true, the method of obtaining user  
 3044 consent MUST include user verification.

3003 If requireUserPresence is true, the method of obtaining user  
3004 consent MUST include a test of user presence.  
3005 If the user does not consent, return an error code equivalent to  
3006 "NotAllowedError" and terminate the operation.  
3007 8. Let processedExtensions be the result of authenticator extension  
3008 processing for each supported extension identifier -> authenticator  
3009 extension input in extensions.  
3010 9. Increment the RP ID-associated signature counter or the global  
3011 signature counter value, depending on which approach is implemented  
3012 by the authenticator, by some positive value.  
3013 10. Let authenticatorData be the byte array specified in 6.1  
3014 Authenticator data including processedExtensions, if any, as the  
3015 extensions and excluding attestedCredentialData.  
3016 11. Let signature be the assertion signature of the concatenation  
3017 authenticatorData || hash using the privateKey of  
3018 selectedCredential as shown in Figure 2, below. A simple,  
3019 undelimited concatenation is safe to use here because the  
3020 authenticator data describes its own length. The hash of the  
3021 serialized client data (which potentially has a variable length) is  
3022 always the last element.  
3023 [fido-signature-formats-figure2.svg] Generating an assertion  
3024 signature.  
3025 12. If any error occurred while generating the assertion signature,  
3026 return an error code equivalent to "UnknownError" and terminate the  
3027 operation.  
3028 13. Return to the user agent:  
3029 + selectedCredential.id, if either a list of credentials (i.e.,  
3030 allowCredentialDescriptorList) of length 2 or greater was  
3031 supplied by the client, or no such list was supplied.  
3032 Note: If, within allowCredentialDescriptorList, the client  
3033 supplied exactly one credential and it was successfully  
3034 employed, then its credential ID is not returned since the  
3035 client already knows it. This saves transmitting these bytes  
3036 over what may be a constrained connection in what is likely a  
3037 common case.  
3038 + authenticatorData  
3039 + signature  
3040 + selectedCredential.userHandle  
3041 Note: the returned userHandle value may be null, see:  
3042 userHandleResult.  
3043  
3044 If the authenticator cannot find any credential corresponding to the  
3045 specified Relying Party that matches the specified criteria, it  
3046 terminates the operation and returns an error.  
3047  
3048 6.2.4. The authenticatorCancel operation  
3049  
3050 This operation takes no input parameters and returns no result.  
3051  
3052 When this operation is invoked by the client in an authenticator  
3053 session, it has the effect of terminating any  
3054 authenticatorMakeCredential or authenticatorGetAssertion operation  
3055 currently in progress in that authenticator session. The authenticator  
3056 stops prompting for, or accepting, any user input related to  
3057 authorizing the canceled operation. The client ignores any further  
3058 responses from the authenticator for the canceled operation.  
3059  
3060 This operation is ignored if it is invoked in an authenticator session  
3061 which does not have an authenticatorMakeCredential or  
3062 authenticatorGetAssertion operation currently in progress.  
3063  
3064 6.3. Attestation  
3065  
3066 Authenticators MUST also provide some form of attestation. The basic  
3067 requirement is that the authenticator can produce, for each credential  
3068 public key, an attestation statement verifiable by the Relying Party.  
3069 Typically, this attestation statement contains a signature by an  
3070 attestation private key over the attested credential public key and a  
3071 challenge, as well as a certificate or similar data providing  
3072 provenance information for the attestation public key, enabling the

3045 If requireUserPresence is true, the method of obtaining user  
3046 consent MUST include a test of user presence.  
3047 If the user does not consent, return an error code equivalent to  
3048 "NotAllowedError" and terminate the operation.  
3049 8. Let processedExtensions be the result of authenticator extension  
3050 processing for each supported extension identifier -> authenticator  
3051 extension input in extensions.  
3052 9. Increment the RP ID-associated signature counter or the global  
3053 signature counter value, depending on which approach is implemented  
3054 by the authenticator, by some positive value.  
3055 10. Let authenticatorData be the byte array specified in 6.1  
3056 Authenticator data including processedExtensions, if any, as the  
3057 extensions and excluding attestedCredentialData.  
3058 11. Let signature be the assertion signature of the concatenation  
3059 authenticatorData || hash using the privateKey of  
3060 selectedCredential as shown in Figure 2, below. A simple,  
3061 undelimited concatenation is safe to use here because the  
3062 authenticator data describes its own length. The hash of the  
3063 serialized client data (which potentially has a variable length) is  
3064 always the last element.  
3065 [fido-signature-formats-figure2.svg] Generating an assertion  
3066 signature.  
3067 12. If any error occurred while generating the assertion signature,  
3068 return an error code equivalent to "UnknownError" and terminate the  
3069 operation.  
3070 13. Return to the user agent:  
3071 + selectedCredential.id, if either a list of credentials (i.e.,  
3072 allowCredentialDescriptorList) of length 2 or greater was  
3073 supplied by the client, or no such list was supplied.  
3074 Note: If, within allowCredentialDescriptorList, the client  
3075 supplied exactly one credential and it was successfully  
3076 employed, then its credential ID is not returned since the  
3077 client already knows it. This saves transmitting these bytes  
3078 over what may be a constrained connection in what is likely a  
3079 common case.  
3080 + authenticatorData  
3081 + signature  
3082 + selectedCredential.userHandle  
3083 Note: the returned userHandle value may be null, see:  
3084 userHandleResult.  
3085  
3086 If the authenticator cannot find any credential corresponding to the  
3087 specified Relying Party that matches the specified criteria, it  
3088 terminates the operation and returns an error.  
3089  
3090 6.2.4. The authenticatorCancel operation  
3091  
3092 This operation takes no input parameters and returns no result.  
3093  
3094 When this operation is invoked by the client in an authenticator  
3095 session, it has the effect of terminating any  
3096 authenticatorMakeCredential or authenticatorGetAssertion operation  
3097 currently in progress in that authenticator session. The authenticator  
3098 stops prompting for, or accepting, any user input related to  
3099 authorizing the canceled operation. The client ignores any further  
3100 responses from the authenticator for the canceled operation.  
3101  
3102 This operation is ignored if it is invoked in an authenticator session  
3103 which does not have an authenticatorMakeCredential or  
3104 authenticatorGetAssertion operation currently in progress.  
3105  
3106 6.3. Attestation  
3107  
3108 Authenticators MUST also provide some form of attestation. The basic  
3109 requirement is that the authenticator can produce, for each credential  
3110 public key, an attestation statement verifiable by the Relying Party.  
3111 Typically, this attestation statement contains a signature by an  
3112 attestation private key over the attested credential public key and a  
3113 challenge, as well as a certificate or similar data providing  
3114 provenance information for the attestation public key, enabling the

3073 Relying Party to make a trust decision. However, if an attestation key  
3074 pair is not available, then the authenticator MUST perform self  
3075 attestation of the credential public key with the corresponding  
3076 credential private key. All this information is returned by  
3077 authenticators any time a new public key credential is generated, in  
3078 the overall form of an attestation object. The relationship of the  
3079 attestation object with authenticator data (containing attested  
3080 credential data) and the attestation statement is illustrated in figure  
3081 3, below.

3082 [fido-attestation-structures.svg] Attestation object layout  
3083 illustrating the included authenticator data (containing attested  
3084 credential data) and the attestation statement.

3085  
3086 This figure illustrates only the packed attestation statement format.  
3087 Several additional attestation statement formats are defined in 8  
3088 Defined Attestation Statement Formats.

3089  
3090 An important component of the attestation object is the attestation  
3091 statement. This is a specific type of signed data object, containing  
3092 statements about a public key credential itself and the authenticator  
3093 that created it. It contains an attestation signature created using the  
3094 key of the attesting authority (except for the case of self  
3095 attestation, when it is created using the credential private key). In  
3096 order to correctly interpret an attestation statement, a Relying Party  
3097 needs to understand these two aspects of attestation:

- 3098  
3099 1. The attestation statement format is the manner in which the  
3100 signature is represented and the various contextual bindings are  
3101 incorporated into the attestation statement by the authenticator.  
3102 In other words, this defines the syntax of the statement. Various  
3103 existing devices and platforms (such as TPMs and the Android OS)  
3104 have previously defined attestation statement formats. This  
3105 specification supports a variety of such formats in an extensible  
3106 way, as defined in 6.3.2 Attestation Statement Formats.  
3107 2. The attestation type defines the semantics of attestation  
3108 statements and their underlying trust models. Specifically, it  
3109 defines how a Relying Party establishes trust in a particular  
3110 attestation statement, after verifying that it is cryptographically  
3111 valid. This specification supports a number of attestation types,  
3112 as described in 6.3.3 Attestation Types.

3113 In general, there is no simple mapping between attestation statement  
3114 formats and attestation types. For example, the "packed" attestation  
3115 statement format defined in 8.2 Packed Attestation Statement Format  
3116 can be used in conjunction with all attestation types, while other  
3117 formats and types have more limited applicability.

3118  
3119 The privacy, security and operational characteristics of attestation  
3120 depend on:

- 3121 \* The attestation type, which determines the trust model,  
3122 \* The attestation statement format, which MAY constrain the strength  
3123 of the attestation by limiting what can be expressed in an  
3124 attestation statement, and  
3125 \* The characteristics of the individual authenticator, such as its  
3126 construction, whether part or all of it runs in a secure operating  
3127 environment, and so on.

3128  
3129 It is expected that most authenticators will support a small number of  
3130 attestation types and attestation statement formats, while Relying  
3131 Parties will decide what attestation types are acceptable to them by  
3132 policy. Relying Parties will also need to understand the  
3133 characteristics of the authenticators that they trust, based on  
3134 information they have about these authenticators. For example, the FIDO  
3135 Metadata Service [FIDOMetadataService] provides one way to access such  
3136 information.

3137  
3138 6.3.1. Attested credential data

3139  
3140 Attested credential data is a variable-length byte array added to the  
3141 authenticator data when generating an attestation object for a given  
3142 credential. It has the following format:

3115 Relying Party to make a trust decision. However, if an attestation key  
3116 pair is not available, then the authenticator MUST perform self  
3117 attestation of the credential public key with the corresponding  
3118 credential private key. All this information is returned by  
3119 authenticators any time a new public key credential is generated, in  
3120 the overall form of an attestation object. The relationship of the  
3121 attestation object with authenticator data (containing attested  
3122 credential data) and the attestation statement is illustrated in figure  
3123 3, below.

3124 [fido-attestation-structures.svg] Attestation object layout  
3125 illustrating the included authenticator data (containing attested  
3126 credential data) and the attestation statement.

3127  
3128 This figure illustrates only the packed attestation statement format.  
3129 Several additional attestation statement formats are defined in 8  
3130 Defined Attestation Statement Formats.

3131  
3132 An important component of the attestation object is the attestation  
3133 statement. This is a specific type of signed data object, containing  
3134 statements about a public key credential itself and the authenticator  
3135 that created it. It contains an attestation signature created using the  
3136 key of the attesting authority (except for the case of self  
3137 attestation, when it is created using the credential private key). In  
3138 order to correctly interpret an attestation statement, a Relying Party  
3139 needs to understand these two aspects of attestation:

- 3140  
3141 1. The attestation statement format is the manner in which the  
3142 signature is represented and the various contextual bindings are  
3143 incorporated into the attestation statement by the authenticator.  
3144 In other words, this defines the syntax of the statement. Various  
3145 existing devices and platforms (such as TPMs and the Android OS)  
3146 have previously defined attestation statement formats. This  
3147 specification supports a variety of such formats in an extensible  
3148 way, as defined in 6.3.2 Attestation Statement Formats.  
3149 2. The attestation type defines the semantics of attestation  
3150 statements and their underlying trust models. Specifically, it  
3151 defines how a Relying Party establishes trust in a particular  
3152 attestation statement, after verifying that it is cryptographically  
3153 valid. This specification supports a number of attestation types,  
3154 as described in 6.3.3 Attestation Types.

3155 In general, there is no simple mapping between attestation statement  
3156 formats and attestation types. For example, the "packed" attestation  
3157 statement format defined in 8.2 Packed Attestation Statement Format  
3158 can be used in conjunction with all attestation types, while other  
3159 formats and types have more limited applicability.

3160  
3161 The privacy, security and operational characteristics of attestation  
3162 depend on:

- 3163 \* The attestation type, which determines the trust model,  
3164 \* The attestation statement format, which MAY constrain the strength  
3165 of the attestation by limiting what can be expressed in an  
3166 attestation statement, and  
3167 \* The characteristics of the individual authenticator, such as its  
3168 construction, whether part or all of it runs in a secure operating  
3169 environment, and so on.

3170  
3171 It is expected that most authenticators will support a small number of  
3172 attestation types and attestation statement formats, while Relying  
3173 Parties will decide what attestation types are acceptable to them by  
3174 policy. Relying Parties will also need to understand the  
3175 characteristics of the authenticators that they trust, based on  
3176 information they have about these authenticators. For example, the FIDO  
3177 Metadata Service [FIDOMetadataService] provides one way to access such  
3178 information.

3179  
3180 6.3.1. Attested credential data

3181  
3182 Attested credential data is a variable-length byte array added to the  
3183 authenticator data when generating an attestation object for a given  
3184 credential. It has the following format:

3143 Name Length (in bytes) Description  
 3144 aaguid 16 The AAGUID of the authenticator.  
 3145 credentialIdLength 2 Byte length L of Credential ID, 16-bit unsigned  
 3146 big-endian integer.  
 3147 credentialId L Credential ID  
 3148 credentialPublicKey variable The credential public key encoded in  
 3149 COSE\_Key format, as defined in Section 7 of [RFC8152], using the CTAP2  
 3150 canonical CBOR encoding form. The COSE\_Key-encoded credential public  
 3151 key MUST contain the optional "alg" parameter and MUST NOT contain any  
 3152 other optional parameters. The "alg" parameter MUST contain a  
 3153 COSEAlgorithmIdentifier value. The encoded credential public key MUST  
 3154 also contain any additional required parameters stipulated by the  
 3155 relevant key type specification, i.e., required for the key type "kty"  
 3156 and algorithm "alg" (see Section 8 of [RFC8152]).  
 3157  
 3158 NOTE: The names in the Name column in the above table are only for  
 3159 reference within this document, and are not present in the actual  
 3160 representation of the attested credential data.  
 3161  
 3162 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE\_Key format  
 3163  
 3164 This section provides examples of COSE\_Key-encoded Elliptic Curve and  
 3165 RSA public keys for the ES256, PS256, and RS256 signature algorithms.  
 3166 These examples adhere to the rules defined above for the  
 3167 credentialPublicKey value, and are presented in [CDDL] for clarity.  
 3168  
 3169 [RFC8152] Section 7 defines the general framework for all  
 3170 COSE\_Key-encoded keys. Specific key types for specific algorithms are  
 3171 defined in other sections of [RFC8152] as well as in other  
 3172 specifications, as noted below.  
 3173  
 3174 Below is an example of a COSE\_Key-encoded Elliptic Curve public key in  
 3175 EC2 format (see [RFC8152] Section 13.1), on the P-256 curve, to be used  
 3176 with the ES256 signature algorithm (ECDSA w/ SHA-256, see [RFC8152]  
 3177 Section 8.1):  
 3178 {  
 3179 1: 2, ; kty: EC2 key type  
 3180 3: -7, ; alg: ES256 signature algorithm  
 3181 -1: 1, ; crv: P-256 curve  
 3182 -2: x, ; x-coordinate as byte string 32 bytes in length  
 3183 ; e.g., in hex: 65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de108d  
 3184 e439c08551d  
 3185 -3: y, ; y-coordinate as byte string 32 bytes in length  
 3186 ; e.g., in hex: 1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9e  
 3187 ecd0084d19c  
 3188 }  
 3189  
 3190 Below is the above Elliptic Curve public key encoded in the CTAP2  
 3191 canonical CBOR encoding form, whitespace and line breaks are included  
 3192 here for clarity and to match the [CDDL] presentation above:  
 3193 A5  
 3194 01 02  
 3195  
 3196 03 26  
 3197  
 3198 20 01  
 3199  
 3200 21 58 20 65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de108de439c08551d  
 3201  
 3202 22 58 20 1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9eecd0084d19c  
 3203  
 3204 Below is an example of a COSE\_Key-encoded 2048-bit RSA public key (see  
 3205 [RFC8230] Section 4), to be used with the PS256 signature algorithm  
 3206 (RSASSA-PSS with SHA-256, see [RFC8230] Section 2):  
 3207 {  
 3208 1: 3, ; kty: RSA key type  
 3209 3: -37, ; alg: PS256  
 3210 -1: n, ; n: RSA modulus n byte string 256 bytes in length  
 3211 ; e.g., in hex (middle bytes elided for brevity): DB5F651550...6  
 3212

3185 Name Length (in bytes) Description  
 3186 aaguid 16 The AAGUID of the authenticator.  
 3187 credentialIdLength 2 Byte length L of Credential ID, 16-bit unsigned  
 3188 big-endian integer.  
 3189 credentialId L Credential ID  
 3190 credentialPublicKey variable The credential public key encoded in  
 3191 COSE\_Key format, as defined in Section 7 of [RFC8152], using the CTAP2  
 3192 canonical CBOR encoding form. The COSE\_Key-encoded credential public  
 3193 key MUST contain the optional "alg" parameter and MUST NOT contain any  
 3194 other optional parameters. The "alg" parameter MUST contain a  
 3195 COSEAlgorithmIdentifier value. The encoded credential public key MUST  
 3196 also contain any additional required parameters stipulated by the  
 3197 relevant key type specification, i.e., required for the key type "kty"  
 3198 and algorithm "alg" (see Section 8 of [RFC8152]).  
 3199  
 3200 NOTE: The names in the Name column in the above table are only for  
 3201 reference within this document, and are not present in the actual  
 3202 representation of the attested credential data.  
 3203  
 3204 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE\_Key format  
 3205  
 3206 This section provides examples of COSE\_Key-encoded Elliptic Curve and  
 3207 RSA public keys for the ES256, PS256, and RS256 signature algorithms.  
 3208 These examples adhere to the rules defined above for the  
 3209 credentialPublicKey value, and are presented in [CDDL] for clarity.  
 3210  
 3211 [RFC8152] Section 7 defines the general framework for all  
 3212 COSE\_Key-encoded keys. Specific key types for specific algorithms are  
 3213 defined in other sections of [RFC8152] as well as in other  
 3214 specifications, as noted below.  
 3215  
 3216 Below is an example of a COSE\_Key-encoded Elliptic Curve public key in  
 3217 EC2 format (see [RFC8152] Section 13.1), on the P-256 curve, to be used  
 3218 with the ES256 signature algorithm (ECDSA w/ SHA-256, see [RFC8152]  
 3219 Section 8.1):  
 3220 {  
 3221 1: 2, ; kty: EC2 key type  
 3222 3: -7, ; alg: ES256 signature algorithm  
 3223 -1: 1, ; crv: P-256 curve  
 3224 -2: x, ; x-coordinate as byte string 32 bytes in length  
 3225 ; e.g., in hex: 65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de108d  
 3226 e439c08551d  
 3227 -3: y, ; y-coordinate as byte string 32 bytes in length  
 3228 ; e.g., in hex: 1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9e  
 3229 ecd0084d19c  
 3230 }  
 3231  
 3232 Below is the above Elliptic Curve public key encoded in the CTAP2  
 3233 canonical CBOR encoding form, whitespace and line breaks are included  
 3234 here for clarity and to match the [CDDL] presentation above:  
 3235 A5  
 3236 01 02  
 3237  
 3238 03 26  
 3239  
 3240 20 01  
 3241  
 3242 21 58 20 65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de108de439c08551d  
 3243  
 3244 22 58 20 1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9eecd0084d19c  
 3245  
 3246 Below is an example of a COSE\_Key-encoded 2048-bit RSA public key (see  
 3247 [RFC8230] Section 4), to be used with the PS256 signature algorithm  
 3248 (RSASSA-PSS with SHA-256, see [RFC8230] Section 2):  
 3249 {  
 3250 1: 3, ; kty: RSA key type  
 3251 3: -37, ; alg: PS256  
 3252 -1: n, ; n: RSA modulus n byte string 256 bytes in length  
 3253 ; e.g., in hex (middle bytes elided for brevity): DB5F651550...6  
 3254

```

3213 DC6548ACC3
3214 -2: e ; e: RSA public exponent e byte string 3 bytes in length
3215 ; e.g., in hex: 010001
3216 }
3217
3218 Below is an example of the same COSE_Key-encoded RSA public key as
3219 above, to be used with the RS256 signature algorithm (RSASSA-PKCS1-v1_5
3220 with SHA-256, see 11.3 COSE Algorithm Registrations):
3221 {
3222 1: 3, ; kty: RSA key type
3223 3:-257, ; alg: RS256
3224 -1: n, ; n: RSA modulus n byte string 256 bytes in length
3225 ; e.g., in hex (middle bytes elided for brevity): DB5F651550...6
3226 DC6548ACC3
3227 -2: e ; e: RSA public exponent e byte string 3 bytes in length
3228 ; e.g., in hex: 010001
3229 }

```

### 6.3.2. Attestation Statement Formats

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

- \* Attestation statement format identifier:
  - \* Supported attestation types:
  - \* Syntax: The syntax of an attestation statement produced in this format, defined using [CDDL] for the extension point \$attStmtFormat defined in 6.3.4 Generating an Attestation Object.
  - \* Signing procedure: The signing procedure for computing an attestation statement in this format given the public key credential to be attested, the authenticator data structure containing the authenticator data for the attestation, and the hash of the serialized client data.
  - \* Verification procedure: The procedure for verifying an attestation statement, which takes the following verification procedure inputs:
    - + attStmt: The attestation statement structure
    - + authenticatorData: The authenticator data claimed to have been used for the attestation
    - + clientDataHash: The hash of the serialized client data
- The procedure returns either:
- + An error indicating that the attestation is invalid, or
  - + The attestation type, and the trust path. This attestation trust path is either empty (in case of self attestation), an identifier of an ECDAA-Issuer public key (in the case of ECDAA), or a set of X.509 certificates.

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

### 6.3.3. Attestation Types

WebAuthn supports multiple attestation types:

#### Basic Attestation (Basic)

In the case of basic attestation [UAFProtocol], the authenticator's attestation key pair is specific to an authenticator model. Thus, authenticators of the same model often share the same attestation key pair. See 14.1 Attestation Privacy for further information.

#### Self Attestation (Self)

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

#### Attestation CA (AttCA)

```

3255 DC6548ACC3
3256 -2: e ; e: RSA public exponent e byte string 3 bytes in length
3257 ; e.g., in hex: 010001
3258 }
3259
3260 Below is an example of the same COSE_Key-encoded RSA public key as
3261 above, to be used with the RS256 signature algorithm (RSASSA-PKCS1-v1_5
3262 with SHA-256, see 11.3 COSE Algorithm Registrations):
3263 {
3264 1: 3, ; kty: RSA key type
3265 3:-257, ; alg: RS256
3266 -1: n, ; n: RSA modulus n byte string 256 bytes in length
3267 ; e.g., in hex (middle bytes elided for brevity): DB5F651550...6
3268 DC6548ACC3
3269 -2: e ; e: RSA public exponent e byte string 3 bytes in length
3270 ; e.g., in hex: 010001
3271 }

```

### 6.3.2. Attestation Statement Formats

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

- \* Attestation statement format identifier:
  - \* Supported attestation types:
  - \* Syntax: The syntax of an attestation statement produced in this format, defined using [CDDL] for the extension point \$attStmtFormat defined in 6.3.4 Generating an Attestation Object.
  - \* Signing procedure: The signing procedure for computing an attestation statement in this format given the public key credential to be attested, the authenticator data structure containing the authenticator data for the attestation, and the hash of the serialized client data.
  - \* Verification procedure: The procedure for verifying an attestation statement, which takes the following verification procedure inputs:
    - + attStmt: The attestation statement structure
    - + authenticatorData: The authenticator data claimed to have been used for the attestation
    - + clientDataHash: The hash of the serialized client data
- The procedure returns either:
- + An error indicating that the attestation is invalid, or
  - + The attestation type, and the trust path. This attestation trust path is either empty (in case of self attestation), an identifier of an ECDAA-Issuer public key (in the case of ECDAA), or a set of X.509 certificates.

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

### 6.3.3. Attestation Types

WebAuthn supports multiple attestation types:

#### Basic Attestation (Basic)

In the case of basic attestation [UAFProtocol], the authenticator's attestation key pair is specific to an authenticator model. Thus, authenticators of the same model often share the same attestation key pair. See 14.1 Attestation Privacy for further information.

#### Self Attestation (Self)

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

#### Attestation CA (AttCA)



3283 In this case, an authenticator is based on a Trusted Platform  
3284 Module (TPM) and holds an authenticator-specific "endorsement  
3285 key" (EK). This key is used to securely communicate with a  
3286 trusted third party, the Attestation CA  
3287 [TCG-CMCPProfile-AIKCertEnroll] (formerly known as a "Privacy  
3288 CA"). The authenticator can generate multiple attestation  
3289 identity key pairs (AIK) and requests an Attestation CA to issue  
3290 an AIK certificate for each. Using this approach, such an  
3291 authenticator can limit the exposure of the EK (which is a  
3292 global correlation handle) to Attestation CA(s). AIKs can be  
3293 requested for each authenticator-generated public key credential  
3294 individually, and conveyed to Relying Parties as attestation  
3295 certificates.

3296  
3297 Note: This concept typically leads to multiple attestation  
3298 certificates. The attestation certificate requested most  
3299 recently is called "active".  
3300

3301 Elliptic Curve based Direct Anonymous Attestation (ECDAA)

3302 In this case, the Authenticator receives direct anonymous  
3303 attestation (DAA) credentials from a single DAA-Issuer. These  
3304 DAA credentials are used along with blinding to sign the  
3305 attested credential data. The concept of blinding avoids the DAA  
3306 credentials being misused as global correlation handle. WebAuthn  
3307 supports DAA using elliptic curve cryptography and bilinear  
3308 pairings, called ECDAA (see [FIDOEcdaaAlgorithm]) in this  
3309 specification. Consequently we denote the DAA-Issuer as  
3310 ECDAA-Issuer (see [FIDOEcdaaAlgorithm]).  
3311

3312 No attestation statement (None)

3313 In this case, no attestation information is available.  
3314

3315 6.3.4. Generating an Attestation Object

3316 To generate an attestation object (see: Figure 3) given:

3317 attestationFormat

3318 An attestation statement format.  
3319

3320 authData

3321 A byte array containing authenticator data.  
3322

3323 hash

3324 The hash of the serialized client data.  
3325

3326 the authenticator MUST:

- 3327 1. Let attStmt be the result of running attestationFormat's signing  
3328 procedure given authData and hash.
- 3329 2. Let fmt be attestationFormat's attestation statement format  
3330 identifier
- 3331 3. Return the attestation object as a CBOR map with the following  
3332 syntax, filled in with variables initialized by this algorithm:

```
3333 attObj = {  
3334   authData: bytes,  
3335   $$attStmtType  
3336 }
```

```
3337 attStmtTemplate = (  
3338   fmt: text,  
3339   attStmt: { * tstr => any } ; Map is filled in by each  
3340 concrete attStmtType  
3341 )
```

3342 ; Every attestation statement format must have the above fields  
3343 attStmtTemplate .within \$\$attStmtType  
3344

3345 6.3.5. Signature Formats for Packed Attestation, FIDO U2F Attestation, and  
3346 Assertion Signatures

3347 \* For COSEAlgorithmIdentifier -7 (ES256), and other ECDSA-based  
3348  
3349

3325 In this case, an authenticator is based on a Trusted Platform  
3326 Module (TPM) and holds an authenticator-specific "endorsement  
3327 key" (EK). This key is used to securely communicate with a  
3328 trusted third party, the Attestation CA  
3329 [TCG-CMCPProfile-AIKCertEnroll] (formerly known as a "Privacy  
3330 CA"). The authenticator can generate multiple attestation  
3331 identity key pairs (AIK) and requests an Attestation CA to issue  
3332 an AIK certificate for each. Using this approach, such an  
3333 authenticator can limit the exposure of the EK (which is a  
3334 global correlation handle) to Attestation CA(s). AIKs can be  
3335 requested for each authenticator-generated public key credential  
3336 individually, and conveyed to Relying Parties as attestation  
3337 certificates.

3338  
3339 Note: This concept typically leads to multiple attestation  
3340 certificates. The attestation certificate requested most  
3341 recently is called "active".  
3342

3343 Elliptic Curve based Direct Anonymous Attestation (ECDAA)

3344 In this case, the Authenticator receives direct anonymous  
3345 attestation (DAA) credentials from a single DAA-Issuer. These  
3346 DAA credentials are used along with blinding to sign the  
3347 attested credential data. The concept of blinding avoids the DAA  
3348 credentials being misused as global correlation handle. WebAuthn  
3349 supports DAA using elliptic curve cryptography and bilinear  
3350 pairings, called ECDAA (see [FIDOEcdaaAlgorithm]) in this  
3351 specification. Consequently we denote the DAA-Issuer as  
3352 ECDAA-Issuer (see [FIDOEcdaaAlgorithm]).  
3353

3354 No attestation statement (None)

3355 In this case, no attestation information is available.  
3356

3357 6.3.4. Generating an Attestation Object

3358 To generate an attestation object (see: Figure 3) given:

3359 attestationFormat

3360 An attestation statement format.  
3361

3362 authData

3363 A byte array containing authenticator data.  
3364

3365 hash

3366 The hash of the serialized client data.  
3367

3368 the authenticator MUST:

- 3369 1. Let attStmt be the result of running attestationFormat's signing  
3370 procedure given authData and hash.
- 3371 2. Let fmt be attestationFormat's attestation statement format  
3372 identifier
- 3373 3. Return the attestation object as a CBOR map with the following  
3374 syntax, filled in with variables initialized by this algorithm:

```
3375 attObj = {  
3376   authData: bytes,  
3377   $$attStmtType  
3378 }
```

```
3379 attStmtTemplate = (  
3380   fmt: text,  
3381   attStmt: { * tstr => any } ; Map is filled in by each  
3382 concrete attStmtType  
3383 )
```

3384 ; Every attestation statement format must have the above fields  
3385 attStmtTemplate .within \$\$attStmtType  
3386

3387 6.3.5. Signature Formats for Packed Attestation, FIDO U2F Attestation, and  
3388 Assertion Signatures

3389 \* For COSEAlgorithmIdentifier -7 (ES256), and other ECDSA-based  
3390  
3391

```

3353 algorithms, a signature value is encoded as an ASN.1 DER
3354 Ecdsa-Sig-Value, as defined in [RFC3279] section 2.2.3.
3355 Example:
3356 30 44                ; SEQUENCE (68 Bytes)
3357 02 20                ; INTEGER (32 Bytes)
3358 | 3d 46 28 7b 8c 6e 8c 8c 26 1c 1b 88 f2 73 b0 9a
3359 | 32 a6 cf 28 09 fd 6e 30 d5 a7 9f 26 37 00 8f 54
3360 02 20                ; INTEGER (32 Bytes)
3361 | 4e 72 23 6e a3 90 a9 a1 7b cf 5f 7a 09 d6 3a b2
3362 | 17 6c 92 bb 8e 36 c0 41 98 a2 7b 90 9b 6e 8f 13
3363

```

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

- \* For COSEAlgorithmIdentifier -257 (RS256), sig contains the signature generated using the RSASSA-PKCS1-v1\_5 signature scheme defined in section 8.2.1 in [RFC8017] with SHA-256 as the hash function. The signature is not ASN.1 wrapped.
- \* For COSEAlgorithmIdentifier -37 (PS256), sig contains the signature generated using the RSASSA-PSS signature scheme defined in section 8.1.1 in [RFC8017] with SHA-256 as the hash function. The signature is not ASN.1 wrapped.

## 7. Relying Party Operations

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

### 7.1. Registering a new credential

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

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

```

3395 algorithms, a signature value is encoded as an ASN.1 DER
3396 Ecdsa-Sig-Value, as defined in [RFC3279] section 2.2.3.
3397 Example:
3398 30 44                ; SEQUENCE (68 Bytes)
3399 02 20                ; INTEGER (32 Bytes)
3400 | 3d 46 28 7b 8c 6e 8c 8c 26 1c 1b 88 f2 73 b0 9a
3401 | 32 a6 cf 28 09 fd 6e 30 d5 a7 9f 26 37 00 8f 54
3402 02 20                ; INTEGER (32 Bytes)
3403 | 4e 72 23 6e a3 90 a9 a1 7b cf 5f 7a 09 d6 3a b2
3404 | 17 6c 92 bb 8e 36 c0 41 98 a2 7b 90 9b 6e 8f 13
3405

```

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

- \* For COSEAlgorithmIdentifier -257 (RS256), sig contains the signature generated using the RSASSA-PKCS1-v1\_5 signature scheme defined in section 8.2.1 in [RFC8017] with SHA-256 as the hash function. The signature is not ASN.1 wrapped.
- \* For COSEAlgorithmIdentifier -37 (PS256), sig contains the signature generated using the RSASSA-PSS signature scheme defined in section 8.1.1 in [RFC8017] with SHA-256 as the hash function. The signature is not ASN.1 wrapped.

## 7. Relying Party Operations

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

### 7.1. Registering a new credential

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

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

3423 attestation statement format fmt, the authenticator data authData,  
3424 and the attestation statement attStmt.  
3425 9. Verify that the RP ID hash in authData is indeed the SHA-256 hash  
3426 of the RP ID expected by the RP.  
3427 10. Verify that the User Present bit of the flags in authData is set.  
3428 11. If user verification is required for this registration, verify that  
3429 the User Verified bit of the flags in authData is set.  
3430 12. Verify that the values of the client extension outputs in  
3431 clientExtensionResults and the authenticator extension outputs in  
3432 the extensions in authData are as expected, considering the client  
3433 extension input values that were given as the extensions option in  
3434 the create() call. In particular, any extension identifier values  
3435 in the clientExtensionResults and the extensions in authData MUST  
3436 be also be present as extension identifier values in the extensions  
3437 member of options, i.e., no extensions are present that were not  
3438 requested. In the general case, the meaning of "are as expected" is  
3439 specific to the Relying Party and which extensions are in use.  
3440 Note: Since all extensions are OPTIONAL for both the client and the  
3441 authenticator, the Relying Party MUST be prepared to handle cases  
3442 where none or not all of the requested extensions were acted upon.  
3443 13. Determine the attestation statement format by performing a USASCI  
3444 case-sensitive match on fmt against the set of supported WebAuthn  
3445 Attestation Statement Format Identifier values. The up-to-date list  
3446 of registered WebAuthn Attestation Statement Format Identifier  
3447 values is maintained in the in the IANA registry of the same name  
3448 [WebAuthn-Registries].  
3449 14. Verify that attStmt is a correct attestation statement, conveying a  
3450 valid attestation signature, by using the attestation statement  
3451 format fmt's verification procedure given attStmt, authData and the  
3452 hash of the serialized client data computed in step 7.  
3453 Note: Each attestation statement format specifies its own  
3454 verification procedure. See 8 Defined Attestation Statement  
3455 Formats for the initially-defined formats, and  
3456 [WebAuthn-Registries] for the up-to-date list.  
3457 15. If validation is successful, obtain a list of acceptable trust  
3458 anchors (attestation root certificates or ECDAA-Issuer public keys)  
3459 for that attestation type and attestation statement format fmt,  
3460 from a trusted source or from policy. For example, the FIDO  
3461 Metadata Service [FIDOMetadataService] provides one way to obtain  
3462 such information, using the aaguid in the attestedCredentialData in  
3463 authData.  
3464 16. Assess the attestation trustworthiness using the outputs of the  
3465 verification procedure in step 14, as follows:  
3466 + If self attestation was used, check if self attestation is  
3467 acceptable under Relying Party policy.  
3468 + If ECDAA was used, verify that the identifier of the  
3469 ECDAA-Issuer public key used is included in the set of  
3470 acceptable trust anchors obtained in step 15.  
3471 + Otherwise, use the X.509 certificates returned by the  
3472 verification procedure to verify that the attestation public  
3473 key correctly chains up to an acceptable root certificate.  
3474 17. Check that the credentialId is not yet registered to any other  
3475 user. If registration is requested for a credential that is already  
3476 registered to a different user, the Relying Party SHOULD fail this  
3477 registration ceremony, or it MAY decide to accept the registration,  
3478 e.g. while deleting the older registration.  
3479 18. If the attestation statement attStmt verified successfully and is  
3480 found to be trustworthy, then register the new credential with the  
3481 account that was denoted in the options.user passed to create(), by  
3482 associating it with the credentialId and credentialPublicKey in the  
3483 attestedCredentialData in authData, as appropriate for the Relying  
3484 Party's system.  
3485 19. If the attestation statement attStmt successfully verified but is  
3486 not trustworthy per step 16 above, the Relying Party SHOULD fail the  
3487 registration ceremony.  
3488 NOTE: However, if permitted by policy, the Relying Party MAY  
3489 register the credential ID and credential public key but treat the  
3490 credential as one with self attestation (see 6.3.3 Attestation  
3491 Types). If doing so, the Relying Party is asserting there is no  
3492 cryptographic proof that the public key credential has been

3465 attestation statement format fmt, the authenticator data authData,  
3466 and the attestation statement attStmt.  
3467 9. Verify that the RP ID hash in authData is indeed the SHA-256 hash  
3468 of the RP ID expected by the RP.  
3469 10. Verify that the User Present bit of the flags in authData is set.  
3470 11. If user verification is required for this registration, verify that  
3471 the User Verified bit of the flags in authData is set.  
3472 12. Verify that the values of the client extension outputs in  
3473 clientExtensionResults and the authenticator extension outputs in  
3474 the extensions in authData are as expected, considering the client  
3475 extension input values that were given as the extensions option in  
3476 the create() call. In particular, any extension identifier values  
3477 in the clientExtensionResults and the extensions in authData MUST  
3478 be also be present as extension identifier values in the extensions  
3479 member of options, i.e., no extensions are present that were not  
3480 requested. In the general case, the meaning of "are as expected" is  
3481 specific to the Relying Party and which extensions are in use.  
3482 Note: Since all extensions are OPTIONAL for both the client and the  
3483 authenticator, the Relying Party MUST be prepared to handle cases  
3484 where none or not all of the requested extensions were acted upon.  
3485 13. Determine the attestation statement format by performing a USASCI  
3486 case-sensitive match on fmt against the set of supported WebAuthn  
3487 Attestation Statement Format Identifier values. The up-to-date list  
3488 of registered WebAuthn Attestation Statement Format Identifier  
3489 values is maintained in the in the IANA registry of the same name  
3490 [WebAuthn-Registries].  
3491 14. Verify that attStmt is a correct attestation statement, conveying a  
3492 valid attestation signature, by using the attestation statement  
3493 format fmt's verification procedure given attStmt, authData and the  
3494 hash of the serialized client data computed in step 7.  
3495 Note: Each attestation statement format specifies its own  
3496 verification procedure. See 8 Defined Attestation Statement  
3497 Formats for the initially-defined formats, and  
3498 [WebAuthn-Registries] for the up-to-date list.  
3499 15. If validation is successful, obtain a list of acceptable trust  
3500 anchors (attestation root certificates or ECDAA-Issuer public keys)  
3501 for that attestation type and attestation statement format fmt,  
3502 from a trusted source or from policy. For example, the FIDO  
3503 Metadata Service [FIDOMetadataService] provides one way to obtain  
3504 such information, using the aaguid in the attestedCredentialData in  
3505 authData.  
3506 16. Assess the attestation trustworthiness using the outputs of the  
3507 verification procedure in step 14, as follows:  
3508 + If self attestation was used, check if self attestation is  
3509 acceptable under Relying Party policy.  
3510 + If ECDAA was used, verify that the identifier of the  
3511 ECDAA-Issuer public key used is included in the set of  
3512 acceptable trust anchors obtained in step 15.  
3513 + Otherwise, use the X.509 certificates returned by the  
3514 verification procedure to verify that the attestation public  
3515 key correctly chains up to an acceptable root certificate.  
3516 17. Check that the credentialId is not yet registered to any other  
3517 user. If registration is requested for a credential that is already  
3518 registered to a different user, the Relying Party SHOULD fail this  
3519 registration ceremony, or it MAY decide to accept the registration,  
3520 e.g. while deleting the older registration.  
3521 18. If the attestation statement attStmt verified successfully and is  
3522 found to be trustworthy, then register the new credential with the  
3523 account that was denoted in the options.user passed to create(), by  
3524 associating it with the credentialId and credentialPublicKey in the  
3525 attestedCredentialData in authData, as appropriate for the Relying  
3526 Party's system.  
3527 19. If the attestation statement attStmt successfully verified but is  
3528 not trustworthy per step 16 above, the Relying Party SHOULD fail the  
3529 registration ceremony.  
3530 NOTE: However, if permitted by policy, the Relying Party MAY  
3531 register the credential ID and credential public key but treat the  
3532 credential as one with self attestation (see 6.3.3 Attestation  
3533 Types). If doing so, the Relying Party is asserting there is no  
3534 cryptographic proof that the public key credential has been

3493 generated by a particular authenticator model. See [FIDOSecRef] and  
3494 [UAFProtocol] for a more detailed discussion.  
3495

3496 Verification of attestation objects requires that the Relying Party has  
3497 a trusted method of determining acceptable trust anchors in step 15  
3498 above. Also, if certificates are being used, the Relying Party MUST  
3499 have access to certificate status information for the intermediate CA  
3500 certificates. The Relying Party MUST also be able to build the  
3501 attestation certificate chain if the client did not provide this chain  
3502 in the attestation information.  
3503

## 3504 7.2. Verifying an authentication assertion 3505

3506 When verifying a given PublicKeyCredential structure (credential) and  
3507 an AuthenticationExtensionsClientOutputs structure  
3508 clientExtensionResults, as part of an authentication ceremony, the  
3509 Relying Party MUST proceed as follows:

- 3510 1. If the allowCredentiaals option was given when this authentication  
3511 ceremony was initiated, verify that credential.id identifies one of  
3512 the public key credentials that were listed in allowCredentiaals.
- 3513 2. If credential.response.userHandle is present, verify that the user  
3514 identified by this value is the owner of the public key credential  
3515 identified by credential.id.
- 3516 3. Using credential's id attribute (or the corresponding rawId, if  
3517 base64url encoding is inappropriate for your use case), look up the  
3518 corresponding credential public key.
- 3519 4. Let cData, authData and sig denote the value of credential's  
3520 response's clientDataJSON, authenticatorData, and signature  
3521 respectively.
- 3522 5. Let JSONtext be the result of running UTF-8 decode on the value of  
3523 cData.  
3524 Note: Using any implementation of UTF-8 decode is acceptable as  
3525 long as it yields the same result as that yielded by the UTF-8  
3526 decode algorithm. In particular, any leading byte order mark (BOM)  
3527 MUST be stripped.
- 3528 6. Let C, the client data claimed as used for the signature, be the  
3529 result of running an implementation-specific JSON parser on  
3530 JSONtext.  
3531 Note: C may be any implementation-specific data structure  
3532 representation, as long as C's components are referenceable, as  
3533 required by this algorithm.
- 3534 7. Verify that the value of C.type is the string webauthn.get.
- 3535 8. Verify that the value of C.challenge matches the challenge that was  
3536 sent to the authenticator in the PublicKeyCredentialRequestOptions  
3537 passed to the get() call.
- 3538 9. Verify that the value of C.origin matches the Relying Party's  
3539 origin.
- 3540 10. Verify that the value of C.tokenBinding.status matches the state of  
3541 Token Binding for the TLS connection over which the attestation was  
3542 obtained. If Token Binding was used on that TLS connection, also  
3543 verify that C.tokenBinding.id matches the base64url encoding of the  
3544 Token Binding ID for the connection.
- 3545 11. Verify that the rpIdHash in authData is the SHA-256 hash of the RP  
3546 ID expected by the Relying Party.
- 3547 12. Verify that the User Present bit of the flags in authData is set.
- 3548 13. If user verification is required for this assertion, verify that  
3549 the User Verified bit of the flags in authData is set.
- 3550 14. Verify that the values of the client extension outputs in  
3551 clientExtensionResults and the authenticator extension outputs in  
3552 the extensions in authData are as expected, considering the client  
3553 extension input values that were given as the extensions option in  
3554 the get() call. In particular, any extension identifier values in  
3555 the clientExtensionResults and the extensions in authData MUST be  
3556 also be present as extension identifier values in the extensions  
3557 member of options, i.e., no extensions are present that were not  
3558 requested. In the general case, the meaning of "are as expected" is  
3559 specific to the Relying Party and which extensions are in use.  
3560 Note: Since all extensions are OPTIONAL for both the client and the  
3561 authenticator, the Relying Party MUST be prepared to handle cases  
3562 where none or not all of the requested extensions were acted upon.

3535 generated by a particular authenticator model. See [FIDOSecRef] and  
3536 [UAFProtocol] for a more detailed discussion.  
3537

3538 Verification of attestation objects requires that the Relying Party has  
3539 a trusted method of determining acceptable trust anchors in step 15  
3540 above. Also, if certificates are being used, the Relying Party MUST  
3541 have access to certificate status information for the intermediate CA  
3542 certificates. The Relying Party MUST also be able to build the  
3543 attestation certificate chain if the client did not provide this chain  
3544 in the attestation information.  
3545

## 3546 7.2. Verifying an authentication assertion 3547

3548 When verifying a given PublicKeyCredential structure (credential) and  
3549 an AuthenticationExtensionsClientOutputs structure  
3550 clientExtensionResults, as part of an authentication ceremony, the  
3551 Relying Party MUST proceed as follows:

- 3552 1. If the allowCredentiaals option was given when this authentication  
3553 ceremony was initiated, verify that credential.id identifies one of  
3554 the public key credentials that were listed in allowCredentiaals.
- 3555 2. If credential.response.userHandle is present, verify that the user  
3556 identified by this value is the owner of the public key credential  
3557 identified by credential.id.
- 3558 3. Using credential's id attribute (or the corresponding rawId, if  
3559 base64url encoding is inappropriate for your use case), look up the  
3560 corresponding credential public key.
- 3561 4. Let cData, authData and sig denote the value of credential's  
3562 response's clientDataJSON, authenticatorData, and signature  
3563 respectively.
- 3564 5. Let JSONtext be the result of running UTF-8 decode on the value of  
3565 cData.  
3566 Note: Using any implementation of UTF-8 decode is acceptable as  
3567 long as it yields the same result as that yielded by the UTF-8  
3568 decode algorithm. In particular, any leading byte order mark (BOM)  
3569 MUST be stripped.
- 3570 6. Let C, the client data claimed as used for the signature, be the  
3571 result of running an implementation-specific JSON parser on  
3572 JSONtext.  
3573 Note: C may be any implementation-specific data structure  
3574 representation, as long as C's components are referenceable, as  
3575 required by this algorithm.
- 3576 7. Verify that the value of C.type is the string webauthn.get.
- 3577 8. Verify that the value of C.challenge matches the challenge that was  
3578 sent to the authenticator in the PublicKeyCredentialRequestOptions  
3579 passed to the get() call.
- 3580 9. Verify that the value of C.origin matches the Relying Party's  
3581 origin.
- 3582 10. Verify that the value of C.tokenBinding.status matches the state of  
3583 Token Binding for the TLS connection over which the attestation was  
3584 obtained. If Token Binding was used on that TLS connection, also  
3585 verify that C.tokenBinding.id matches the base64url encoding of the  
3586 Token Binding ID for the connection.
- 3587 11. Verify that the rpIdHash in authData is the SHA-256 hash of the RP  
3588 ID expected by the Relying Party.
- 3589 12. Verify that the User Present bit of the flags in authData is set.
- 3590 13. If user verification is required for this assertion, verify that  
3591 the User Verified bit of the flags in authData is set.
- 3592 14. Verify that the values of the client extension outputs in  
3593 clientExtensionResults and the authenticator extension outputs in  
3594 the extensions in authData are as expected, considering the client  
3595 extension input values that were given as the extensions option in  
3596 the get() call. In particular, any extension identifier values in  
3597 the clientExtensionResults and the extensions in authData MUST be  
3598 also be present as extension identifier values in the extensions  
3599 member of options, i.e., no extensions are present that were not  
3600 requested. In the general case, the meaning of "are as expected" is  
3601 specific to the Relying Party and which extensions are in use.  
3602 Note: Since all extensions are OPTIONAL for both the client and the  
3603 authenticator, the Relying Party MUST be prepared to handle cases  
3604 where none or not all of the requested extensions were acted upon.

3563 15. Let hash be the result of computing a hash over the cData using  
3564 SHA-256.  
3565 16. Using the credential public key looked up in step 3, verify that  
3566 sig is a valid signature over the binary concatenation of authData  
3567 and hash.  
3568 Note: This verification step is compatible with signatures  
3569 generated by FIDO U2F authenticators. See 6.1.2 FIDO U2F signature  
3570 format compatibility.  
3571 17. If the signature counter value authData.signCount is nonzero or the  
3572 value stored in conjunction with credential's id attribute is  
3573 nonzero, then run the following sub-step:  
3574 + If the signature counter value authData.signCount is  
3575  
3576 greater than the signature counter value stored in  
3577 conjunction with credential's id attribute.  
3578 Update the stored signature counter value,  
3579 associated with credential's id attribute, to be the  
3580 value of authData.signCount.  
3581  
3582 less than or equal to the signature counter value stored in  
3583 conjunction with credential's id attribute.  
3584 This is a signal that the authenticator may be  
3585 cloned, i.e. at least two copies of the credential  
3586 private key may exist and are being used in  
3587 parallel. Relying Parties should incorporate this  
3588 information into their risk scoring. Whether the  
3589 Relying Party updates the stored signature counter  
3590 value in this case, or not, or fails the  
3591 authentication ceremony or not, is Relying  
3592 Party-specific.

3593  
3594 18. If all the above steps are successful, continue with the  
3595 authentication ceremony as appropriate. Otherwise, fail the  
3596 authentication ceremony.  
3597

3598 8. Defined Attestation Statement Formats

3599 WebAuthn supports pluggable attestation statement formats. This section  
3600 defines an initial set of such formats.

3601 8.1. Attestation Statement Format Identifiers

3602 Attestation statement formats are identified by a string, called an  
3603 attestation statement format identifier, chosen by the author of the  
3604 attestation statement format.

3605 Attestation statement format identifiers SHOULD be registered per  
3606 [WebAuthn-Registries] "Registries for Web Authentication (WebAuthn)".  
3607 All registered attestation statement format identifiers are unique  
3608 amongst themselves as a matter of course.

3609 Unregistered attestation statement format identifiers SHOULD use  
3610 lowercase reverse domain-name naming, using a domain name registered by  
3611 the developer, in order to assure uniqueness of the identifier. All  
3612 attestation statement format identifiers MUST be a maximum of 32 octets  
3613 in length and MUST consist only of printable USASCII characters,  
3614 excluding backslash and doublequote, i.e., VCHAR as defined in  
3615 [RFC5234] but without %x22 and %x5c.

3616 Note: This means attestation statement format identifiers based on  
3617 domain names MUST incorporate only LDH Labels [RFC5890].

3618 Implementations MUST match WebAuthn attestation statement format  
3619 identifiers in a case-sensitive fashion.

3620 Attestation statement formats that may exist in multiple versions  
3621 SHOULD include a version in their identifier. In effect, different  
3622 versions are thus treated as different formats, e.g., packed2 as a new  
3623 version of the packed attestation statement format.  
3624

3605 15. Let hash be the result of computing a hash over the cData using  
3606 SHA-256.  
3607 16. Using the credential public key looked up in step 3, verify that  
3608 sig is a valid signature over the binary concatenation of authData  
3609 and hash.  
3610 Note: This verification step is compatible with signatures  
3611 generated by FIDO U2F authenticators. See 6.1.2 FIDO U2F signature  
3612 format compatibility.  
3613 17. If the signature counter value authData.signCount is nonzero or the  
3614 value stored in conjunction with credential's id attribute is  
3615 nonzero, then run the following sub-step:  
3616 + If the signature counter value authData.signCount is  
3617  
3618 greater than the signature counter value stored in  
3619 conjunction with credential's id attribute.  
3620 Update the stored signature counter value,  
3621 associated with credential's id attribute, to be the  
3622 value of authData.signCount.  
3623  
3624 less than or equal to the signature counter value stored in  
3625 conjunction with credential's id attribute.  
3626 This is a signal that the authenticator may be  
3627 cloned, i.e. at least two copies of the credential  
3628 private key may exist and are being used in  
3629 parallel. Relying Parties should incorporate this  
3630 information into their risk scoring. Whether the  
3631 Relying Party updates the stored signature counter  
3632 value in this case, or not, or fails the  
3633 authentication ceremony or not, is Relying  
3634 Party-specific.

3635  
3636 18. If all the above steps are successful, continue with the  
3637 authentication ceremony as appropriate. Otherwise, fail the  
3638 authentication ceremony.  
3639

3640 8. Defined Attestation Statement Formats

3641 WebAuthn supports pluggable attestation statement formats. This section  
3642 defines an initial set of such formats.

3643 8.1. Attestation Statement Format Identifiers

3644 Attestation statement formats are identified by a string, called an  
3645 attestation statement format identifier, chosen by the author of the  
3646 attestation statement format.

3647 Attestation statement format identifiers SHOULD be registered per  
3648 [WebAuthn-Registries] "Registries for Web Authentication (WebAuthn)".  
3649 All registered attestation statement format identifiers are unique  
3650 amongst themselves as a matter of course.

3651 Unregistered attestation statement format identifiers SHOULD use  
3652 lowercase reverse domain-name naming, using a domain name registered by  
3653 the developer, in order to assure uniqueness of the identifier. All  
3654 attestation statement format identifiers MUST be a maximum of 32 octets  
3655 in length and MUST consist only of printable USASCII characters,  
3656 excluding backslash and doublequote, i.e., VCHAR as defined in  
3657 [RFC5234] but without %x22 and %x5c.

3658 Note: This means attestation statement format identifiers based on  
3659 domain names MUST incorporate only LDH Labels [RFC5890].

3660 Implementations MUST match WebAuthn attestation statement format  
3661 identifiers in a case-sensitive fashion.

3662 Attestation statement formats that may exist in multiple versions  
3663 SHOULD include a version in their identifier. In effect, different  
3664 versions are thus treated as different formats, e.g., packed2 as a new  
3665 version of the packed attestation statement format.  
3666

3633 The following sections present a set of currently-defined and  
 3634 registered attestation statement formats and their identifiers. The  
 3635 up-to-date list of registered WebAuthn Extensions is maintained in the  
 3636 IANA "WebAuthn Attestation Statement Format Identifier" registry  
 3637 established by [WebAuthn-Registries].  
 3638

## 3639 8.2. Packed Attestation Statement Format

3640 This is a WebAuthn optimized attestation statement format. It uses a  
 3641 very compact but still extensible encoding method. It is implementable  
 3642 by authenticators with limited resources (e.g., secure elements).  
 3643

3644 Attestation statement format identifier  
 3645 packed

3646 Attestation types supported  
 3647 All

3648 Syntax  
 3649 The syntax of a Packed Attestation statement is defined by the  
 3650 following CDDL:

```
3651 $$attStmtType ::= (
3652     fmt: "packed",
3653     attStmt: packedStmtFormat
3654 )
3655
3656 packedStmtFormat = {
3657     alg: COSEAlgorithmIdentifier,
3658     sig: bytes,
3659     x5c: [ attestnCert: bytes, * (caCert: bytes) ]
3660 } //
3661 {
3662     alg: COSEAlgorithmIdentifier, (-260 for ED256 / -261
3663 for ED512)
3664     sig: bytes,
3665     ecdaaKeyld: bytes
3666 } //
3667 {
3668     alg: COSEAlgorithmIdentifier
3669     sig: bytes,
3670 }
```

3671 The semantics of the fields are as follows:

3672 alg A COSEAlgorithmIdentifier containing the identifier of the  
 3673 algorithm used to generate the attestation signature.

3674 sig A byte string containing the attestation signature.

3675 x5c The elements of this array contain the attestation  
 3676 certificate and its certificate chain, each encoded in  
 3677 X.509 format. The attestation certificate MUST be the  
 3678 first element in the array.

3679 ecdaaKeyld  
 3680 The identifier of the ECDSA-Issuer public key. This is the  
 3681 BigIntegerToB encoding of the component "c" of the  
 3682 ECDSA-Issuer public key as defined section 3.3, step 3.5  
 3683 in [FIDOEcdaaAlgorithm].

3684 Signing procedure  
 3685 The signing procedure for this attestation statement format is  
 3686 similar to the procedure for generating assertion signatures.

- 3687 1. Let authenticatorData denote the authenticator data for the  
 3688 attestation, and let clientDataHash denote the hash of the

3702

3676 The following sections present a set of currently-defined and  
 3677 registered attestation statement formats and their identifiers. The  
 3678 up-to-date list of registered WebAuthn Extensions is maintained in the  
 3679 IANA "WebAuthn Attestation Statement Format Identifier" registry  
 3680 established by [WebAuthn-Registries].

## 3681 8.2. Packed Attestation Statement Format

3682 This is a WebAuthn optimized attestation statement format. It uses a  
 3683 very compact but still extensible encoding method. It is implementable  
 3684 by authenticators with limited resources (e.g., secure elements).  
 3685

3686 Attestation statement format identifier  
 3687 packed

3688 Attestation types supported  
 3689 All

3690 Syntax  
 3691 The syntax of a Packed Attestation statement is defined by the  
 3692 following CDDL:

```
3693 $$attStmtType ::= (
3694     fmt: "packed",
3695     attStmt: packedStmtFormat
3696 )
3697
3698 packedStmtFormat = {
3699     alg: COSEAlgorithmIdentifier,
3700     sig: bytes,
3701     x5c: [ attestnCert: bytes, * (caCert: bytes) ]
3702 } //
3703 {
3704     alg: COSEAlgorithmIdentifier, (-260 for ED256 / -261
3705 for ED512)
3706     sig: bytes,
3707     ecdaaKeyld: bytes
3708 } //
3709 {
3710     alg: COSEAlgorithmIdentifier
3711     sig: bytes,
3712 }
```

3713 The semantics of the fields are as follows:

3714 alg A COSEAlgorithmIdentifier containing the identifier of the  
 3715 algorithm used to generate the attestation signature.

3716 sig A byte string containing the attestation signature.

3717 x5c The elements of this array contain the attestation  
 3718 certificate and its certificate chain, each encoded in  
 3719 X.509 format. The attestation certificate MUST be the  
 3720 first element in the array.

3721 ecdaaKeyld  
 3722 The identifier of the ECDSA-Issuer public key. This is the  
 3723 BigIntegerToB encoding of the component "c" of the  
 3724 ECDSA-Issuer public key as defined section 3.3, step 3.5  
 3725 in [FIDOEcdaaAlgorithm].

3726 Signing procedure  
 3727 The signing procedure for this attestation statement format is  
 3728 similar to the procedure for generating assertion signatures.

- 3729 1. Let authenticatorData denote the authenticator data for the  
 3730 attestation, and let clientDataHash denote the hash of the

3744

3703 serialized client data.  
 3704 2. If Basic or AttCA attestation is in use, the authenticator  
 3705 produces the sig by concatenating authenticatorData and  
 3706 clientDataHash, and signing the result using an attestation  
 3707 private key selected through an authenticator-specific  
 3708 mechanism. It sets x5c to the certificate chain of the  
 3709 attestation public key and alg to the algorithm of the  
 3710 attestation private key.  
 3711 3. If ECDAA is in use, the authenticator produces sig by  
 3712 concatenating authenticatorData and clientDataHash, and  
 3713 signing the result using ECDAA-Sign (see section 3.5 of  
 3714 [FIDOEcdaaAlgorithm]) after selecting an ECDAA-Issuer public  
 3715 key related to the ECDAA signature private key through an  
 3716 authenticator-specific mechanism (see [FIDOEcdaaAlgorithm]).  
 3717 It sets alg to the algorithm of the selected ECDAA-Issuer  
 3718 public key and ecdaaKeyld to the identifier of the  
 3719 ECDAA-Issuer public key (see above).  
 3720 4. If self attestation is in use, the authenticator produces sig  
 3721 by concatenating authenticatorData and clientDataHash, and  
 3722 signing the result using the credential private key. It sets  
 3723 alg to the algorithm of the credential private key and omits  
 3724 the other fields.  
 3725

3726 Verification procedure

3727 Given the verification procedure inputs attStmt,  
 3728 authenticatorData and clientDataHash, the verification procedure  
 3729 is as follows:

- 3730 1. Verify that attStmt is valid CBOR conforming to the syntax  
 3731 defined above and perform CBOR decoding on it to extract the  
 3732 contained fields.
- 3733 2. If x5c is present, this indicates that the attestation type is  
 3734 not ECDAA. In this case:  
 3735 o Verify that sig is a valid signature over the  
 3736 concatenation of authenticatorData and clientDataHash  
 3737 using the attestation public key in x5c with the  
 3738 algorithm specified in alg.  
 3739 o Verify that x5c meets the requirements in 8.2.1 Packed  
 3740 attestation statement certificate requirements.  
 3741 o If x5c contains an extension with OID  
 3742 1.3.6.1.4.1.45724.1.1.4 (id-fido-gen-ce-aaguid) verify  
 3743 that the value of this extension matches the aaguid in  
 3744 authenticatorData.  
 3745 o If successful, return attestation type Basic and  
 3746 attestation trust path x5c.
- 3747 3. If ecdaaKeyld is present, then the attestation type is ECDAA.  
 3748 In this case:  
 3749 o Verify that sig is a valid signature over the  
 3750 concatenation of authenticatorData and clientDataHash  
 3751 using ECDAA-Verify with ECDAA-Issuer public key  
 3752 identified by ecdaaKeyld (see [FIDOEcdaaAlgorithm]).  
 3753 o If successful, return attestation type ECDAA and  
 3754 attestation trust path ecdaaKeyld.
- 3755 4. If neither x5c nor ecdaaKeyld is present, self attestation is  
 3756 in use.  
 3757 o Validate that alg matches the algorithm of the  
 3758 credentialPublicKey in authenticatorData.  
 3759 o Verify that sig is a valid signature over the  
 3760 concatenation of authenticatorData and clientDataHash  
 3761 using the credential public key with alg.  
 3762 o If successful, return attestation type Self and empty  
 3763 attestation trust path.

3764 8.2.1. Packed attestation statement certificate requirements

3765 The attestation certificate MUST have the following fields/extensions:  
 3766 \* Version MUST be set to 3 (which is indicated by an ASN.1 INTEGER  
 3767 with value 2).  
 3770 \* Subject field MUST be set to:

3771  
 3772

3745 serialized client data.  
 3746 2. If Basic or AttCA attestation is in use, the authenticator  
 3747 produces the sig by concatenating authenticatorData and  
 3748 clientDataHash, and signing the result using an attestation  
 3749 private key selected through an authenticator-specific  
 3750 mechanism. It sets x5c to the certificate chain of the  
 3751 attestation public key and alg to the algorithm of the  
 3752 attestation private key.  
 3753 3. If ECDAA is in use, the authenticator produces sig by  
 3754 concatenating authenticatorData and clientDataHash, and  
 3755 signing the result using ECDAA-Sign (see section 3.5 of  
 3756 [FIDOEcdaaAlgorithm]) after selecting an ECDAA-Issuer public  
 3757 key related to the ECDAA signature private key through an  
 3758 authenticator-specific mechanism (see [FIDOEcdaaAlgorithm]).  
 3759 It sets alg to the algorithm of the selected ECDAA-Issuer  
 3760 public key and ecdaaKeyld to the identifier of the  
 3761 ECDAA-Issuer public key (see above).  
 3762 4. If self attestation is in use, the authenticator produces sig  
 3763 by concatenating authenticatorData and clientDataHash, and  
 3764 signing the result using the credential private key. It sets  
 3765 alg to the algorithm of the credential private key and omits  
 3766 the other fields.  
 3767

3768 Verification procedure

3769 Given the verification procedure inputs attStmt,  
 3770 authenticatorData and clientDataHash, the verification procedure  
 3771 is as follows:

- 3772 1. Verify that attStmt is valid CBOR conforming to the syntax  
 3773 defined above and perform CBOR decoding on it to extract the  
 3774 contained fields.
- 3775 2. If x5c is present, this indicates that the attestation type is  
 3776 not ECDAA. In this case:  
 3777 o Verify that sig is a valid signature over the  
 3778 concatenation of authenticatorData and clientDataHash  
 3779 using the attestation public key in x5c with the  
 3780 algorithm specified in alg.  
 3781 o Verify that x5c meets the requirements in 8.2.1 Packed  
 3782 attestation statement certificate requirements.  
 3783 o If x5c contains an extension with OID  
 3784 1.3.6.1.4.1.45724.1.1.4 (id-fido-gen-ce-aaguid) verify  
 3785 that the value of this extension matches the aaguid in  
 3786 authenticatorData.  
 3787 o If successful, return attestation type Basic and  
 3788 attestation trust path x5c.
- 3789 3. If ecdaaKeyld is present, then the attestation type is ECDAA.  
 3790 In this case:  
 3791 o Verify that sig is a valid signature over the  
 3792 concatenation of authenticatorData and clientDataHash  
 3793 using ECDAA-Verify with ECDAA-Issuer public key  
 3794 identified by ecdaaKeyld (see [FIDOEcdaaAlgorithm]).  
 3795 o If successful, return attestation type ECDAA and  
 3796 attestation trust path ecdaaKeyld.
- 3797 4. If neither x5c nor ecdaaKeyld is present, self attestation is  
 3798 in use.  
 3799 o Validate that alg matches the algorithm of the  
 3800 credentialPublicKey in authenticatorData.  
 3801 o Verify that sig is a valid signature over the  
 3802 concatenation of authenticatorData and clientDataHash  
 3803 using the credential public key with alg.  
 3804 o If successful, return attestation type Self and empty  
 3805 attestation trust path.

3806 8.2.1. Packed attestation statement certificate requirements

3807 The attestation certificate MUST have the following fields/extensions:  
 3808 \* Version MUST be set to 3 (which is indicated by an ASN.1 INTEGER  
 3809 with value 2).  
 3810 \* Subject field MUST be set to:

3811  
 3812  
 3813  
 3814

```

3773 Subject-C
3774 ISO 3166 code specifying the country where the
3775 Authenticator vendor is incorporated (PrintableString)
3776
3777 Subject-O
3778 Legal name of the Authenticator vendor (UTF8String)
3779
3780 Subject-OU
3781 Literal string "Authenticator Attestation" (UTF8String)
3782
3783 Subject-CN
3784 A UTF8String of the vendor's choosing
3785
3786 * If the related attestation root certificate is used for multiple
3787 authenticator models, the Extension OID 1.3.6.1.4.1.45724.1.1.4
3788 (id-fido-gen-ce-aaguid) MUST be present, containing the AAGUID as a
3789 16-byte OCTET STRING. The extension MUST NOT be marked as critical.
3790 Note that an X.509 Extension encodes the DER-encoding of the value
3791 in an OCTET STRING. Thus, the AAGUID must be wrapped in two OCTET
3792 STRINGS to be valid. Here is a sample, encoded Extension structure:
3793
3794 30 21 -- SEQUENCE
3795 06 0b 2b 06 01 04 01 82 e5 1c 01 01 04 -- 1.3.6.1.4.1.45724.1.1.4
3796 04 12 -- OCTET STRING
3797 04 10 -- OCTET STRING
3798 cd 8c 39 5c 26 ed ee de -- AAGUID
3799 65 3b 00 79 7d 03 ca 3c
3800
3801 * The Basic Constraints extension MUST have the CA component set to
3802 false.
3803 * An Authority Information Access (AIA) extension with entry
3804 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are
3805 both OPTIONAL as the status of many attestation certificates is
3806 available through authenticator metadata services. See, for
3807 example, the FIDO Metadata Service [FIDOMetadataService].
3808
3809 8.3. TPM Attestation Statement Format
3810
3811 This attestation statement format is generally used by authenticators
3812 that use a Trusted Platform Module as their cryptographic engine.
3813
3814 Attestation statement format identifier
3815 tpm
3816
3817 Attestation types supported
3818 AttCA, ECDA
3819
3820 Syntax
3821 The syntax of a TPM Attestation statement is as follows:
3822
3823 $$attStmtType // = (
3824     fmt: "tpm",
3825     attStmt: tpmStmtFormat
3826 )
3827
3828 tpmStmtFormat = {
3829     ver: "2.0",
3830     (
3831         alg: COSEAlgorithmIdentifier,
3832         x5c: [ aikCert: bytes, * (caCert: bytes) ]
3833     ) //
3834     alg: COSEAlgorithmIdentifier, (-260 for ED256 / -26
3835 1 for ED512)
3836     ecdaaKeyId: bytes
3837 ),
3838     sig: bytes,
3839     certInfo: bytes,
3840     pubArea: bytes
3841 }
3842

```

```

3815 Subject-C
3816 ISO 3166 code specifying the country where the
3817 Authenticator vendor is incorporated (PrintableString)
3818
3819 Subject-O
3820 Legal name of the Authenticator vendor (UTF8String)
3821
3822 Subject-OU
3823 Literal string "Authenticator Attestation" (UTF8String)
3824
3825 Subject-CN
3826 A UTF8String of the vendor's choosing
3827
3828 * If the related attestation root certificate is used for multiple
3829 authenticator models, the Extension OID 1.3.6.1.4.1.45724.1.1.4
3830 (id-fido-gen-ce-aaguid) MUST be present, containing the AAGUID as a
3831 16-byte OCTET STRING. The extension MUST NOT be marked as critical.
3832 Note that an X.509 Extension encodes the DER-encoding of the value
3833 in an OCTET STRING. Thus, the AAGUID must be wrapped in two OCTET
3834 STRINGS to be valid. Here is a sample, encoded Extension structure:
3835
3836 30 21 -- SEQUENCE
3837 06 0b 2b 06 01 04 01 82 e5 1c 01 01 04 -- 1.3.6.1.4.1.45724.1.1.4
3838 04 12 -- OCTET STRING
3839 04 10 -- OCTET STRING
3840 cd 8c 39 5c 26 ed ee de -- AAGUID
3841 65 3b 00 79 7d 03 ca 3c
3842
3843 * The Basic Constraints extension MUST have the CA component set to
3844 false.
3845 * An Authority Information Access (AIA) extension with entry
3846 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are
3847 both OPTIONAL as the status of many attestation certificates is
3848 available through authenticator metadata services. See, for
3849 example, the FIDO Metadata Service [FIDOMetadataService].
3850
3851 8.3. TPM Attestation Statement Format
3852
3853 This attestation statement format is generally used by authenticators
3854 that use a Trusted Platform Module as their cryptographic engine.
3855
3856 Attestation statement format identifier
3857 tpm
3858
3859 Attestation types supported
3860 AttCA, ECDA
3861
3862 Syntax
3863 The syntax of a TPM Attestation statement is as follows:
3864
3865 $$attStmtType // = (
3866     fmt: "tpm",
3867     attStmt: tpmStmtFormat
3868 )
3869
3870 tpmStmtFormat = {
3871     ver: "2.0",
3872     (
3873         alg: COSEAlgorithmIdentifier,
3874         x5c: [ aikCert: bytes, * (caCert: bytes) ]
3875     ) //
3876     alg: COSEAlgorithmIdentifier, (-260 for ED256 / -26
3877 1 for ED512)
3878     ecdaaKeyId: bytes
3879 ),
3880     sig: bytes,
3881     certInfo: bytes,
3882     pubArea: bytes
3883 }
3884

```



3843 The semantics of the above fields are as follows:  
3844  
3845 ver  
3846 The version of the TPM specification to which the  
3847 signature conforms.  
3848  
3849 alg  
3850 A COSEAlgorithmIdentifier containing the identifier of the  
3851 algorithm used to generate the attestation signature.  
3852  
3853 x5c  
3854 The AIK certificate used for the attestation and its  
3855 certificate chain, in X.509 encoding.  
3856  
3857 ecdaaKeyId  
3858 The identifier of the ECDAA-Issuer public key. This is the  
3859 BigNumberToB encoding of the component "c" as defined  
3860 section 3.3, step 3.5 in [FIDOecdaaAlgorithm].  
3861  
3862 sig  
3863 The attestation signature, in the form of a TPMT\_SIGNATURE  
3864 structure as specified in [TPMv2-Part2] section 11.3.4.  
3865  
3866 certInfo  
3867 The TPMS\_ATTEST structure over which the above signature  
3868 was computed, as specified in [TPMv2-Part2] section  
3869 10.12.8.  
3870  
3871 pubArea  
3872 The TPMT\_PUBLIC structure (see [TPMv2-Part2] section  
3873 12.2.4) used by the TPM to represent the credential public  
3874 key.  
3875  
3876 Signing procedure  
3877 Let authenticatorData denote the authenticator data for the  
3878 attestation, and let clientDataHash denote the hash of the  
3879 serialized client data.  
3880  
3881 Concatenate authenticatorData and clientDataHash to form  
3882 attToBeSigned.  
3883  
3884 Generate a signature using the procedure specified in  
3885 [TPMv2-Part3] Section 18.2, using the attestation private key  
3886 and setting the extraData parameter to the digest of  
3887 attToBeSigned using the hash algorithm corresponding to the  
3888 "alg" signature algorithm. (For the "RS256" algorithm, this  
3889 would be a SHA-256 digest.)  
3890  
3891 Set the pubArea field to the public area of the credential  
3892 public key, the certInfo field to the output parameter of the  
3893 same name, and the sig field to the signature obtained from the  
3894 above procedure.  
3895  
3896 Verification procedure  
3897 Given the verification procedure inputs attStmt,  
3898 authenticatorData and clientDataHash, the verification procedure  
3899 is as follows:  
3900  
3901 Verify that attStmt is valid CBOR conforming to the syntax  
3902 defined above and perform CBOR decoding on it to extract the  
3903 contained fields.  
3904  
3905 Verify that the public key specified by the parameters and  
3906 unique fields of pubArea is identical to the credentialPublicKey  
3907 in the attestedCredentialData in authenticatorData.  
3908  
3909 Concatenate authenticatorData and clientDataHash to form  
3910 attToBeSigned.  
3911  
3912 Validate that certInfo is valid:

3885 The semantics of the above fields are as follows:  
3886  
3887 ver  
3888 The version of the TPM specification to which the  
3889 signature conforms.  
3890  
3891 alg  
3892 A COSEAlgorithmIdentifier containing the identifier of the  
3893 algorithm used to generate the attestation signature.  
3894  
3895 x5c  
3896 The AIK certificate used for the attestation and its  
3897 certificate chain, in X.509 encoding.  
3898  
3899 ecdaaKeyId  
3900 The identifier of the ECDAA-Issuer public key. This is the  
3901 BigNumberToB encoding of the component "c" as defined  
3902 section 3.3, step 3.5 in [FIDOecdaaAlgorithm].  
3903  
3904 sig  
3905 The attestation signature, in the form of a TPMT\_SIGNATURE  
3906 structure as specified in [TPMv2-Part2] section 11.3.4.  
3907  
3908 certInfo  
3909 The TPMS\_ATTEST structure over which the above signature  
3910 was computed, as specified in [TPMv2-Part2] section  
3911 10.12.8.  
3912  
3913 pubArea  
3914 The TPMT\_PUBLIC structure (see [TPMv2-Part2] section  
3915 12.2.4) used by the TPM to represent the credential public  
3916 key.  
3917  
3918 Signing procedure  
3919 Let authenticatorData denote the authenticator data for the  
3920 attestation, and let clientDataHash denote the hash of the  
3921 serialized client data.  
3922  
3923 Concatenate authenticatorData and clientDataHash to form  
3924 attToBeSigned.  
3925  
3926 Generate a signature using the procedure specified in  
3927 [TPMv2-Part3] Section 18.2, using the attestation private key  
3928 and setting the extraData parameter to the digest of  
3929 attToBeSigned using the hash algorithm corresponding to the  
3930 "alg" signature algorithm. (For the "RS256" algorithm, this  
3931 would be a SHA-256 digest.)  
3932  
3933 Set the pubArea field to the public area of the credential  
3934 public key, the certInfo field to the output parameter of the  
3935 same name, and the sig field to the signature obtained from the  
3936 above procedure.  
3937  
3938 Verification procedure  
3939 Given the verification procedure inputs attStmt,  
3940 authenticatorData and clientDataHash, the verification procedure  
3941 is as follows:  
3942  
3943 Verify that attStmt is valid CBOR conforming to the syntax  
3944 defined above and perform CBOR decoding on it to extract the  
3945 contained fields.  
3946  
3947 Verify that the public key specified by the parameters and  
3948 unique fields of pubArea is identical to the credentialPublicKey  
3949 in the attestedCredentialData in authenticatorData.  
3950  
3951 Concatenate authenticatorData and clientDataHash to form  
3952 attToBeSigned.  
3953  
3954 Validate that certInfo is valid:

3913 + Verify that magic is set to TPM\_GENERATED\_VALUE.  
3914 + Verify that type is set to TPM\_ST\_ATTEST\_CERTIFY.  
3915 + Verify that extraData is set to the hash of attToBeSigned  
3916 using the hash algorithm employed in "alg".  
3917 + Verify that attested contains a TPMS\_CERTIFY\_INFO structure as  
3918 specified in [TPMv2-Part2] section 10.12.3, whose name field  
3919 contains a valid Name for pubArea, as computed using the  
3920 algorithm in the nameAlg field of pubArea using the procedure  
3921 specified in [TPMv2-Part1] section 16.  
3922 + Note that the remaining fields in the "Standard Attestation  
3923 Structure" [TPMv2-Part1] section 31.2, i.e., qualifiedSigner,  
3924 clockInfo and firmwareVersion are ignored. These fields MAY be  
3925 used as an input to risk engines.  
3926  
3927 If x5c is present, this indicates that the attestation type is  
3928 not ECDA. In this case:  
3929  
3930 + Verify the sig is a valid signature over certInfo using the  
3931 attestation public key in x5c with the algorithm specified in  
3932 alg.  
3933 + Verify that x5c meets the requirements in 8.3.1 TPM  
3934 attestation statement certificate requirements.  
3935 + If x5c contains an extension with OID 1.3.6.1.4.1.45724.1.1.4  
3936 (id-fido-gen-ce-aaguid) verify that the value of this  
3937 extension matches the aaguid in authenticatorData.  
3938 + If successful, return attestation type AttCA and attestation  
3939 trust path x5c.  
3940  
3941 If ecdaaKeyId is present, then the attestation type is ECDA.  
3942  
3943 + Perform ECDA-Verify on sig to verify that it is a valid  
3944 signature over certInfo (see [FIDOEcdaaAlgorithm]).  
3945 + If successful, return attestation type ECDA and the  
3946 identifier of the ECDA-Issuer public key ecdaaKeyId.  
3947  
3948 **8.3.1. TPM attestation statement certificate requirements**  
3949 TPM attestation certificate MUST have the following fields/extensions:  
3950 \* Version MUST be set to 3.  
3951 \* Subject field MUST be set to empty.  
3952 \* The Subject Alternative Name extension MUST be set as defined in  
3953 [TPMv2-EK-Profile] section 3.2.9.  
3954 \* The Extended Key Usage extension MUST contain the  
3955 "joint-iso-itu-t(2) internationalorganizations(23) 133 tcg-kp(8)  
3956 tcg-kp-AIKCertificate(3)" OID.  
3957 \* The Basic Constraints extension MUST have the CA component set to  
3958 false.  
3959 \* An Authority Information Access (AIA) extension with entry  
3960 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are  
3961 both OPTIONAL as the status of many attestation certificates is  
3962 available through metadata services. See, for example, the FIDO  
3963 Metadata Service [FIDOMetadataService].  
3964  
3965 **8.4. Android Key Attestation Statement Format**  
3966 When the authenticator in question is a platform-provided Authenticator  
3967 on the Android "N" or later platform, the attestation statement is  
3968 based on the Android key attestation. In these cases, the attestation  
3969 statement is produced by a component running in a secure operating  
3970 environment, but the authenticator data for the attestation is produced  
3971 outside this environment. The Relying Party is expected to check that  
3972 the authenticator data claimed to have been used for the attestation is  
3973 consistent with the fields of the attestation certificate's extension  
3974 data.  
3975  
3976 Attestation statement format identifier  
3977 android-key  
3978  
3979 Attestation types supported

3955 + Verify that magic is set to TPM\_GENERATED\_VALUE.  
3956 + Verify that type is set to TPM\_ST\_ATTEST\_CERTIFY.  
3957 + Verify that extraData is set to the hash of attToBeSigned  
3958 using the hash algorithm employed in "alg".  
3959 + Verify that attested contains a TPMS\_CERTIFY\_INFO structure as  
3960 specified in [TPMv2-Part2] section 10.12.3, whose name field  
3961 contains a valid Name for pubArea, as computed using the  
3962 algorithm in the nameAlg field of pubArea using the procedure  
3963 specified in [TPMv2-Part1] section 16.  
3964 + Note that the remaining fields in the "Standard Attestation  
3965 Structure" [TPMv2-Part1] section 31.2, i.e., qualifiedSigner,  
3966 clockInfo and firmwareVersion are ignored. These fields MAY be  
3967 used as an input to risk engines.  
3968  
3969 If x5c is present, this indicates that the attestation type is  
3970 not ECDA. In this case:  
3971  
3972 + Verify the sig is a valid signature over certInfo using the  
3973 attestation public key in x5c with the algorithm specified in  
3974 alg.  
3975 + Verify that x5c meets the requirements in 8.3.1 TPM  
3976 attestation statement certificate requirements.  
3977 + If x5c contains an extension with OID 1.3.6.1.4.1.45724.1.1.4  
3978 (id-fido-gen-ce-aaguid) verify that the value of this  
3979 extension matches the aaguid in authenticatorData.  
3980 + If successful, return attestation type AttCA and attestation  
3981 trust path x5c.  
3982  
3983 If ecdaaKeyId is present, then the attestation type is ECDA.  
3984  
3985 + Perform ECDA-Verify on sig to verify that it is a valid  
3986 signature over certInfo (see [FIDOEcdaaAlgorithm]).  
3987 + If successful, return attestation type ECDA and the  
3988 identifier of the ECDA-Issuer public key ecdaaKeyId.  
3989  
3990 **8.3.1. TPM attestation statement certificate requirements**  
3991 TPM attestation certificate MUST have the following fields/extensions:  
3992 \* Version MUST be set to 3.  
3993 \* Subject field MUST be set to empty.  
3994 \* The Subject Alternative Name extension MUST be set as defined in  
3995 [TPMv2-EK-Profile] section 3.2.9.  
3996 \* The Extended Key Usage extension MUST contain the  
3997 "joint-iso-itu-t(2) internationalorganizations(23) 133 tcg-kp(8)  
3998 tcg-kp-AIKCertificate(3)" OID.  
3999 \* The Basic Constraints extension MUST have the CA component set to  
4000 false.  
4001 \* An Authority Information Access (AIA) extension with entry  
4002 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are  
4003 both OPTIONAL as the status of many attestation certificates is  
4004 available through metadata services. See, for example, the FIDO  
4005 Metadata Service [FIDOMetadataService].  
4006  
4007 **8.4. Android Key Attestation Statement Format**  
4008 When the authenticator in question is a platform-provided Authenticator  
4009 on the Android "N" or later platform, the attestation statement is  
4010 based on the Android key attestation. In these cases, the attestation  
4011 statement is produced by a component running in a secure operating  
4012 environment, but the authenticator data for the attestation is produced  
4013 outside this environment. The Relying Party is expected to check that  
4014 the authenticator data claimed to have been used for the attestation is  
4015 consistent with the fields of the attestation certificate's extension  
4016 data.  
4017  
4018 Attestation statement format identifier  
4019 android-key  
4020  
4021 Attestation types supported

```

3983 Basic
3984
3985 Syntax
3986 An Android key attestation statement consists simply of the
3987 Android attestation statement, which is a series of DER encoded
3988 X.509 certificates. See the Android developer documentation. Its
3989 syntax is defined as follows:
3990
3991 $$attStmtType ::= (
3992     fmt: "android-key",
3993     attStmt: androidStmtFormat
3994 )
3995
3996 androidStmtFormat = {
3997     alg: COSEAlgorithmIdentifier,
3998     sig: bytes,
3999     x5c: [ credCert: bytes, * (caCert: bytes) ]
4000 }
4001
4002 Signing procedure
4003 Let authenticatorData denote the authenticator data for the
4004 attestation, and let clientDataHash denote the hash of the
4005 serialized client data.
4006
4007 Request an Android Key Attestation by calling
4008 keyStore.getCertificateChain(myKeyUUID) providing clientDataHash
4009 as the challenge value (e.g., by using setAttestationChallenge).
4010 Set x5c to the returned value.
4011
4012 The authenticator produces sig by concatenating
4013 authenticatorData and clientDataHash, and signing the result
4014 using the credential private key. It sets alg to the algorithm
4015 of the signature format.
4016
4017 Verification procedure
4018 Given the verification procedure inputs attStmt,
4019 authenticatorData and clientDataHash, the verification procedure
4020 is as follows:
4021
4022 + Verify that attStmt is valid CBOR conforming to the syntax
4023 defined above and perform CBOR decoding on it to extract the
4024 contained fields.
4025 + Verify that sig is a valid signature over the concatenation of
4026 authenticatorData and clientDataHash using the public key in
4027 the first certificate in x5c with the algorithm specified in
4028 alg.
4029 + Verify that the public key in the first certificate in in x5c
4030 matches the credentialPublicKey in the attestedCredentialData
4031 in authenticatorData.
4032 + Verify that in the attestation certificate extension data:
4033     o The value of the attestationChallenge field is identical
4034     to clientDataHash.
4035     o The AuthorizationList.allApplications field is not
4036     present, since PublicKeyCredential must be bound to the
4037     RP ID.
4038     o The value in the AuthorizationList.origin field is equal
4039     to KM_TAG_GENERATED.
4040     o The value in the AuthorizationList.purpose field is equal
4041     to KM_PURPOSE_SIGN.
4042 + If successful, return attestation type Basic with the
4043 attestation trust path set to x5c.
4044
4045 8.5. Android SafetyNet Attestation Statement Format
4046
4047 When the authenticator in question is a platform-provided Authenticator
4048 on certain Android platforms, the attestation statement is based on the
4049 SafetyNet API. In this case the authenticator data is completely
4050 controlled by the caller of the SafetyNet API (typically an application
4051 running on the Android platform) and the attestation statement only
4052

```

```

4025 Basic
4026
4027 Syntax
4028 An Android key attestation statement consists simply of the
4029 Android attestation statement, which is a series of DER encoded
4030 X.509 certificates. See the Android developer documentation. Its
4031 syntax is defined as follows:
4032
4033 $$attStmtType ::= (
4034     fmt: "android-key",
4035     attStmt: androidStmtFormat
4036 )
4037
4038 androidStmtFormat = {
4039     alg: COSEAlgorithmIdentifier,
4040     sig: bytes,
4041     x5c: [ credCert: bytes, * (caCert: bytes) ]
4042 }
4043
4044 Signing procedure
4045 Let authenticatorData denote the authenticator data for the
4046 attestation, and let clientDataHash denote the hash of the
4047 serialized client data.
4048
4049 Request an Android Key Attestation by calling
4050 keyStore.getCertificateChain(myKeyUUID) providing clientDataHash
4051 as the challenge value (e.g., by using setAttestationChallenge).
4052 Set x5c to the returned value.
4053
4054 The authenticator produces sig by concatenating
4055 authenticatorData and clientDataHash, and signing the result
4056 using the credential private key. It sets alg to the algorithm
4057 of the signature format.
4058
4059 Verification procedure
4060 Given the verification procedure inputs attStmt,
4061 authenticatorData and clientDataHash, the verification procedure
4062 is as follows:
4063
4064 + Verify that attStmt is valid CBOR conforming to the syntax
4065 defined above and perform CBOR decoding on it to extract the
4066 contained fields.
4067 + Verify that sig is a valid signature over the concatenation of
4068 authenticatorData and clientDataHash using the public key in
4069 the first certificate in x5c with the algorithm specified in
4070 alg.
4071 + Verify that the public key in the first certificate in in x5c
4072 matches the credentialPublicKey in the attestedCredentialData
4073 in authenticatorData.
4074 + Verify that in the attestation certificate extension data:
4075     o The value of the attestationChallenge field is identical
4076     to clientDataHash.
4077     o The AuthorizationList.allApplications field is not
4078     present, since PublicKeyCredential must be bound to the
4079     RP ID.
4080     o The value in the AuthorizationList.origin field is equal
4081     to KM_TAG_GENERATED.
4082     o The value in the AuthorizationList.purpose field is equal
4083     to KM_PURPOSE_SIGN.
4084 + If successful, return attestation type Basic with the
4085 attestation trust path set to x5c.
4086
4087 8.5. Android SafetyNet Attestation Statement Format
4088
4089 When the authenticator in question is a platform-provided Authenticator
4090 on certain Android platforms, the attestation statement is based on the
4091 SafetyNet API. In this case the authenticator data is completely
4092 controlled by the caller of the SafetyNet API (typically an application
4093 running on the Android platform) and the attestation statement only
4094

```

4053 provides some statements about the health of the platform and the  
4054 identity of the calling application. This attestation does not provide  
4055 information regarding provenance of the authenticator and its  
4056 associated data. Therefore platform-provided authenticators should make  
4057 use of the Android Key Attestation when available, even if the  
4058 SafetyNet API is also present.

4060 Attestation statement format identifier  
4061 android-safetynet

4063 Attestation types supported  
4064 Basic

#### 4066 Syntax

4067 The syntax of an Android Attestation statement is defined as  
4068 follows:

```
4070 $$attStmtType ::= (
4071     fmt: "android-safetynet",
4072     attStmt: safetynetStmtFormat
4073 )
```

```
4075 safetynetStmtFormat = {
4076     ver: text,
4077     response: bytes
4078 }
```

4079 The semantics of the above fields are as follows:

4081 ver The version number of Google Play Services responsible for  
4082 providing the SafetyNet API.

4084 response The UTF-8 encoded result of the getJwsResult() call of the  
4085 SafetyNet API. This value is a JWS [RFC7515] object (see  
4086 SafetyNet online documentation) in Compact Serialization.

#### 4091 Signing procedure

4092 Let authenticatorData denote the authenticator data for the  
4093 attestation, and let clientDataHash denote the hash of the  
4094 serialized client data.

4096 Concatenate authenticatorData and clientDataHash, perform  
4097 SHA-256 hash of the concatenated string, and let the result of  
4098 the hash form attToBeSigned.

4100 Request a SafetyNet attestation, providing attToBeSigned as the  
4101 nonce value. Set response to the result, and ver to the version  
4102 of Google Play Services running in the authenticator.

#### 4104 Verification procedure

4105 Given the verification procedure inputs attStmt,  
4106 authenticatorData and clientDataHash, the verification procedure  
4107 is as follows:

- 4109 + Verify that attStmt is valid CBOR conforming to the syntax  
4110 defined above and perform CBOR decoding on it to extract the  
4111 contained fields.
- 4112 + Verify that response is a valid SafetyNet response of version  
4113 ver.
- 4114 + Verify that the nonce in the response is identical to the  
4115 SHA-256 hash of the concatenation of authenticatorData and  
4116 clientDataHash.
- 4117 + Verify that the attestation certificate is issued to the  
4118 hostname "attest.android.com" (see SafetyNet online  
4119 documentation).
- 4120 + Verify that the ctsProfileMatch attribute in the payload of  
4121 response is true.
- 4122 + If successful, return attestation type Basic with the

4095 provides some statements about the health of the platform and the  
4096 identity of the calling application. This attestation does not provide  
4097 information regarding provenance of the authenticator and its  
4098 associated data. Therefore platform-provided authenticators should make  
4099 use of the Android Key Attestation when available, even if the  
4100 SafetyNet API is also present.

4102 Attestation statement format identifier  
4103 android-safetynet

4105 Attestation types supported  
4106 Basic

#### 4108 Syntax

4109 The syntax of an Android Attestation statement is defined as  
4110 follows:

```
4112 $$attStmtType ::= (
4113     fmt: "android-safetynet",
4114     attStmt: safetynetStmtFormat
4115 )
```

```
4117 safetynetStmtFormat = {
4118     ver: text,
4119     response: bytes
4120 }
```

4121 The semantics of the above fields are as follows:

4123 ver The version number of Google Play Services responsible for  
4124 providing the SafetyNet API.

4126 response The UTF-8 encoded result of the getJwsResult() call of the  
4127 SafetyNet API. This value is a JWS [RFC7515] object (see  
4128 SafetyNet online documentation) in Compact Serialization.

#### 4131 Signing procedure

4132 Let authenticatorData denote the authenticator data for the  
4133 attestation, and let clientDataHash denote the hash of the  
4134 serialized client data.

4136 Concatenate authenticatorData and clientDataHash, perform  
4137 SHA-256 hash of the concatenated string, and let the result of  
4138 the hash form attToBeSigned.

4140 Request a SafetyNet attestation, providing attToBeSigned as the  
4141 nonce value. Set response to the result, and ver to the version  
4142 of Google Play Services running in the authenticator.

#### 4144 Verification procedure

4145 Given the verification procedure inputs attStmt,  
4146 authenticatorData and clientDataHash, the verification procedure  
4147 is as follows:

- 4149 + Verify that attStmt is valid CBOR conforming to the syntax  
4150 defined above and perform CBOR decoding on it to extract the  
4151 contained fields.
- 4152 + Verify that response is a valid SafetyNet response of version  
4153 ver.
- 4154 + Verify that the nonce in the response is identical to the  
4155 SHA-256 hash of the concatenation of authenticatorData and  
4156 clientDataHash.
- 4157 + Verify that the attestation certificate is issued to the  
4158 hostname "attest.android.com" (see SafetyNet online  
4159 documentation).
- 4160 + Verify that the ctsProfileMatch attribute in the payload of  
4161 response is true.
- 4162 + If successful, return attestation type Basic with the

4123 attestation trust path set to the above attestation  
 4124 certificate.  
 4125  
 4126 **8.6. FIDO U2F Attestation Statement Format**  
 4127  
 4128 This attestation statement format is used with FIDO U2F authenticators  
 4129 using the formats defined in [FIDO-U2F-Message-Formats].  
 4130  
 4131 Attestation statement format identifier  
 4132 fido-u2f  
 4133  
 4134 Attestation types supported  
 4135 Basic, AttCA  
 4136  
 4137 Syntax  
 4138 The syntax of a FIDO U2F attestation statement is defined as  
 4139 follows:  
 4140  
 4141 `$$attStmtType //=  
 4142 fmt: "fido-u2f",  
 4143 attStmt: u2fStmtFormat  
 4144 )`  
 4145  
 4146 `u2fStmtFormat = {  
 4147 x5c: [ attestnCert: bytes ],  
 4148 sig: bytes  
 4149 }`  
 4150  
 4151 The semantics of the above fields are as follows:  
 4152  
 4153 **x5c**  
 4154 A single element array containing the attestation  
 4155 certificate in X.509 format.  
 4156  
 4157 **sig**  
 4158 The attestation signature. The signature was calculated  
 4159 over the (raw) U2F registration response message  
 4160 [FIDO-U2F-Message-Formats] received by the platform from  
 4161 the authenticator.  
 4162  
 4163 **Signing procedure**  
 4164 If the credential public key of the given credential is not of  
 4165 algorithm -7 ("ES256"), stop and return an error. Otherwise, let  
 4166 authenticatorData denote the authenticator data for the  
 4167 attestation, and let clientDataHash denote the hash of the  
 4168 serialized client data. (Since SHA-256 is used to hash the  
 4169 serialized client data, clientDataHash will be 32 bytes long.)  
 4170  
 4171 Generate a Registration Response Message as specified in  
 4172 [FIDO-U2F-Message-Formats] Section 4.3, with the application  
 4173 parameter set to the SHA-256 hash of the RP ID associated with  
 4174 the given credential, the challenge parameter set to  
 4175 clientDataHash, and the key handle parameter set to the  
 4176 credential ID of the given credential. Set the raw signature  
 4177 part of this Registration Response Message (i.e., without the  
 4178 user public key, key handle, and attestation certificates) as  
 4179 sig and set the attestation certificates of the attestation  
 4180 public key as x5c.  
 4181  
 4182 **Verification procedure**  
 4183 Given the verification procedure inputs attStmt,  
 4184 authenticatorData and clientDataHash, the verification procedure  
 4185 is as follows:  
 4186  
 4187 1. Verify that attStmt is valid CBOR conforming to the syntax  
 4188 defined above and perform CBOR decoding on it to extract the  
 4189 contained fields.  
 4190 2. Check that x5c has exactly one element and let attCert be that  
 4191 element. Let certificate public key be the public key conveyed  
 4192 by attCert. If certificate public key is not an Elliptic Curve

4165 attestation trust path set to the above attestation  
 4166 certificate.  
 4167  
 4168 **8.6. FIDO U2F Attestation Statement Format**  
 4169  
 4170 This attestation statement format is used with FIDO U2F authenticators  
 4171 using the formats defined in [FIDO-U2F-Message-Formats].  
 4172  
 4173 Attestation statement format identifier  
 4174 fido-u2f  
 4175  
 4176 Attestation types supported  
 4177 Basic, AttCA  
 4178  
 4179 Syntax  
 4180 The syntax of a FIDO U2F attestation statement is defined as  
 4181 follows:  
 4182  
 4183 `$$attStmtType //=  
 4184 fmt: "fido-u2f",  
 4185 attStmt: u2fStmtFormat  
 4186 )`  
 4187  
 4188 `u2fStmtFormat = {  
 4189 x5c: [ attestnCert: bytes ],  
 4190 sig: bytes  
 4191 }`  
 4192  
 4193 The semantics of the above fields are as follows:  
 4194  
 4195 **x5c**  
 4196 A single element array containing the attestation  
 4197 certificate in X.509 format.  
 4198  
 4199 **sig**  
 4200 The attestation signature. The signature was calculated  
 4201 over the (raw) U2F registration response message  
 4202 [FIDO-U2F-Message-Formats] received by the platform from  
 4203 the authenticator.  
 4204  
 4205 **Signing procedure**  
 4206 If the credential public key of the given credential is not of  
 4207 algorithm -7 ("ES256"), stop and return an error. Otherwise, let  
 4208 authenticatorData denote the authenticator data for the  
 4209 attestation, and let clientDataHash denote the hash of the  
 4210 serialized client data. (Since SHA-256 is used to hash the  
 4211 serialized client data, clientDataHash will be 32 bytes long.)  
 4212  
 4213 Generate a Registration Response Message as specified in  
 4214 [FIDO-U2F-Message-Formats] Section 4.3, with the application  
 4215 parameter set to the SHA-256 hash of the RP ID associated with  
 4216 the given credential, the challenge parameter set to  
 4217 clientDataHash, and the key handle parameter set to the  
 4218 credential ID of the given credential. Set the raw signature  
 4219 part of this Registration Response Message (i.e., without the  
 4220 user public key, key handle, and attestation certificates) as  
 4221 sig and set the attestation certificates of the attestation  
 4222 public key as x5c.  
 4223  
 4224 **Verification procedure**  
 4225 Given the verification procedure inputs attStmt,  
 4226 authenticatorData and clientDataHash, the verification procedure  
 4227 is as follows:  
 4228  
 4229 1. Verify that attStmt is valid CBOR conforming to the syntax  
 4230 defined above and perform CBOR decoding on it to extract the  
 4231 contained fields.  
 4232 2. Check that x5c has exactly one element and let attCert be that  
 4233 element. Let certificate public key be the public key conveyed  
 4234 by attCert. If certificate public key is not an Elliptic Curve

4193 (EC) public key over the P-256 curve, terminate this algorithm  
 4194 and return an appropriate error.  
 4195 3. Extract the claimed rpldHash from authenticatorData, and the  
 4196 claimed credentialId and credentialPublicKey from  
 4197 authenticatorData.attestedCredentialData.  
 4198 4. Convert the COSE\_KEY formatted credentialPublicKey (see  
 4199 Section 7 of [RFC8152]) to Raw ANSI X9.62 public key format  
 4200 (see ALG\_KEY\_ECC\_X962\_RAW in Section 3.6.2 Public Key  
 4201 Representation Formats of [FIDO-Registry]).  
 4202 o Let x be the value corresponding to the "-2" key  
 4203 (representing x coordinate) in credentialPublicKey, and  
 4204 confirm its size to be of 32 bytes. If size differs or  
 4205 "-2" key is not found, terminate this algorithm and  
 4206 return an appropriate error.  
 4207 o Let y be the value corresponding to the "-3" key  
 4208 (representing y coordinate) in credentialPublicKey, and  
 4209 confirm its size to be of 32 bytes. If size differs or  
 4210 "-3" key is not found, terminate this algorithm and  
 4211 return an appropriate error.  
 4212 o Let publicKeyU2F be the concatenation 0x04 || x || y.  
 4213 Note: This signifies uncompressed ECC key format.  
 4214 5. Let verificationData be the concatenation of (0x00 || rpldHash  
 4215 || clientDataHash || credentialId || publicKeyU2F) (see  
 4216 Section 4.3 of [FIDO-U2F-Message-Formats]).  
 4217 6. Verify the sig using verificationData and certificate public  
 4218 key per [SEC1].  
 4219 7. If successful, return attestation type Basic with the  
 4220 attestation trust path set to x5c.

### 8.7. None Attestation Statement Format

4221 The none attestation statement format is used to replace any  
 4222 authenticator-provided attestation statement when a Relying Party  
 4223 indicates it does not wish to receive attestation information, see  
 4224 5.4.6 Attestation Conveyance Preference enumeration (enum  
 4225 AttestationConveyancePreference).

4226 Attestation statement format identifier  
 4227 none

4228 Attestation types supported  
 4229 None

4230 Syntax  
 4231 The syntax of a none attestation statement is defined as  
 4232 follows:

```
4233 $$attStmtType ::= (
4234     fmt: "none",
4235     attStmt: emptyMap
4236 )
```

4237 emptyMap = {}

4238 Signing procedure  
 4239 Return the fixed attestation statement defined above.

4240 Verification procedure  
 4241 Return attestation type None with an empty trust path.

### 9. WebAuthn Extensions

4242 The mechanism for generating public key credentials, as well as  
 4243 requesting and generating Authentication assertions, as defined in 5  
 4244 Web Authentication API, can be extended to suit particular use cases.  
 4245 Each case is addressed by defining a registration extension and/or an  
 4246 authentication extension.

4247 Every extension is a client extension, meaning that the extension  
 4248 involves communication with and processing by the client. Client

4235 (EC) public key over the P-256 curve, terminate this algorithm  
 4236 and return an appropriate error.  
 4237 3. Extract the claimed rpldHash from authenticatorData, and the  
 4238 claimed credentialId and credentialPublicKey from  
 4239 authenticatorData.attestedCredentialData.  
 4240 4. Convert the COSE\_KEY formatted credentialPublicKey (see  
 4241 Section 7 of [RFC8152]) to Raw ANSI X9.62 public key format  
 4242 (see ALG\_KEY\_ECC\_X962\_RAW in Section 3.6.2 Public Key  
 4243 Representation Formats of [FIDO-Registry]).  
 4244 o Let x be the value corresponding to the "-2" key  
 4245 (representing x coordinate) in credentialPublicKey, and  
 4246 confirm its size to be of 32 bytes. If size differs or  
 4247 "-2" key is not found, terminate this algorithm and  
 4248 return an appropriate error.  
 4249 o Let y be the value corresponding to the "-3" key  
 4250 (representing y coordinate) in credentialPublicKey, and  
 4251 confirm its size to be of 32 bytes. If size differs or  
 4252 "-3" key is not found, terminate this algorithm and  
 4253 return an appropriate error.  
 4254 o Let publicKeyU2F be the concatenation 0x04 || x || y.  
 4255 Note: This signifies uncompressed ECC key format.  
 4256 5. Let verificationData be the concatenation of (0x00 || rpldHash  
 4257 || clientDataHash || credentialId || publicKeyU2F) (see  
 4258 Section 4.3 of [FIDO-U2F-Message-Formats]).  
 4259 6. Verify the sig using verificationData and certificate public  
 4260 key per [SEC1].  
 4261 7. If successful, return attestation type Basic with the  
 4262 attestation trust path set to x5c.

### 8.7. None Attestation Statement Format

4263 The none attestation statement format is used to replace any  
 4264 authenticator-provided attestation statement when a Relying Party  
 4265 indicates it does not wish to receive attestation information, see  
 4266 5.4.6 Attestation Conveyance Preference enumeration (enum  
 4267 AttestationConveyancePreference).

4268 Attestation statement format identifier  
 4269 none

4270 Attestation types supported  
 4271 None

4272 Syntax  
 4273 The syntax of a none attestation statement is defined as  
 4274 follows:

```
4275 $$attStmtType ::= (
4276     fmt: "none",
4277     attStmt: emptyMap
4278 )
```

4279 emptyMap = {}

4280 Signing procedure  
 4281 Return the fixed attestation statement defined above.

4282 Verification procedure  
 4283 Return attestation type None with an empty trust path.

### 9. WebAuthn Extensions

4284 The mechanism for generating public key credentials, as well as  
 4285 requesting and generating Authentication assertions, as defined in 5  
 4286 Web Authentication API, can be extended to suit particular use cases.  
 4287 Each case is addressed by defining a registration extension and/or an  
 4288 authentication extension.

4289 Every extension is a client extension, meaning that the extension  
 4290 involves communication with and processing by the client. Client

4263 extensions define the following steps and data:  
 4264 \* navigator.credentials.create() extension request parameters and  
 4265 response values for registration extensions.  
 4266 \* navigator.credentials.get() extension request parameters and  
 4267 response values for authentication extensions.  
 4268 \* Client extension processing for registration extensions and  
 4269 authentication extensions.  
 4270  
 4271 When creating a public key credential or requesting an authentication  
 4272 assertion, a Relying Party can request the use of a set of extensions.  
 4273 These extensions will be invoked during the requested operation if they  
 4274 are supported by the client and/or the authenticator. The Relying Party  
 4275 sends the client extension input for each extension in the get() call  
 4276 (for authentication extensions) or create() call (for registration  
 4277 extensions) to the client platform. The client platform performs client  
 4278 extension processing for each extension that it supports, and augments  
 4279 the client data as specified by each extension, by including the  
 4280 extension identifier and client extension output values.  
 4281  
 4282 An extension can also be an authenticator extension, meaning that the  
 4283 extension involves communication with and processing by the  
 4284 authenticator. Authenticator extensions define the following steps and  
 4285 data:  
 4286 \* authenticatorMakeCredential extension request parameters and  
 4287 response values for registration extensions.  
 4288 \* authenticatorGetAssertion extension request parameters and response  
 4289 values for authentication extensions.  
 4290 \* Authenticator extension processing for registration extensions and  
 4291 authentication extensions.  
 4292  
 4293 For authenticator extensions, as part of the client extension  
 4294 processing, the client also creates the CBOR authenticator extension  
 4295 input value for each extension (often based on the corresponding client  
 4296 extension input value), and passes them to the authenticator in the  
 4297 create() call (for registration extensions) or the get() call (for  
 4298 authentication extensions). These authenticator extension input values  
 4299 are represented in CBOR and passed as name-value pairs, with the  
 4300 extension identifier as the name, and the corresponding authenticator  
 4301 extension input as the value. The authenticator, in turn, performs  
 4302 additional processing for the extensions that it supports, and returns  
 4303 the CBOR authenticator extension output for each as specified by the  
 4304 extension. Part of the client extension processing for authenticator  
 4305 extensions is to use the authenticator extension output as an input to  
 4306 creating the client extension output.  
 4307  
 4308 All WebAuthn extensions are OPTIONAL for both clients and  
 4309 authenticators. Thus, any extensions requested by a Relying Party MAY  
 4310 be ignored by the client browser or OS and not passed to the  
 4311 authenticator at all, or they MAY be ignored by the authenticator.  
 4312 Ignoring an extension is never considered a failure in WebAuthn API  
 4313 processing, so when Relying Parties include extensions with any API  
 4314 calls, they MUST be prepared to handle cases where some or all of those  
 4315 extensions are ignored.  
 4316  
 4317 Clients wishing to support the widest possible range of extensions MAY  
 4318 choose to pass through any extensions that they do not recognize to  
 4319 authenticators, generating the authenticator extension input by simply  
 4320 encoding the client extension input in CBOR. All WebAuthn extensions  
 4321 MUST be defined in such a way that this implementation choice does not  
 4322 endanger the user's security or privacy. For instance, if an extension  
 4323 requires client processing, it could be defined in a manner that  
 4324 ensures such a naive pass-through will produce a semantically invalid  
 4325 authenticator extension input value, resulting in the extension being  
 4326 ignored by the authenticator. Since all extensions are OPTIONAL, this  
 4327 will not cause a functional failure in the API operation. Likewise,  
 4328 clients can choose to produce a client extension output value for an  
 4329 extension that it does not understand by encoding the authenticator  
 4330 extension output value into JSON, provided that the CBOR output uses  
 4331 only types present in JSON.  
 4332

4305 extensions define the following steps and data:  
 4306 \* navigator.credentials.create() extension request parameters and  
 4307 response values for registration extensions.  
 4308 \* navigator.credentials.get() extension request parameters and  
 4309 response values for authentication extensions.  
 4310 \* Client extension processing for registration extensions and  
 4311 authentication extensions.  
 4312  
 4313 When creating a public key credential or requesting an authentication  
 4314 assertion, a Relying Party can request the use of a set of extensions.  
 4315 These extensions will be invoked during the requested operation if they  
 4316 are supported by the client and/or the authenticator. The Relying Party  
 4317 sends the client extension input for each extension in the get() call  
 4318 (for authentication extensions) or create() call (for registration  
 4319 extensions) to the client platform. The client platform performs client  
 4320 extension processing for each extension that it supports, and augments  
 4321 the client data as specified by each extension, by including the  
 4322 extension identifier and client extension output values.  
 4323  
 4324 An extension can also be an authenticator extension, meaning that the  
 4325 extension involves communication with and processing by the  
 4326 authenticator. Authenticator extensions define the following steps and  
 4327 data:  
 4328 \* authenticatorMakeCredential extension request parameters and  
 4329 response values for registration extensions.  
 4330 \* authenticatorGetAssertion extension request parameters and response  
 4331 values for authentication extensions.  
 4332 \* Authenticator extension processing for registration extensions and  
 4333 authentication extensions.  
 4334  
 4335 For authenticator extensions, as part of the client extension  
 4336 processing, the client also creates the CBOR authenticator extension  
 4337 input value for each extension (often based on the corresponding client  
 4338 extension input value), and passes them to the authenticator in the  
 4339 create() call (for registration extensions) or the get() call (for  
 4340 authentication extensions). These authenticator extension input values  
 4341 are represented in CBOR and passed as name-value pairs, with the  
 4342 extension identifier as the name, and the corresponding authenticator  
 4343 extension input as the value. The authenticator, in turn, performs  
 4344 additional processing for the extensions that it supports, and returns  
 4345 the CBOR authenticator extension output for each as specified by the  
 4346 extension. Part of the client extension processing for authenticator  
 4347 extensions is to use the authenticator extension output as an input to  
 4348 creating the client extension output.  
 4349  
 4350 All WebAuthn extensions are OPTIONAL for both clients and  
 4351 authenticators. Thus, any extensions requested by a Relying Party MAY  
 4352 be ignored by the client browser or OS and not passed to the  
 4353 authenticator at all, or they MAY be ignored by the authenticator.  
 4354 Ignoring an extension is never considered a failure in WebAuthn API  
 4355 processing, so when Relying Parties include extensions with any API  
 4356 calls, they MUST be prepared to handle cases where some or all of those  
 4357 extensions are ignored.  
 4358  
 4359 Clients wishing to support the widest possible range of extensions MAY  
 4360 choose to pass through any extensions that they do not recognize to  
 4361 authenticators, generating the authenticator extension input by simply  
 4362 encoding the client extension input in CBOR. All WebAuthn extensions  
 4363 MUST be defined in such a way that this implementation choice does not  
 4364 endanger the user's security or privacy. For instance, if an extension  
 4365 requires client processing, it could be defined in a manner that  
 4366 ensures such a naive pass-through will produce a semantically invalid  
 4367 authenticator extension input value, resulting in the extension being  
 4368 ignored by the authenticator. Since all extensions are OPTIONAL, this  
 4369 will not cause a functional failure in the API operation. Likewise,  
 4370 clients can choose to produce a client extension output value for an  
 4371 extension that it does not understand by encoding the authenticator  
 4372 extension output value into JSON, provided that the CBOR output uses  
 4373 only types present in JSON.  
 4374

4333 When clients choose to pass through extensions they do not recognize,  
4334 the JavaScript values in the client extension inputs are converted to  
4335 CBOR values in the authenticator extension inputs. When the JavaScript  
4336 value is an %ArrayBuffer%, it is converted to a CBOR byte array. When  
4337 the JavaScript value is a non-integer number, it is converted to a  
4338 64-bit CBOR floating point number. Otherwise, when the JavaScript type  
4339 corresponds to a JSON type, the conversion is done using the rules  
4340 defined in Section 4.2 of [RFC7049] (Converting from JSON to CBOR), but  
4341 operating on inputs of JavaScript type values rather than inputs of  
4342 JSON type values. Once these conversions are done, canonicalization of  
4343 the resulting CBOR MUST be performed using the CTAP2 canonical CBOR  
4344 encoding form.

4345 Likewise, when clients receive outputs from extensions they have passed  
4346 through that they do not recognize, the CBOR values in the  
4347 authenticator extension outputs are converted to JavaScript values in  
4348 the client extension outputs. When the CBOR value is a byte string, it  
4349 is converted to a JavaScript %ArrayBuffer% (rather than a  
4350 base64url-encoded string). Otherwise, when the CBOR type corresponds to  
4351 a JSON type, the conversion is done using the rules defined in Section  
4352 4.1 of [RFC7049] (Converting from CBOR to JSON), but producing outputs  
4353 of JavaScript type values rather than outputs of JSON type values.

4354 Note that some clients may choose to implement this pass-through  
4355 capability under a feature flag. Supporting this capability can  
4356 facilitate innovation, allowing authenticators to experiment with new  
4357 extensions and Relying Parties to use them before there is explicit  
4358 support for them in clients.

4359 The IANA "WebAuthn Extension Identifier" registry established by  
4360 [WebAuthn-Registries] can be consulted for an up-to-date list of  
4361 registered WebAuthn Extensions.

4362 9.1. Extension Identifiers

4363 Extensions are identified by a string, called an extension identifier,  
4364 chosen by the extension author.

4365 Extension identifiers SHOULD be registered per [WebAuthn-Registries]  
4366 "Registries for Web Authentication (WebAuthn)". All registered  
4367 extension identifiers are unique amongst themselves as a matter of  
4368 course.

4369 Unregistered extension identifiers SHOULD aim to be globally unique,  
4370 e.g., by including the defining entity such as myCompany\_extension.

4371 All extension identifiers MUST be a maximum of 32 octets in length and  
4372 MUST consist only of printable USASCII characters, excluding backslash  
4373 and doublequote, i.e., VCHAR as defined in [RFC5234] but without %x22  
4374 and %x5c. Implementations MUST match WebAuthn extension identifiers in  
4375 a case-sensitive fashion.

4376 Extensions that may exist in multiple versions should take care to  
4377 include a version in their identifier. In effect, different versions  
4378 are thus treated as different extensions, e.g., myCompany\_extension\_01

4379 10 Defined Extensions defines an initial set of extensions and their  
4380 identifiers. See the IANA "WebAuthn Extension Identifier" registry  
4381 established by [WebAuthn-Registries] for an up-to-date list of  
4382 registered WebAuthn Extension Identifiers.

4383 9.2. Defining extensions

4384 A definition of an extension MUST specify an extension identifier, a  
4385 client extension input argument to be sent via the get() or create()  
4386 call, the client extension processing rules, and a client extension  
4387 output value. If the extension communicates with the authenticator  
4388 (meaning it is an authenticator extension), it MUST also specify the  
4389 CBOR authenticator extension input argument sent via the  
4390 authenticatorGetAssertion or authenticatorMakeCredential call, the

4375 When clients choose to pass through extensions they do not recognize,  
4376 the JavaScript values in the client extension inputs are converted to  
4377 CBOR values in the authenticator extension inputs. When the JavaScript  
4378 value is an %ArrayBuffer%, it is converted to a CBOR byte array. When  
4379 the JavaScript value is a non-integer number, it is converted to a  
4380 64-bit CBOR floating point number. Otherwise, when the JavaScript type  
4381 corresponds to a JSON type, the conversion is done using the rules  
4382 defined in Section 4.2 of [RFC7049] (Converting from JSON to CBOR), but  
4383 operating on inputs of JavaScript type values rather than inputs of  
4384 JSON type values. Once these conversions are done, canonicalization of  
4385 the resulting CBOR MUST be performed using the CTAP2 canonical CBOR  
4386 encoding form.

4387 Likewise, when clients receive outputs from extensions they have passed  
4388 through that they do not recognize, the CBOR values in the  
4389 authenticator extension outputs are converted to JavaScript values in  
4390 the client extension outputs. When the CBOR value is a byte string, it  
4391 is converted to a JavaScript %ArrayBuffer% (rather than a  
4392 base64url-encoded string). Otherwise, when the CBOR type corresponds to  
4393 a JSON type, the conversion is done using the rules defined in Section  
4394 4.1 of [RFC7049] (Converting from CBOR to JSON), but producing outputs  
4395 of JavaScript type values rather than outputs of JSON type values.

4396 Note that some clients may choose to implement this pass-through  
4397 capability under a feature flag. Supporting this capability can  
4398 facilitate innovation, allowing authenticators to experiment with new  
4399 extensions and Relying Parties to use them before there is explicit  
4400 support for them in clients.

4401 The IANA "WebAuthn Extension Identifier" registry established by  
4402 [WebAuthn-Registries] can be consulted for an up-to-date list of  
4403 registered WebAuthn Extensions.

4404 9.1. Extension Identifiers

4405 Extensions are identified by a string, called an extension identifier,  
4406 chosen by the extension author.

4407 Extension identifiers SHOULD be registered per [WebAuthn-Registries]  
4408 "Registries for Web Authentication (WebAuthn)". All registered  
4409 extension identifiers are unique amongst themselves as a matter of  
4410 course.

4411 Unregistered extension identifiers SHOULD aim to be globally unique,  
4412 e.g., by including the defining entity such as myCompany\_extension.

4413 All extension identifiers MUST be a maximum of 32 octets in length and  
4414 MUST consist only of printable USASCII characters, excluding backslash  
4415 and doublequote, i.e., VCHAR as defined in [RFC5234] but without %x22  
4416 and %x5c. Implementations MUST match WebAuthn extension identifiers in  
4417 a case-sensitive fashion.

4418 Extensions that may exist in multiple versions should take care to  
4419 include a version in their identifier. In effect, different versions  
4420 are thus treated as different extensions, e.g., myCompany\_extension\_01

4421 10 Defined Extensions defines an initial set of extensions and their  
4422 identifiers. See the IANA "WebAuthn Extension Identifier" registry  
4423 established by [WebAuthn-Registries] for an up-to-date list of  
4424 registered WebAuthn Extension Identifiers.

4425 9.2. Defining extensions

4426 A definition of an extension MUST specify an extension identifier, a  
4427 client extension input argument to be sent via the get() or create()  
4428 call, the client extension processing rules, and a client extension  
4429 output value. If the extension communicates with the authenticator  
4430 (meaning it is an authenticator extension), it MUST also specify the  
4431 CBOR authenticator extension input argument sent via the  
4432 authenticatorGetAssertion or authenticatorMakeCredential call, the



4403 authenticator extension processing rules, and the CBOR authenticator  
4404 extension output value.  
4405

4406 Any client extension that is processed by the client MUST return a  
4407 client extension output value so that the Relying Party knows that the  
4408 extension was honored by the client. Similarly, any extension that  
4409 requires authenticator processing MUST return an authenticator  
4410 extension output to let the Relying Party know that the extension was  
4411 honored by the authenticator. If an extension does not otherwise  
4412 require any result values, it SHOULD be defined as returning a JSON  
4413 Boolean client extension output result, set to true to signify that the  
4414 extension was understood and processed. Likewise, any authenticator  
4415 extension that does not otherwise require any result values MUST return  
4416 a value and SHOULD return a CBOR Boolean authenticator extension output  
4417 result, set to true to signify that the extension was understood and  
4418 processed.

### 4419 9.3. Extending request parameters

4420

4421 An extension defines one or two request arguments. The client extension  
4422 input, which is a value that can be encoded in JSON, is passed from the  
4423 Relying Party to the client in the get() or create() call, while the  
4424 CBOR authenticator extension input is passed from the client to the  
4425 authenticator for authenticator extensions during the processing of  
4426 these calls.

4427 A Relying Party simultaneously requests the use of an extension and  
4428 sets its client extension input by including an entry in the extensions  
4429 option to the create() or get() call. The entry key is the extension  
4430 identifier and the value is the client extension input.

```
4431 var assertionPromise = navigator.credentials.get({
4432   publicKey: {
4433     // The challenge must be produced by the server, see the Security Consid
4434     erations
4435     challenge: new Uint8Array([4,99,22 /* 29 more random bytes generated by
4436     the server */]),
4437     extensions: {
4438       "webauthnExample_foobar": 42
4439     }
4440   }
4441 });
```

4442

4443 Extension definitions MUST specify the valid values for their client  
4444 extension input. Clients SHOULD ignore extensions with an invalid  
4445 client extension input. If an extension does not require any parameters  
4446 from the Relying Party, it SHOULD be defined as taking a Boolean client  
4447 argument, set to true to signify that the extension is requested by the  
4448 Relying Party.

4449 Extensions that only affect client processing need not specify  
4450 authenticator extension input. Extensions that have authenticator  
4451 processing MUST specify the method of computing the authenticator  
4452 extension input from the client extension input. For extensions that do  
4453 not require input parameters and are defined as taking a Boolean client  
4454 extension input value set to true, this method SHOULD consist of  
4455 passing an authenticator extension input value of true (CBOR major type  
4456 7, value 21).

4457 Note: Extensions should aim to define authenticator arguments that are  
4458 as small as possible. Some authenticators communicate over  
4459 low-bandwidth links such as Bluetooth Low-Energy or NFC.

### 4460 9.4. Client extension processing

4461

4462 Extensions MAY define additional processing requirements on the client  
4463 platform during the creation of credentials or the generation of an  
4464 assertion. The client extension input for the extension is used as an  
4465 input to this client processing. For each supported client extension,  
4466 the client adds an entry to the clientExtensions map with the extension  
4467 identifier as the key, and the extension's client extension input as  
4468

4445 authenticator extension processing rules, and the CBOR authenticator  
4446 extension output value.  
4447

4448 Any client extension that is processed by the client MUST return a  
4449 client extension output value so that the Relying Party knows that the  
4450 extension was honored by the client. Similarly, any extension that  
4451 requires authenticator processing MUST return an authenticator  
4452 extension output to let the Relying Party know that the extension was  
4453 honored by the authenticator. If an extension does not otherwise  
4454 require any result values, it SHOULD be defined as returning a JSON  
4455 Boolean client extension output result, set to true to signify that the  
4456 extension was understood and processed. Likewise, any authenticator  
4457 extension that does not otherwise require any result values MUST return  
4458 a value and SHOULD return a CBOR Boolean authenticator extension output  
4459 result, set to true to signify that the extension was understood and  
4460 processed.

### 4461 9.3. Extending request parameters

4462

4463 An extension defines one or two request arguments. The client extension  
4464 input, which is a value that can be encoded in JSON, is passed from the  
4465 Relying Party to the client in the get() or create() call, while the  
4466 CBOR authenticator extension input is passed from the client to the  
4467 authenticator for authenticator extensions during the processing of  
4468 these calls.

4469 A Relying Party simultaneously requests the use of an extension and  
4470 sets its client extension input by including an entry in the extensions  
4471 option to the create() or get() call. The entry key is the extension  
4472 identifier and the value is the client extension input.

```
4473 var assertionPromise = navigator.credentials.get({
4474   publicKey: {
4475     // The challenge must be produced by the server, see the Security Consid
4476     erations
4477     challenge: new Uint8Array([4,99,22 /* 29 more random bytes generated by
4478     the server */]),
4479     extensions: {
4480       "webauthnExample_foobar": 42
4481     }
4482   }
4483 });
```

4484

4485 Extension definitions MUST specify the valid values for their client  
4486 extension input. Clients SHOULD ignore extensions with an invalid  
4487 client extension input. If an extension does not require any parameters  
4488 from the Relying Party, it SHOULD be defined as taking a Boolean client  
4489 argument, set to true to signify that the extension is requested by the  
4490 Relying Party.

4491 Extensions that only affect client processing need not specify  
4492 authenticator extension input. Extensions that have authenticator  
4493 processing MUST specify the method of computing the authenticator  
4494 extension input from the client extension input. For extensions that do  
4495 not require input parameters and are defined as taking a Boolean client  
4496 extension input value set to true, this method SHOULD consist of  
4497 passing an authenticator extension input value of true (CBOR major type  
4498 7, value 21).

4499 Note: Extensions should aim to define authenticator arguments that are  
4500 as small as possible. Some authenticators communicate over  
4501 low-bandwidth links such as Bluetooth Low-Energy or NFC.

### 4502 9.4. Client extension processing

4503

4504 Extensions MAY define additional processing requirements on the client  
4505 platform during the creation of credentials or the generation of an  
4506 assertion. The client extension input for the extension is used as an  
4507 input to this client processing. For each supported client extension,  
4508 the client adds an entry to the clientExtensions map with the extension  
4509 identifier as the key, and the extension's client extension input as  
4510

4473 the value.  
4474  
4475 Likewise, the client extension outputs are represented as a dictionary  
4476 in the result of getClientExtensionResults() with extension identifiers  
4477 as keys, and the client extension output value of each extension as the  
4478 value. Like the client extension input, the client extension output is  
4479 a value that can be encoded in JSON.  
4480  
4481 Extensions that require authenticator processing MUST define the  
4482 process by which the client extension input can be used to determine  
4483 the CBOR authenticator extension input and the process by which the  
4484 CBOR authenticator extension output can be used to determine the client  
4485 extension output.  
4486  
4487 **9.5. Authenticator extension processing**  
4488  
4489 The CBOR authenticator extension input value of each processed  
4490 authenticator extension is included in the extensions parameter of the  
4491 authenticatorMakeCredential and authenticatorGetAssertion operations.  
4492 The extensions parameter is a CBOR map where each key is an extension  
4493 identifier and the corresponding value is the authenticator extension  
4494 input for that extension.  
4495  
4496 Likewise, the extension output is represented in the extensions part of  
4497 the authenticator data. The extensions part of the authenticator data  
4498 is a CBOR map where each key is an extension identifier and the  
4499 corresponding value is the authenticator extension output for that  
4500 extension.  
4501  
4502 For each supported extension, the authenticator extension processing  
4503 rule for that extension is used create the authenticator extension  
4504 output from the authenticator extension input and possibly also other  
4505 inputs.  
4506  
4507 **10. Defined Extensions**  
4508  
4509 This section defines the initial set of extensions to be registered in  
4510 the IANA "WebAuthn Extension Identifier" registry established by  
4511 [WebAuthn-Registries]. These are RECOMMENDED for implementation by user  
4512 agents targeting broad interoperability.  
4513  
4514 **10.1. FIDO AppID Extension (appid)**  
4515  
4516 This client extension allows Relying Parties that have previously  
4517 registered a credential using the legacy FIDO JavaScript APIs to  
4518 request an assertion. The FIDO APIs use an alternative identifier for  
4519 relying parties called an AppID [FIDO-APPID], and any credentials  
4520 created using those APIs will be bound to that identifier. Without this  
4521 extension, they would need to be re-registered in order to be bound to  
4522 an RP ID.  
4523  
4524 This extension does not allow FIDO-compatible credentials to be  
4525 created. Thus, credentials created with WebAuthn are not backwards  
4526 compatible with the FIDO JavaScript APIs.  
4527  
4528 Extension identifier  
4529 appid  
4530  
4531 Client extension input  
4532 A single USVString specifying a FIDO AppID.  
4533  
4534 partial dictionary AuthenticationExtensionsClientInputs {  
4535 USVString appid;  
4536 };  
4537  
4538 Client extension processing  
4539  
4540 1. If present in a create() call, return a "NotSupportedError"  
4541 DOMException--this extension is only valid when requesting an  
4542 assertion.

4515 the value.  
4516  
4517 Likewise, the client extension outputs are represented as a dictionary  
4518 in the result of getClientExtensionResults() with extension identifiers  
4519 as keys, and the client extension output value of each extension as the  
4520 value. Like the client extension input, the client extension output is  
4521 a value that can be encoded in JSON.  
4522  
4523 Extensions that require authenticator processing MUST define the  
4524 process by which the client extension input can be used to determine  
4525 the CBOR authenticator extension input and the process by which the  
4526 CBOR authenticator extension output can be used to determine the client  
4527 extension output.  
4528  
4529 **9.5. Authenticator extension processing**  
4530  
4531 The CBOR authenticator extension input value of each processed  
4532 authenticator extension is included in the extensions parameter of the  
4533 authenticatorMakeCredential and authenticatorGetAssertion operations.  
4534 The extensions parameter is a CBOR map where each key is an extension  
4535 identifier and the corresponding value is the authenticator extension  
4536 input for that extension.  
4537  
4538 Likewise, the extension output is represented in the extensions part of  
4539 the authenticator data. The extensions part of the authenticator data  
4540 is a CBOR map where each key is an extension identifier and the  
4541 corresponding value is the authenticator extension output for that  
4542 extension.  
4543  
4544 For each supported extension, the authenticator extension processing  
4545 rule for that extension is used create the authenticator extension  
4546 output from the authenticator extension input and possibly also other  
4547 inputs.  
4548  
4549 **10. Defined Extensions**  
4550  
4551 This section defines the initial set of extensions to be registered in  
4552 the IANA "WebAuthn Extension Identifier" registry established by  
4553 [WebAuthn-Registries]. These are RECOMMENDED for implementation by user  
4554 agents targeting broad interoperability.  
4555  
4556 **10.1. FIDO AppID Extension (appid)**  
4557  
4558 This client extension allows Relying Parties that have previously  
4559 registered a credential using the legacy FIDO JavaScript APIs to  
4560 request an assertion. The FIDO APIs use an alternative identifier for  
4561 relying parties called an AppID [FIDO-APPID], and any credentials  
4562 created using those APIs will be bound to that identifier. Without this  
4563 extension, they would need to be re-registered in order to be bound to  
4564 an RP ID.  
4565  
4566 This extension does not allow FIDO-compatible credentials to be  
4567 created. Thus, credentials created with WebAuthn are not backwards  
4568 compatible with the FIDO JavaScript APIs.  
4569  
4570 Extension identifier  
4571 appid  
4572  
4573 Client extension input  
4574 A single USVString specifying a FIDO AppID.  
4575  
4576 partial dictionary AuthenticationExtensionsClientInputs {  
4577 USVString appid;  
4578 };  
4579  
4580 Client extension processing  
4581  
4582 1. If present in a create() call, return a "NotSupportedError"  
4583 DOMException--this extension is only valid when requesting an  
4584 assertion.

4543 2. Let facetId be the result of passing the caller's origin to  
 4544 the FIDO algorithm for determining the FacetID of a calling  
 4545 application.  
 4546 3. Let appld be the extension input.  
 4547 4. Pass facetId and appld to the FIDO algorithm for determining  
 4548 if a caller's FacetID is authorized for an AppID. If that  
 4549 algorithm rejects appld then return a "SecurityError"  
 4550 DOMException.  
 4551 5. When building allowCredentialDescriptorList, if a U2F  
 4552 authenticator indicates that a credential is inapplicable  
 4553 (i.e. by returning SW\_WRONG\_DATA) then the client MUST retry  
 4554 with the U2F application parameter set to the SHA-256 hash of  
 4555 appld. If this results in an applicable credential, the client  
 4556 MUST include the credential in allowCredentialDescriptorList.  
 4557 The value of appld then replaces the rpld parameter of  
 4558 authenticatorGetAssertion.  
 4559  
 4560 Client extension output  
 4561 Returns the value true to indicate to the RP that the extension  
 4562 was acted upon.  
 4563  
 4564 partial dictionary AuthenticationExtensionsClientOutputs {  
 4565 boolean appid;  
 4566 };  
 4567  
 4568 Authenticator extension input  
 4569 None.  
 4570  
 4571 Authenticator extension processing  
 4572 None.  
 4573  
 4574 Authenticator extension output  
 4575 None.  
 4576  
 4577 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
 4578  
 4579 This registration extension and authentication extension allows for a  
 4580 simple form of transaction authorization. A Relying Party can specify a  
 4581 prompt string, intended for display on a trusted device on the  
 4582 authenticator.  
 4583  
 4584 Extension identifier  
 4585 txAuthSimple  
 4586  
 4587 Client extension input  
 4588 A single USVString prompt.  
 4589  
 4590 partial dictionary AuthenticationExtensionsClientInputs {  
 4591 USVString txAuthSimple;  
 4592 };  
 4593  
 4594 Client extension processing  
 4595 None, except creating the authenticator extension input from the  
 4596 client extension input.  
 4597  
 4598 Client extension output  
 4599 Returns the authenticator extension output string UTF-8 decoded  
 4600 into a USVString.  
 4601  
 4602 partial dictionary AuthenticationExtensionsClientOutputs {  
 4603 USVString txAuthSimple;  
 4604 };  
 4605  
 4606 Authenticator extension input  
 4607 The client extension input encoded as a CBOR text string (major  
 4608 type 3).  
 4609  
 4610 CDDL:  
 4611 txAuthSimpleInput = (tstr)  
 4612

4585 2. Let facetId be the result of passing the caller's origin to  
 4586 the FIDO algorithm for determining the FacetID of a calling  
 4587 application.  
 4588 3. Let appld be the extension input.  
 4589 4. Pass facetId and appld to the FIDO algorithm for determining  
 4590 if a caller's FacetID is authorized for an AppID. If that  
 4591 algorithm rejects appld then return a "SecurityError"  
 4592 DOMException.  
 4593 5. When building allowCredentialDescriptorList, if a U2F  
 4594 authenticator indicates that a credential is inapplicable  
 4595 (i.e. by returning SW\_WRONG\_DATA) then the client MUST retry  
 4596 with the U2F application parameter set to the SHA-256 hash of  
 4597 appld. If this results in an applicable credential, the client  
 4598 MUST include the credential in allowCredentialDescriptorList.  
 4599 The value of appld then replaces the rpld parameter of  
 4600 authenticatorGetAssertion.  
 4601  
 4602 Client extension output  
 4603 Returns the value true to indicate to the RP that the extension  
 4604 was acted upon.  
 4605  
 4606 partial dictionary AuthenticationExtensionsClientOutputs {  
 4607 boolean appid;  
 4608 };  
 4609  
 4610 Authenticator extension input  
 4611 None.  
 4612  
 4613 Authenticator extension processing  
 4614 None.  
 4615  
 4616 Authenticator extension output  
 4617 None.  
 4618  
 4619 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
 4620  
 4621 This registration extension and authentication extension allows for a  
 4622 simple form of transaction authorization. A Relying Party can specify a  
 4623 prompt string, intended for display on a trusted device on the  
 4624 authenticator.  
 4625  
 4626 Extension identifier  
 4627 txAuthSimple  
 4628  
 4629 Client extension input  
 4630 A single USVString prompt.  
 4631  
 4632 partial dictionary AuthenticationExtensionsClientInputs {  
 4633 USVString txAuthSimple;  
 4634 };  
 4635  
 4636 Client extension processing  
 4637 None, except creating the authenticator extension input from the  
 4638 client extension input.  
 4639  
 4640 Client extension output  
 4641 Returns the authenticator extension output string UTF-8 decoded  
 4642 into a USVString.  
 4643  
 4644 partial dictionary AuthenticationExtensionsClientOutputs {  
 4645 USVString txAuthSimple;  
 4646 };  
 4647  
 4648 Authenticator extension input  
 4649 The client extension input encoded as a CBOR text string (major  
 4650 type 3).  
 4651  
 4652 CDDL:  
 4653 txAuthSimpleInput = (tstr)  
 4654

4613 Authenticator extension processing  
 4614 The authenticator MUST display the prompt to the user before  
 4615 performing either user verification or test of user presence.  
 4616 The authenticator MAY insert line breaks if needed.  
 4617  
 4618 Authenticator extension output  
 4619 A single CBOR string, representing the prompt as displayed  
 4620 (including any eventual line breaks).  
 4621  
 4622 CDDL:  
 4623 txAuthSimpleOutput = (tstr)  
 4624  
 4625 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
 4626  
 4627 This registration extension and authentication extension allows images  
 4628 to be used as transaction authorization prompts as well. This allows  
 4629 authenticators without a font rendering engine to be used and also  
 4630 supports a richer visual appearance.  
 4631  
 4632 Extension identifier  
 4633 txAuthGeneric  
 4634  
 4635 Client extension input  
 4636 A JavaScript object defined as follows:  
 4637  
 4638 dictionary txAuthGenericArg {  
 4639 required USVString contentType; // MIME-Type of the content, e.g., "image  
 4640 /png"  
 4641 required ArrayBuffer content;  
 4642 };  
 4643  
 4644 partial dictionary AuthenticationExtensionsClientInputs {  
 4645 txAuthGenericArg txAuthGeneric;  
 4646 };  
 4647  
 4648 Client extension processing  
 4649 None, except creating the authenticator extension input from the  
 4650 client extension input.  
 4651  
 4652 Client extension output  
 4653 Returns the authenticator extension output value as an  
 4654 ArrayBuffer.  
 4655  
 4656 partial dictionary AuthenticationExtensionsClientOutputs {  
 4657 ArrayBuffer txAuthGeneric;  
 4658 };  
 4659  
 4660 Authenticator extension input  
 4661 The client extension input encoded as a CBOR map.  
 4662  
 4663 Authenticator extension processing  
 4664 The authenticator MUST display the content to the user before  
 4665 performing either user verification or test of user presence.  
 4666 The authenticator MAY add other information below the content.  
 4667 No changes are allowed to the content itself, i.e., inside  
 4668 content boundary box.  
 4669  
 4670 Authenticator extension output  
 4671 The hash value of the content which was displayed. The  
 4672 authenticator MUST use the same hash algorithm as it uses for  
 4673 the signature itself.  
 4674  
 4675 10.4. Authenticator Selection Extension (authnSel)  
 4676  
 4677 This registration extension allows a Relying Party to guide the  
 4678 selection of the authenticator that will be leveraged when creating the  
 4679 credential. It is intended primarily for Relying Parties that wish to  
 4680 tightly control the experience around credential creation.  
 4681  
 4682 Extension identifier

4655 Authenticator extension processing  
 4656 The authenticator MUST display the prompt to the user before  
 4657 performing either user verification or test of user presence.  
 4658 The authenticator MAY insert line breaks if needed.  
 4659  
 4660 Authenticator extension output  
 4661 A single CBOR string, representing the prompt as displayed  
 4662 (including any eventual line breaks).  
 4663  
 4664 CDDL:  
 4665 txAuthSimpleOutput = (tstr)  
 4666  
 4667 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
 4668  
 4669 This registration extension and authentication extension allows images  
 4670 to be used as transaction authorization prompts as well. This allows  
 4671 authenticators without a font rendering engine to be used and also  
 4672 supports a richer visual appearance.  
 4673  
 4674 Extension identifier  
 4675 txAuthGeneric  
 4676  
 4677 Client extension input  
 4678 A JavaScript object defined as follows:  
 4679  
 4680 dictionary txAuthGenericArg {  
 4681 required USVString contentType; // MIME-Type of the content, e.g., "image  
 4682 /png"  
 4683 required ArrayBuffer content;  
 4684 };  
 4685  
 4686 partial dictionary AuthenticationExtensionsClientInputs {  
 4687 txAuthGenericArg txAuthGeneric;  
 4688 };  
 4689  
 4690 Client extension processing  
 4691 None, except creating the authenticator extension input from the  
 4692 client extension input.  
 4693  
 4694 Client extension output  
 4695 Returns the authenticator extension output value as an  
 4696 ArrayBuffer.  
 4697  
 4698 partial dictionary AuthenticationExtensionsClientOutputs {  
 4699 ArrayBuffer txAuthGeneric;  
 4700 };  
 4701  
 4702 Authenticator extension input  
 4703 The client extension input encoded as a CBOR map.  
 4704  
 4705 Authenticator extension processing  
 4706 The authenticator MUST display the content to the user before  
 4707 performing either user verification or test of user presence.  
 4708 The authenticator MAY add other information below the content.  
 4709 No changes are allowed to the content itself, i.e., inside  
 4710 content boundary box.  
 4711  
 4712 Authenticator extension output  
 4713 The hash value of the content which was displayed. The  
 4714 authenticator MUST use the same hash algorithm as it uses for  
 4715 the signature itself.  
 4716  
 4717 10.4. Authenticator Selection Extension (authnSel)  
 4718  
 4719 This registration extension allows a Relying Party to guide the  
 4720 selection of the authenticator that will be leveraged when creating the  
 4721 credential. It is intended primarily for Relying Parties that wish to  
 4722 tightly control the experience around credential creation.  
 4723  
 4724 Extension identifier

```

4683     authnSel
4684
4685     Client extension input
4686     A sequence of AAGUIDs:
4687
4688     typedef sequence<AAGUID> AuthenticatorSelectionList;
4689
4690     partial dictionary AuthenticationExtensionsClientInputs {
4691     AuthenticatorSelectionList authnSel;
4692     };
4693
4694     Each AAGUID corresponds to an authenticator model that is
4695     acceptable to the Relying Party for this credential creation.
4696     The list is ordered by decreasing preference.
4697
4698     An AAGUID is defined as an array containing the globally unique
4699     identifier of the authenticator model being sought.
4700
4701     typedef BufferSource    AAGUID;
4702
4703     Client extension processing
4704     This extension can only be used during create(). If the client
4705     supports the Authenticator Selection Extension, it MUST use the
4706     first available authenticator whose AAGUID is present in the
4707     AuthenticatorSelectionList. If none of the available
4708     authenticators match a provided AAGUID, the client MUST select
4709     an authenticator from among the available authenticators to
4710     generate the credential.
4711
4712     Client extension output
4713     Returns the value true to indicate to the RP that the extension
4714     was acted upon.
4715
4716     partial dictionary AuthenticationExtensionsClientOutputs {
4717     boolean authnSel;
4718     };
4719
4720     Authenticator extension input
4721     None.
4722
4723     Authenticator extension processing
4724     None.
4725
4726     Authenticator extension output
4727     None.
4728
4729     10.5. Supported Extensions Extension (exts)
4730
4731     This registration extension enables the Relying Party to determine
4732     which extensions the authenticator supports.
4733
4734     Extension identifier
4735     exts
4736
4737     Client extension input
4738     The Boolean value true to indicate that this extension is
4739     requested by the Relying Party.
4740
4741     partial dictionary AuthenticationExtensionsClientInputs {
4742     boolean exts;
4743     };
4744
4745     Client extension processing
4746     None, except creating the authenticator extension input from the
4747     client extension input.
4748
4749     Client extension output
4750     Returns the list of supported extensions as an array of
4751     extension identifier strings.
4752

```

```

4725     authnSel
4726
4727     Client extension input
4728     A sequence of AAGUIDs:
4729
4730     typedef sequence<AAGUID> AuthenticatorSelectionList;
4731
4732     partial dictionary AuthenticationExtensionsClientInputs {
4733     AuthenticatorSelectionList authnSel;
4734     };
4735
4736     Each AAGUID corresponds to an authenticator model that is
4737     acceptable to the Relying Party for this credential creation.
4738     The list is ordered by decreasing preference.
4739
4740     An AAGUID is defined as an array containing the globally unique
4741     identifier of the authenticator model being sought.
4742
4743     typedef BufferSource    AAGUID;
4744
4745     Client extension processing
4746     This extension can only be used during create(). If the client
4747     supports the Authenticator Selection Extension, it MUST use the
4748     first available authenticator whose AAGUID is present in the
4749     AuthenticatorSelectionList. If none of the available
4750     authenticators match a provided AAGUID, the client MUST select
4751     an authenticator from among the available authenticators to
4752     generate the credential.
4753
4754     Client extension output
4755     Returns the value true to indicate to the RP that the extension
4756     was acted upon.
4757
4758     partial dictionary AuthenticationExtensionsClientOutputs {
4759     boolean authnSel;
4760     };
4761
4762     Authenticator extension input
4763     None.
4764
4765     Authenticator extension processing
4766     None.
4767
4768     Authenticator extension output
4769     None.
4770
4771     10.5. Supported Extensions Extension (exts)
4772
4773     This registration extension enables the Relying Party to determine
4774     which extensions the authenticator supports.
4775
4776     Extension identifier
4777     exts
4778
4779     Client extension input
4780     The Boolean value true to indicate that this extension is
4781     requested by the Relying Party.
4782
4783     partial dictionary AuthenticationExtensionsClientInputs {
4784     boolean exts;
4785     };
4786
4787     Client extension processing
4788     None, except creating the authenticator extension input from the
4789     client extension input.
4790
4791     Client extension output
4792     Returns the list of supported extensions as an array of
4793     extension identifier strings.
4794

```

```

4753 typedef sequence<USVString> AuthenticationExtensionsSupported;
4754
4755 partial dictionary AuthenticationExtensionsClientOutputs {
4756   AuthenticationExtensionsSupported exts;
4757 };
4758
4759   Authenticator extension input
4760   The Boolean value true, encoded in CBOR (major type 7, value
4761   21).
4762
4763   Authenticator extension processing
4764   The authenticator sets the authenticator extension output to be
4765   a list of extensions that the authenticator supports, as defined
4766   below. This extension can be added to attestation objects.
4767
4768   Authenticator extension output
4769   The SupportedExtensions extension is a list (CBOR array) of
4770   extension identifier (UTF-8 encoded) strings.
4771
4772 10.6. User Verification Index Extension (uvi)
4773
4774   This registration extension and authentication extension enables use of
4775   a user verification index.
4776
4777   Extension identifier
4778   uvi
4779
4780   Client extension input
4781   The Boolean value true to indicate that this extension is
4782   requested by the Relying Party.
4783
4784 partial dictionary AuthenticationExtensionsClientInputs {
4785   boolean uvi;
4786 };
4787
4788   Client extension processing
4789   None, except creating the authenticator extension input from the
4790   client extension input.
4791
4792   Client extension output
4793   Returns the authenticator extension output as an ArrayBuffer.
4794
4795 partial dictionary AuthenticationExtensionsClientOutputs {
4796   ArrayBuffer uvi;
4797 };
4798
4799   Authenticator extension input
4800   The Boolean value true, encoded in CBOR (major type 7, value
4801   21).
4802
4803   Authenticator extension processing
4804   The authenticator sets the authenticator extension output to be
4805   a user verification index indicating the method used by the user
4806   to authorize the operation, as defined below. This extension can
4807   be added to attestation objects and assertions.
4808
4809   Authenticator extension output
4810   The user verification index (UVI) is a value uniquely
4811   identifying a user verification data record. The UVI is encoded
4812   as CBOR byte string (type 0x58). Each UVI value MUST be specific
4813   to the related key (in order to provide unlinkability). It also
4814   MUST contain sufficient entropy that makes guessing impractical.
4815   UVI values MUST NOT be reused by the Authenticator (for other
4816   biometric data or users).
4817
4818   The UVI data can be used by servers to understand whether an
4819   authentication was authorized by the exact same biometric data
4820   as the initial key generation. This allows the detection and
4821   prevention of "friendly fraud".
4822

```

```

4795 typedef sequence<USVString> AuthenticationExtensionsSupported;
4796
4797 partial dictionary AuthenticationExtensionsClientOutputs {
4798   AuthenticationExtensionsSupported exts;
4799 };
4800
4801   Authenticator extension input
4802   The Boolean value true, encoded in CBOR (major type 7, value
4803   21).
4804
4805   Authenticator extension processing
4806   The authenticator sets the authenticator extension output to be
4807   a list of extensions that the authenticator supports, as defined
4808   below. This extension can be added to attestation objects.
4809
4810   Authenticator extension output
4811   The SupportedExtensions extension is a list (CBOR array) of
4812   extension identifier (UTF-8 encoded) strings.
4813
4814 10.6. User Verification Index Extension (uvi)
4815
4816   This registration extension and authentication extension enables use of
4817   a user verification index.
4818
4819   Extension identifier
4820   uvi
4821
4822   Client extension input
4823   The Boolean value true to indicate that this extension is
4824   requested by the Relying Party.
4825
4826 partial dictionary AuthenticationExtensionsClientInputs {
4827   boolean uvi;
4828 };
4829
4830   Client extension processing
4831   None, except creating the authenticator extension input from the
4832   client extension input.
4833
4834   Client extension output
4835   Returns the authenticator extension output as an ArrayBuffer.
4836
4837 partial dictionary AuthenticationExtensionsClientOutputs {
4838   ArrayBuffer uvi;
4839 };
4840
4841   Authenticator extension input
4842   The Boolean value true, encoded in CBOR (major type 7, value
4843   21).
4844
4845   Authenticator extension processing
4846   The authenticator sets the authenticator extension output to be
4847   a user verification index indicating the method used by the user
4848   to authorize the operation, as defined below. This extension can
4849   be added to attestation objects and assertions.
4850
4851   Authenticator extension output
4852   The user verification index (UVI) is a value uniquely
4853   identifying a user verification data record. The UVI is encoded
4854   as CBOR byte string (type 0x58). Each UVI value MUST be specific
4855   to the related key (in order to provide unlinkability). It also
4856   MUST contain sufficient entropy that makes guessing impractical.
4857   UVI values MUST NOT be reused by the Authenticator (for other
4858   biometric data or users).
4859
4860   The UVI data can be used by servers to understand whether an
4861   authentication was authorized by the exact same biometric data
4862   as the initial key generation. This allows the detection and
4863   prevention of "friendly fraud".
4864

```

4823 As an example, the UVI could be computed as SHA256(KeyID ||  
 4824 SHA256(rawUVI)), where || represents concatenation, and the  
 4825 rawUVI reflects (a) the biometric reference data, (b) the  
 4826 related OS level user ID and (c) an identifier which changes  
 4827 whenever a factory reset is performed for the device, e.g.  
 4828 rawUVI = biometricReferenceData || OSLevelUserID ||  
 4829 FactoryResetCounter.

4830 Example of authenticator data containing one UVI extension

```

4831 ... -- [=RP ID=] hash (32 bytes)
4832 81 -- UP and ED set
4833 00 00 00 01 -- (initial) signature counter
4834 ... -- all public key alg etc.
4835 A1 -- extension: CBOR map of one elemen
4836 t
4837 63 -- Key 1: CBOR text string of 3 byte
4838 s
4839 75 76 69 -- "uvi" [=UTF-8 encoded=] string
4840 58 20 -- Value 1: CBOR byte string with 0x
4841 20 bytes
4842 43 B8 E3 BE 27 95 8C 28 -- the UVI value itself
4843 D5 74 BF 46 8A 85 CF 46
4844 9A 14 F0 E5 16 69 31 DA
4845 4B CF FF C1 BB 11 32 82
    
```

4846 10.7. Location Extension (loc)

4847 The location registration extension and authentication extension  
 4848 provides the client device's current location to the WebAuthn Relying  
 4849 Party.

4850 Extension identifier  
 4851 loc

4852 Client extension input  
 4853 The Boolean value true to indicate that this extension is  
 4854 requested by the Relying Party.

```

4855 partial dictionary AuthenticationExtensionsClientInputs {
4856   boolean loc;
4857 };
    
```

4858 Client extension processing  
 4859 None, except creating the authenticator extension input from the  
 4860 client extension input.

4861 Client extension output  
 4862 Returns a JavaScript object that encodes the location  
 4863 information in the authenticator extension output as a  
 4864 Coordinates value, as defined by [Geolocation-API].

```

4865 partial dictionary AuthenticationExtensionsClientOutputs {
4866   Coordinates loc;
4867 };
    
```

4868 Authenticator extension input  
 4869 The Boolean value true, encoded in CBOR (major type 7, value  
 4870 21).

4871 Authenticator extension processing  
 4872 Determine the Geolocation value.

4873 Authenticator extension output  
 4874 A [Geolocation-API] Coordinates record encoded as a CBOR map.  
 4875 Values represented by the "double" type in JavaScript are  
 4876 represented as 64-bit CBOR floating point numbers. Per the  
 4877 Geolocation specification, the "latitude", "longitude", and  
 4878 "accuracy" values are required and other values such as  
 4879 "altitude" are optional.

4865 As an example, the UVI could be computed as SHA256(KeyID ||  
 4866 SHA256(rawUVI)), where || represents concatenation, and the  
 4867 rawUVI reflects (a) the biometric reference data, (b) the  
 4868 related OS level user ID and (c) an identifier which changes  
 4869 whenever a factory reset is performed for the device, e.g.  
 4870 rawUVI = biometricReferenceData || OSLevelUserID ||  
 4871 FactoryResetCounter.

4872 Example of authenticator data containing one UVI extension

```

4873 ... -- [=RP ID=] hash (32 bytes)
4874 81 -- UP and ED set
4875 00 00 00 01 -- (initial) signature counter
4876 ... -- all public key alg etc.
4877 A1 -- extension: CBOR map of one elemen
4878 t
4879 63 -- Key 1: CBOR text string of 3 byte
4880 s
4881 75 76 69 -- "uvi" [=UTF-8 encoded=] string
4882 58 20 -- Value 1: CBOR byte string with 0x
4883 20 bytes
4884 43 B8 E3 BE 27 95 8C 28 -- the UVI value itself
4885 D5 74 BF 46 8A 85 CF 46
4886 9A 14 F0 E5 16 69 31 DA
4887 4B CF FF C1 BB 11 32 82
    
```

4888 10.7. Location Extension (loc)

4889 The location registration extension and authentication extension  
 4890 provides the client device's current location to the WebAuthn Relying  
 4891 Party.

4892 Extension identifier  
 4893 loc

4894 Client extension input  
 4895 The Boolean value true to indicate that this extension is  
 4896 requested by the Relying Party.

```

4897 partial dictionary AuthenticationExtensionsClientInputs {
4898   boolean loc;
4899 };
    
```

4900 Client extension processing  
 4901 None, except creating the authenticator extension input from the  
 4902 client extension input.

4903 Client extension output  
 4904 Returns a JavaScript object that encodes the location  
 4905 information in the authenticator extension output as a  
 4906 Coordinates value, as defined by [Geolocation-API].

```

4907 partial dictionary AuthenticationExtensionsClientOutputs {
4908   Coordinates loc;
4909 };
    
```

4910 Authenticator extension input  
 4911 The Boolean value true, encoded in CBOR (major type 7, value  
 4912 21).

4913 Authenticator extension processing  
 4914 Determine the Geolocation value.

4915 Authenticator extension output  
 4916 A [Geolocation-API] Coordinates record encoded as a CBOR map.  
 4917 Values represented by the "double" type in JavaScript are  
 4918 represented as 64-bit CBOR floating point numbers. Per the  
 4919 Geolocation specification, the "latitude", "longitude", and  
 4920 "accuracy" values are required and other values such as  
 4921 "altitude" are optional.

4893  
4894 **10.8. User Verification Method Extension (uvm)**  
4895  
4896 This registration extension and authentication extension enables use of  
4897 a user verification method.  
4898  
4899 **Extension identifier**  
4900 uvm  
4901  
4902 **Client extension input**  
4903 The Boolean value true to indicate that this extension is  
4904 requested by the Relying Party.  
4905  
4906 partial dictionary AuthenticationExtensionsClientInputs {  
4907 boolean uvm;  
4908 };  
4909  
4910 **Client extension processing**  
4911 None, except creating the authenticator extension input from the  
4912 client extension input.  
4913  
4914 **Client extension output**  
4915 Returns a JSON array of 3-element arrays of numbers that encodes  
4916 the factors in the authenticator extension output.  
4917  
4918 typedef sequence<unsigned long> UvmEntry;  
4919 typedef sequence<UvmEntry> UvmEntries;  
4920  
4921 partial dictionary AuthenticationExtensionsClientOutputs {  
4922 UvmEntries uvm;  
4923 };  
4924  
4925 **Authenticator extension input**  
4926 The Boolean value true, encoded in CBOR (major type 7, value  
4927 21).  
4928  
4929 **Authenticator extension processing**  
4930 The authenticator sets the authenticator extension output to be  
4931 one or more user verification methods indicating the method(s)  
4932 used by the user to authorize the operation, as defined below.  
4933 This extension can be added to attestation objects and  
4934 assertions.  
4935  
4936 **Authenticator extension output**  
4937 Authenticators can report up to 3 different user verification  
4938 methods (factors) used in a single authentication instance,  
4939 using the CBOR syntax defined below:  
4940  
4941 uvmFormat = [ 1\*3 uvmEntry ]  
4942 uvmEntry = [  
4943 userVerificationMethod: uint .size 4,  
4944 keyProtectionType: uint .size 2,  
4945 matcherProtectionType: uint .size 2  
4946 ]  
4947  
4948 The semantics of the fields in each uvmEntry are as follows:  
4949  
4950 **userVerificationMethod**  
4951 The authentication method/factor used by the authenticator  
4952 to verify the user. Available values are defined in  
4953 Section 3.1 User Verification Methods of [FIDO-Registry].  
4954  
4955 **keyProtectionType**  
4956 The method used by the authenticator to protect the FIDO  
4957 registration private key material. Available values are  
4958 defined in Section 3.2 Key Protection Types of  
4959 [FIDO-Registry].  
4960  
4961 **matcherProtectionType**  
4962 The method used by the authenticator to protect the

4935  
4936 **10.8. User Verification Method Extension (uvm)**  
4937  
4938 This registration extension and authentication extension enables use of  
4939 a user verification method.  
4940  
4941 **Extension identifier**  
4942 uvm  
4943  
4944 **Client extension input**  
4945 The Boolean value true to indicate that this extension is  
4946 requested by the Relying Party.  
4947  
4948 partial dictionary AuthenticationExtensionsClientInputs {  
4949 boolean uvm;  
4950 };  
4951  
4952 **Client extension processing**  
4953 None, except creating the authenticator extension input from the  
4954 client extension input.  
4955  
4956 **Client extension output**  
4957 Returns a JSON array of 3-element arrays of numbers that encodes  
4958 the factors in the authenticator extension output.  
4959  
4960 typedef sequence<unsigned long> UvmEntry;  
4961 typedef sequence<UvmEntry> UvmEntries;  
4962  
4963 partial dictionary AuthenticationExtensionsClientOutputs {  
4964 UvmEntries uvm;  
4965 };  
4966  
4967 **Authenticator extension input**  
4968 The Boolean value true, encoded in CBOR (major type 7, value  
4969 21).  
4970  
4971 **Authenticator extension processing**  
4972 The authenticator sets the authenticator extension output to be  
4973 one or more user verification methods indicating the method(s)  
4974 used by the user to authorize the operation, as defined below.  
4975 This extension can be added to attestation objects and  
4976 assertions.  
4977  
4978 **Authenticator extension output**  
4979 Authenticators can report up to 3 different user verification  
4980 methods (factors) used in a single authentication instance,  
4981 using the CBOR syntax defined below:  
4982  
4983 uvmFormat = [ 1\*3 uvmEntry ]  
4984 uvmEntry = [  
4985 userVerificationMethod: uint .size 4,  
4986 keyProtectionType: uint .size 2,  
4987 matcherProtectionType: uint .size 2  
4988 ]  
4989  
4990 The semantics of the fields in each uvmEntry are as follows:  
4991  
4992 **userVerificationMethod**  
4993 The authentication method/factor used by the authenticator  
4994 to verify the user. Available values are defined in  
4995 Section 3.1 User Verification Methods of [FIDO-Registry].  
4996  
4997 **keyProtectionType**  
4998 The method used by the authenticator to protect the FIDO  
4999 registration private key material. Available values are  
5000 defined in Section 3.2 Key Protection Types of  
5001 [FIDO-Registry].  
5002  
5003 **matcherProtectionType**  
5004 The method used by the authenticator to protect the



4963 matcher that performs user verification. Available values  
 4964 are defined in Section 3.3 Matcher Protection Types of  
 4965 [FIDO-Registry].  
 4966  
 4967 If >3 factors can be used in an authentication instance the  
 4968 authenticator vendor MUST select the 3 factors it believes will  
 4969 be most relevant to the Server to include in the UVM.  
 4970  
 4971 Example for authenticator data containing one UVM extension for  
 4972 a multi-factor authentication instance where 2 factors were  
 4973 used:  
 4974  
 4975 ... -- [=RP ID=] hash (32 bytes)  
 4976 81 -- UP and ED set  
 4977 00 00 00 01 -- (initial) signature counter  
 4978 ... -- all public key alg etc.  
 4979 A1 -- extension: CBOR map of one element  
 4980 63 -- Key 1: CBOR text string of 3 bytes  
 4981 75 76 6d -- "uvm" [=UTF-8 encoded=] string  
 4982 82 -- Value 1: CBOR array of length 2 indicating two factor  
 4983 usage  
 4984 83 -- Item 1: CBOR array of length 3  
 4985 02 -- Subitem 1: CBOR integer for User Verification Method  
 4986 Fingerprint  
 4987 04 -- Subitem 2: CBOR short for Key Protection Type TEE  
 4988 02 -- Subitem 3: CBOR short for Matcher Protection Type TE  
 4989 E  
 4990 83 -- Item 2: CBOR array of length 3  
 4991 04 -- Subitem 1: CBOR integer for User Verification Method  
 4992 Passcode  
 4993 01 -- Subitem 2: CBOR short for Key Protection Type Softwa  
 4994 re  
 4995 01 -- Subitem 3: CBOR short for Matcher Protection Type So  
 4996 ftware  
 4997  
 4998 10.9. Biometric Authenticator Performance Bounds Extension  
 4999 (biometricPerfBounds)  
 5000  
 5001 This registration extension allows Relying Parties to specify the  
 5002 desired performance bounds for selecting biometric authenticators as  
 5003 candidates to be employed in a registration ceremony.  
 5004  
 5005 Extension identifier  
 5006 biometricPerfBounds  
 5007  
 5008 Client extension input  
 5009 Biometric performance bounds:  
 5010  
 5011 dictionary authenticatorBiometricPerfBounds{  
 5012 float FAR;  
 5013 float FRR;  
 5014 };  
 5015  
 5016 The FAR is the maximum false acceptance rate for a biometric  
 5017 authenticator allowed by the Relying Party.  
 5018  
 5019 The FAR is the maximum false rejection rate for a biometric  
 5020 authenticator allowed by the Relying Party.  
 5021  
 5022 Client extension processing  
 5023 This extension can only be used during create(). If the client  
 5024 supports this extension, it MUST NOT use a biometric  
 5025 authenticator whose FAR or FRR does not match the bounds as  
 5026 provided. The client can obtain information about the biometric  
 5027 authenticator's performance from authoritative sources such as  
 5028 the FIDO Metadata Service [FIDOMetadataService] (see Sec. 3.2 of  
 5029 [FIDOUAFAuthenticatorMetadataStatements]).  
 5030  
 5031 Client extension output  
 5032 Returns the JSON value true to indicate to the RP that the

5005 matcher that performs user verification. Available values  
 5006 are defined in Section 3.3 Matcher Protection Types of  
 5007 [FIDO-Registry].  
 5008  
 5009 If >3 factors can be used in an authentication instance the  
 5010 authenticator vendor MUST select the 3 factors it believes will  
 5011 be most relevant to the Server to include in the UVM.  
 5012  
 5013 Example for authenticator data containing one UVM extension for  
 5014 a multi-factor authentication instance where 2 factors were  
 5015 used:  
 5016  
 5017 ... -- [=RP ID=] hash (32 bytes)  
 5018 81 -- UP and ED set  
 5019 00 00 00 01 -- (initial) signature counter  
 5020 ... -- all public key alg etc.  
 5021 A1 -- extension: CBOR map of one element  
 5022 63 -- Key 1: CBOR text string of 3 bytes  
 5023 75 76 6d -- "uvm" [=UTF-8 encoded=] string  
 5024 82 -- Value 1: CBOR array of length 2 indicating two factor  
 5025 usage  
 5026 83 -- Item 1: CBOR array of length 3  
 5027 02 -- Subitem 1: CBOR integer for User Verification Method  
 5028 Fingerprint  
 5029 04 -- Subitem 2: CBOR short for Key Protection Type TEE  
 5030 02 -- Subitem 3: CBOR short for Matcher Protection Type TE  
 5031 E  
 5032 83 -- Item 2: CBOR array of length 3  
 5033 04 -- Subitem 1: CBOR integer for User Verification Method  
 5034 Passcode  
 5035 01 -- Subitem 2: CBOR short for Key Protection Type Softwa  
 5036 re  
 5037 01 -- Subitem 3: CBOR short for Matcher Protection Type So  
 5038 ftware  
 5039  
 5040 10.9. Biometric Authenticator Performance Bounds Extension  
 5041 (biometricPerfBounds)  
 5042  
 5043 This registration extension allows Relying Parties to specify the  
 5044 desired performance bounds for selecting biometric authenticators as  
 5045 candidates to be employed in a registration ceremony.  
 5046  
 5047 Extension identifier  
 5048 biometricPerfBounds  
 5049  
 5050 Client extension input  
 5051 Biometric performance bounds:  
 5052  
 5053 dictionary authenticatorBiometricPerfBounds{  
 5054 float FAR;  
 5055 float FRR;  
 5056 };  
 5057  
 5058 The FAR is the maximum false acceptance rate for a biometric  
 5059 authenticator allowed by the Relying Party.  
 5060  
 5061 The FAR is the maximum false rejection rate for a biometric  
 5062 authenticator allowed by the Relying Party.  
 5063  
 5064 Client extension processing  
 5065 This extension can only be used during create(). If the client  
 5066 supports this extension, it MUST NOT use a biometric  
 5067 authenticator whose FAR or FRR does not match the bounds as  
 5068 provided. The client can obtain information about the biometric  
 5069 authenticator's performance from authoritative sources such as  
 5070 the FIDO Metadata Service [FIDOMetadataService] (see Sec. 3.2 of  
 5071 [FIDOUAFAuthenticatorMetadataStatements]).  
 5072  
 5073 Client extension output  
 5074 Returns the JSON value true to indicate to the RP that the

5033 extension was acted upon  
5034  
5035 Authenticator extension input  
5036 None.  
5037  
5038 Authenticator extension processing  
5039 None.  
5040  
5041 Authenticator extension output  
5042 None.  
5043  
5044 11. IANA Considerations  
5045  
5046 11.1. WebAuthn Attestation Statement Format Identifier Registrations  
5047  
5048 This section registers the attestation statement formats defined in  
5049 Section 8 Defined Attestation Statement Formats in the IANA "WebAuthn  
5050 Attestation Statement Format Identifier" registry established by  
5051 [WebAuthn-Registries].  
5052 \* WebAuthn Attestation Statement Format Identifier: packed  
5053 \* Description: The "packed" attestation statement format is a  
5054 WebAuthn-optimized format for attestation. It uses a very compact  
5055 but still extensible encoding method. This format is implementable  
5056 by authenticators with limited resources (e.g., secure elements).  
5057 \* Specification Document: Section 8.2 Packed Attestation Statement  
5058 Format of this specification  
5059 \* WebAuthn Attestation Statement Format Identifier: tpm  
5060 \* Description: The TPM attestation statement format returns an  
5061 attestation statement in the same format as the packed attestation  
5062 statement format, although the rawData and signature fields are  
5063 computed differently.  
5064 \* Specification Document: Section 8.3 TPM Attestation Statement  
5065 Format of this specification  
5066 \* WebAuthn Attestation Statement Format Identifier: android-key  
5067 \* Description: Platform-provided authenticators based on versions  
5068 "N", and later, may provide this proprietary "hardware attestation"  
5069 statement.  
5070 \* Specification Document: Section 8.4 Android Key Attestation  
5071 Statement Format of this specification  
5072 \* WebAuthn Attestation Statement Format Identifier: android-safetynet  
5073 \* Description: Android-based, platform-provided authenticators MAY  
5074 produce an attestation statement based on the Android SafetyNet  
5075 API.  
5076 \* Specification Document: Section 8.5 Android SafetyNet Attestation  
5077 Statement Format of this specification  
5078 \* WebAuthn Attestation Statement Format Identifier: fido-u2f  
5079 \* Description: Used with FIDO U2F authenticators  
5080 \* Specification Document: Section 8.6 FIDO U2F Attestation Statement  
5081 Format of this specification  
5082  
5083 11.2. WebAuthn Extension Identifier Registrations  
5084  
5085 This section registers the extension identifier values defined in  
5086 Section 9 WebAuthn Extensions in the IANA "WebAuthn Extension  
5087 Identifier" registry established by [WebAuthn-Registries].  
5088 \* WebAuthn Extension Identifier: appid  
5089 \* Description: This authentication extension allows Relying Parties  
5090 that have previously registered a credential using the legacy FIDO  
5091 JavaScript APIs to request an assertion.  
5092 \* Specification Document: Section 10.1 FIDO AppID Extension (appid)  
5093 of this specification  
5094 \* WebAuthn Extension Identifier: txAuthSimple  
5095 \* Description: This registration extension and authentication  
5096 extension allows for a simple form of transaction authorization. A  
5097 WebAuthn Relying Party can specify a prompt string, intended for  
5098 display on a trusted device on the authenticator  
5099 \* Specification Document: Section 10.2 Simple Transaction  
5100 Authorization Extension (txAuthSimple) of this specification  
5101 \* WebAuthn Extension Identifier: txAuthGeneric  
5102 \* Description: This registration extension and authentication

5075 extension was acted upon  
5076  
5077 Authenticator extension input  
5078 None.  
5079  
5080 Authenticator extension processing  
5081 None.  
5082  
5083 Authenticator extension output  
5084 None.  
5085  
5086 11. IANA Considerations  
5087  
5088 11.1. WebAuthn Attestation Statement Format Identifier Registrations  
5089  
5090 This section registers the attestation statement formats defined in  
5091 Section 8 Defined Attestation Statement Formats in the IANA "WebAuthn  
5092 Attestation Statement Format Identifier" registry established by  
5093 [WebAuthn-Registries].  
5094 \* WebAuthn Attestation Statement Format Identifier: packed  
5095 \* Description: The "packed" attestation statement format is a  
5096 WebAuthn-optimized format for attestation. It uses a very compact  
5097 but still extensible encoding method. This format is implementable  
5098 by authenticators with limited resources (e.g., secure elements).  
5099 \* Specification Document: Section 8.2 Packed Attestation Statement  
5100 Format of this specification  
5101 \* WebAuthn Attestation Statement Format Identifier: tpm  
5102 \* Description: The TPM attestation statement format returns an  
5103 attestation statement in the same format as the packed attestation  
5104 statement format, although the rawData and signature fields are  
5105 computed differently.  
5106 \* Specification Document: Section 8.3 TPM Attestation Statement  
5107 Format of this specification  
5108 \* WebAuthn Attestation Statement Format Identifier: android-key  
5109 \* Description: Platform-provided authenticators based on versions  
5110 "N", and later, may provide this proprietary "hardware attestation"  
5111 statement.  
5112 \* Specification Document: Section 8.4 Android Key Attestation  
5113 Statement Format of this specification  
5114 \* WebAuthn Attestation Statement Format Identifier: android-safetynet  
5115 \* Description: Android-based, platform-provided authenticators MAY  
5116 produce an attestation statement based on the Android SafetyNet  
5117 API.  
5118 \* Specification Document: Section 8.5 Android SafetyNet Attestation  
5119 Statement Format of this specification  
5120 \* WebAuthn Attestation Statement Format Identifier: fido-u2f  
5121 \* Description: Used with FIDO U2F authenticators  
5122 \* Specification Document: Section 8.6 FIDO U2F Attestation Statement  
5123 Format of this specification  
5124  
5125 11.2. WebAuthn Extension Identifier Registrations  
5126  
5127 This section registers the extension identifier values defined in  
5128 Section 9 WebAuthn Extensions in the IANA "WebAuthn Extension  
5129 Identifier" registry established by [WebAuthn-Registries].  
5130 \* WebAuthn Extension Identifier: appid  
5131 \* Description: This authentication extension allows Relying Parties  
5132 that have previously registered a credential using the legacy FIDO  
5133 JavaScript APIs to request an assertion.  
5134 \* Specification Document: Section 10.1 FIDO AppID Extension (appid)  
5135 of this specification  
5136 \* WebAuthn Extension Identifier: txAuthSimple  
5137 \* Description: This registration extension and authentication  
5138 extension allows for a simple form of transaction authorization. A  
5139 WebAuthn Relying Party can specify a prompt string, intended for  
5140 display on a trusted device on the authenticator  
5141 \* Specification Document: Section 10.2 Simple Transaction  
5142 Authorization Extension (txAuthSimple) of this specification  
5143 \* WebAuthn Extension Identifier: txAuthGeneric  
5144 \* Description: This registration extension and authentication

5103 extension allows images to be used as transaction authorization  
5104 prompts as well. This allows authenticators without a font  
5105 rendering engine to be used and also supports a richer visual  
5106 appearance than accomplished with the webauthn.txauth.simple  
5107 extension.  
5108 \* Specification Document: Section 10.3 Generic Transaction  
5109 Authorization Extension (txAuthGeneric) of this specification  
5110 \* WebAuthn Extension Identifier: authnSel  
5111 \* Description: This registration extension allows a WebAuthn Relying  
5112 Party to guide the selection of the authenticator that will be  
5113 leveraged when creating the credential. It is intended primarily  
5114 for WebAuthn Relying Parties that wish to tightly control the  
5115 experience around credential creation.  
5116 \* Specification Document: Section 10.4 Authenticator Selection  
5117 Extension (authnSel) of this specification  
5118 \* WebAuthn Extension Identifier: exts  
5119 \* Description: This registration extension enables the Relying Party  
5120 to determine which extensions the authenticator supports. The  
5121 extension data is a list (CBOR array) of extension identifiers  
5122 encoded as UTF-8 Strings. This extension is added automatically by  
5123 the authenticator. This extension can be added to attestation  
5124 statements.  
5125 \* Specification Document: Section 10.5 Supported Extensions  
5126 Extension (exts) of this specification  
5127 \* WebAuthn Extension Identifier: uvi  
5128 \* Description: This registration extension and authentication  
5129 extension enables use of a user verification index. The user  
5130 verification index is a value uniquely identifying a user  
5131 verification data record. The UVI data can be used by servers to  
5132 understand whether an authentication was authorized by the exact  
5133 same biometric data as the initial key generation. This allows the  
5134 detection and prevention of "friendly fraud".  
5135 \* Specification Document: Section 10.6 User Verification Index  
5136 Extension (uvi) of this specification  
5137 \* WebAuthn Extension Identifier: loc  
5138 \* Description: The location registration extension and authentication  
5139 extension provides the client device's current location to the  
5140 WebAuthn relying party, if supported by the client device and  
5141 subject to user consent.  
5142 \* Specification Document: Section 10.7 Location Extension (loc) of  
5143 this specification  
5144 \* WebAuthn Extension Identifier: uvm  
5145 \* Description: This registration extension and authentication  
5146 extension enables use of a user verification method. The user  
5147 verification method extension returns to the Webauthn relying party  
5148 which user verification methods (factors) were used for the  
5149 WebAuthn operation.  
5150 \* Specification Document: Section 10.8 User Verification Method  
5151 Extension (uvm) of this specification

### 11.3. COSE Algorithm Registrations

5152 This section registers identifiers for the following ECDAA algorithms  
5153 in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG]. Note that  
5154 [WebAuthn-COSE-Algs] also registers RSASSA-PKCS1-v1\_5 [RFC8017]  
5155 algorithms using SHA-2 and SHA-1 hash functions in the IANA COSE  
5156 Algorithms registry [IANA-COSE-ALGS-REG], such as registering -257 for  
5157 "RS256".  
5158 \* Name: ED256  
5159 \* Value: TBD (requested assignment -260)  
5160 \* Description: TPM\_ECC\_BN\_P256 curve w/ SHA-256  
5161 \* Reference: Section 4.2 of [FIDOEcdaaAlgorithm]  
5162 \* Recommended: Yes  
5163 \* Name: ED512  
5164 \* Value: TBD (requested assignment -261)  
5165 \* Description: ECC\_BN\_ISOP512 curve w/ SHA-512  
5166 \* Reference: Section 4.2 of [FIDOEcdaaAlgorithm]  
5167 \* Recommended: Yes

### 12. Sample scenarios

5171  
5172

5145 extension allows images to be used as transaction authorization  
5146 prompts as well. This allows authenticators without a font  
5147 rendering engine to be used and also supports a richer visual  
5148 appearance than accomplished with the webauthn.txauth.simple  
5149 extension.  
5150 \* Specification Document: Section 10.3 Generic Transaction  
5151 Authorization Extension (txAuthGeneric) of this specification  
5152 \* WebAuthn Extension Identifier: authnSel  
5153 \* Description: This registration extension allows a WebAuthn Relying  
5154 Party to guide the selection of the authenticator that will be  
5155 leveraged when creating the credential. It is intended primarily  
5156 for WebAuthn Relying Parties that wish to tightly control the  
5157 experience around credential creation.  
5158 \* Specification Document: Section 10.4 Authenticator Selection  
5159 Extension (authnSel) of this specification  
5160 \* WebAuthn Extension Identifier: exts  
5161 \* Description: This registration extension enables the Relying Party  
5162 to determine which extensions the authenticator supports. The  
5163 extension data is a list (CBOR array) of extension identifiers  
5164 encoded as UTF-8 Strings. This extension is added automatically by  
5165 the authenticator. This extension can be added to attestation  
5166 statements.  
5167 \* Specification Document: Section 10.5 Supported Extensions  
5168 Extension (exts) of this specification  
5169 \* WebAuthn Extension Identifier: uvi  
5170 \* Description: This registration extension and authentication  
5171 extension enables use of a user verification index. The user  
5172 verification index is a value uniquely identifying a user  
5173 verification data record. The UVI data can be used by servers to  
5174 understand whether an authentication was authorized by the exact  
5175 same biometric data as the initial key generation. This allows the  
5176 detection and prevention of "friendly fraud".  
5177 \* Specification Document: Section 10.6 User Verification Index  
5178 Extension (uvi) of this specification  
5179 \* WebAuthn Extension Identifier: loc  
5180 \* Description: The location registration extension and authentication  
5181 extension provides the client device's current location to the  
5182 WebAuthn relying party, if supported by the client device and  
5183 subject to user consent.  
5184 \* Specification Document: Section 10.7 Location Extension (loc) of  
5185 this specification  
5186 \* WebAuthn Extension Identifier: uvm  
5187 \* Description: This registration extension and authentication  
5188 extension enables use of a user verification method. The user  
5189 verification method extension returns to the Webauthn relying party  
5190 which user verification methods (factors) were used for the  
5191 WebAuthn operation.  
5192 \* Specification Document: Section 10.8 User Verification Method  
5193 Extension (uvm) of this specification

### 11.3. COSE Algorithm Registrations

5194 This section registers identifiers for the following ECDAA algorithms  
5195 in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG]. Note that  
5196 [WebAuthn-COSE-Algs] also registers RSASSA-PKCS1-v1\_5 [RFC8017]  
5197 algorithms using SHA-2 and SHA-1 hash functions in the IANA COSE  
5198 Algorithms registry [IANA-COSE-ALGS-REG], such as registering -257 for  
5199 "RS256".  
5200 \* Name: ED256  
5201 \* Value: TBD (requested assignment -260)  
5202 \* Description: TPM\_ECC\_BN\_P256 curve w/ SHA-256  
5203 \* Reference: Section 4.2 of [FIDOEcdaaAlgorithm]  
5204 \* Recommended: Yes  
5205 \* Name: ED512  
5206 \* Value: TBD (requested assignment -261)  
5207 \* Description: ECC\_BN\_ISOP512 curve w/ SHA-512  
5208 \* Reference: Section 4.2 of [FIDOEcdaaAlgorithm]  
5209 \* Recommended: Yes

### 12. Sample scenarios

5210  
5211  
5212  
5213  
5214

5173 This section is not normative.  
 5174  
 5175  
 5176 In this section, we walk through some events in the lifecycle of a  
 5177 public key credential, along with the corresponding sample code for  
 5178 using this API. Note that this is an example flow and does not limit  
 5179 the scope of how the API can be used.  
 5180  
 5181 As was the case in earlier sections, this flow focuses on a use case  
 5182 involving an external first-factor authenticator with its own display.  
 5183 One example of such an authenticator would be a smart phone. Other  
 5184 authenticator types are also supported by this API, subject to  
 5185 implementation by the platform. For instance, this flow also works  
 5186 without modification for the case of an authenticator that is embedded  
 5187 in the client platform. The flow also works for the case of an  
 5188 authenticator without its own display (similar to a smart card) subject  
 5189 to specific implementation considerations. Specifically, the client  
 5190 platform needs to display any prompts that would otherwise be shown by  
 5191 the authenticator, and the authenticator needs to allow the client  
 5192 platform to enumerate all the authenticator's credentials so that the  
 5193 client can have information to show appropriate prompts.  
 5194  
 5195 **12.1. Registration**  
 5196  
 5197 This is the first-time flow, in which a new credential is created and  
 5198 registered with the server. In this flow, the Relying Party does not  
 5199 have a preference for platform authenticator or roaming authenticators.  
 5200 **1.** The user visits example.com, which serves up a script. At this  
 5201 point, the user may already be logged in using a legacy username  
 5202 and password, or additional authenticator, or other means  
 5203 acceptable to the Relying Party. Or the user may be in the process  
 5204 of creating a new account.  
 5205 **2.** The Relying Party script runs the code snippet below.  
 5206 **3.** The client platform searches for and locates the authenticator.  
 5207 **4.** The client platform connects to the authenticator, performing any  
 5208 pairing actions if necessary.  
 5209 **5.** The authenticator shows appropriate UI for the user to select the  
 5210 authenticator on which the new credential will be created, and  
 5211 obtains a biometric or other authorization gesture from the user.  
 5212 **6.** The authenticator returns a response to the client platform, which  
 5213 in turn returns a response to the Relying Party script. If the user  
 5214 declined to select an authenticator or provide authorization, an  
 5215 appropriate error is returned.  
 5216 **7.** If a new credential was created,  
 5217 + The Relying Party script sends the newly generated credential  
 5218 public key to the server, along with additional information  
 5219 such as attestation regarding the provenance and  
 5220 characteristics of the authenticator.  
 5221 + The server stores the credential public key in its database  
 5222 and associates it with the user as well as with the  
 5223 characteristics of authentication indicated by attestation,  
 5224 also storing a friendly name for later use.  
 5225 + The script may store data such as the credential ID in local  
 5226 storage, to improve future UX by narrowing the choice of  
 5227 credential for the user.  
 5228  
 5229 The sample code for generating and registering a new key follows:  
 5230 `if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */ }`  
 5231  
 5232 `var publicKey = {`  
 5233  `// The challenge must be produced by the server, see the Security Consideratio`  
 5234 `ns`  
 5235  `challenge: new Uint8Array([21,31,105 /* 29 more random bytes generated by the`  
 5236 `server */]),`  
 5237  
 5238  `// Relying Party:`  
 5239  `rp: {`  
 5240  `name: "ACME Corporation"`  
 5241  `},`  
 5242

5215 This section is not normative.  
 5216  
 5217  
 5218 In this section, we walk through some events in the lifecycle of a  
 5219 public key credential, along with the corresponding sample code for  
 5220 using this API. Note that this is an example flow and does not limit  
 5221 the scope of how the API can be used.  
 5222  
 5223 As was the case in earlier sections, this flow focuses on a use case  
 5224 involving an external first-factor authenticator with its own display.  
 5225 One example of such an authenticator would be a smart phone. Other  
 5226 authenticator types are also supported by this API, subject to  
 5227 implementation by the platform. For instance, this flow also works  
 5228 without modification for the case of an authenticator that is embedded  
 5229 in the client platform. The flow also works for the case of an  
 5230 authenticator without its own display (similar to a smart card) subject  
 5231 to specific implementation considerations. Specifically, the client  
 5232 platform needs to display any prompts that would otherwise be shown by  
 5233 the authenticator, and the authenticator needs to allow the client  
 5234 platform to enumerate all the authenticator's credentials so that the  
 5235 client can have information to show appropriate prompts.  
 5236  
 5237 **12.1. Registration**  
 5238  
 5239 This is the first-time flow, in which a new credential is created and  
 5240 registered with the server. In this flow, the Relying Party does not  
 5241 have a preference for platform authenticator or roaming authenticators.  
 5242 **1.** The user visits example.com, which serves up a script. At this  
 5243 point, the user may already be logged in using a legacy username  
 5244 and password, or additional authenticator, or other means  
 5245 acceptable to the Relying Party. Or the user may be in the process  
 5246 of creating a new account.  
 5247 **2.** The Relying Party script runs the code snippet below.  
 5248 **3.** The client platform searches for and locates the authenticator.  
 5249 **4.** The client platform connects to the authenticator, performing any  
 5250 pairing actions if necessary.  
 5251 **5.** The authenticator shows appropriate UI for the user to select the  
 5252 authenticator on which the new credential will be created, and  
 5253 obtains a biometric or other authorization gesture from the user.  
 5254 **6.** The authenticator returns a response to the client platform, which  
 5255 in turn returns a response to the Relying Party script. If the user  
 5256 declined to select an authenticator or provide authorization, an  
 5257 appropriate error is returned.  
 5258 **7.** If a new credential was created,  
 5259 + The Relying Party script sends the newly generated credential  
 5260 public key to the server, along with additional information  
 5261 such as attestation regarding the provenance and  
 5262 characteristics of the authenticator.  
 5263 + The server stores the credential public key in its database  
 5264 and associates it with the user as well as with the  
 5265 characteristics of authentication indicated by attestation,  
 5266 also storing a friendly name for later use.  
 5267 + The script may store data such as the credential ID in local  
 5268 storage, to improve future UX by narrowing the choice of  
 5269 credential for the user.  
 5270  
 5271 The sample code for generating and registering a new key follows:  
 5272 `if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */ }`  
 5273  
 5274 `var publicKey = {`  
 5275  `// The challenge must be produced by the server, see the Security Consideratio`  
 5276 `ns`  
 5277  `challenge: new Uint8Array([21,31,105 /* 29 more random bytes generated by the`  
 5278 `server */]),`  
 5279  
 5280  `// Relying Party:`  
 5281  `rp: {`  
 5282  `name: "ACME Corporation"`  
 5283  `},`  
 5284

```

5243 // User:
5244 user: {
5245   id: Uint8Array.from(window.atob("MIIBkzCCATigAwIBAjCCAZMwggE4oAMCAQIwggGTMII
5246   ="), c=>c.charCodeAt(0)),
5247   name: "alex.p.mueller@example.com",
5248   displayName: "Alex P. Miller",
5249   icon: "https://pics.example.com/00/p/aBjjjpqPb.png"
5250 },
5251
5252 // This Relying Party will accept either an ES256 or RS256 credential, but
5253 // prefers an ES256 credential.
5254 pubKeyCredParams: [
5255   {
5256     type: "public-key",
5257     alg: -7 // "ES256" as registered in the IANA COSE Algorithms registry
5258   },
5259   {
5260     type: "public-key",
5261     alg: -257 // Value registered by this specification for "RS256"
5262   }
5263 ],
5264
5265 timeout: 60000, // 1 minute
5266 excludeCredentials: [], // No exclude list of PKCredDescriptors
5267 extensions: {"loc": true} // Include location information
5268 // in attestation
5269 };
5270
5271 // Note: The following call will cause the authenticator to display UI.
5272 navigator.credentials.create({ publicKey })
5273 .then(function (newCredentialInfo) {
5274   // Send new credential info to server for verification and registration.
5275   }).catch(function (err) {
5276   // No acceptable authenticator or user refused consent. Handle appropriately
5277   });
5278
5279
5280 12.2. Registration Specifically with User Verifying Platform Authenticator
5281
5282 This is flow for when the Relying Party is specifically interested in
5283 creating a public key credential with a user-verifying platform
5284 authenticator.
5285 1. The user visits example.com and clicks on the login button, which
5286   redirects the user to login.example.com.
5287 2. The user enters a username and password to log in. After successful
5288   login, the user is redirected back to example.com.
5289 3. The Relying Party script runs the code snippet below.
5290 4. The user agent asks the user whether they are willing to register
5291   with the Relying Party using an available platform authenticator.
5292 5. If the user is not willing, terminate this flow.
5293 6. The user is shown appropriate UI and guided in creating a
5294   credential using one of the available platform authenticators. Upon
5295   successful credential creation, the RP script conveys the new
5296   credential to the server.
5297
5298 if (!window.PublicKeyCredential) { /* Platform not capable of the API. Handle er
5299   ror. */ }
5300
5301 PublicKeyCredential.isUserVerifyingPlatformAuthenticatorAvailable()
5302 .then(function (userIntent) {
5303
5304   // If the user has affirmed willingness to register with RP using an ava
5305   ilable platform authenticator
5306   if (userIntent) {
5307     var publicKeyOptions = { /* Public key credential creation options.
5308     */ };
5309
5310     // Create and register credentials.
5311     return navigator.credentials.create({ "publicKey": publicKeyOptions
5312

```

```

5285 // User:
5286 user: {
5287   id: Uint8Array.from(window.atob("MIIBkzCCATigAwIBAjCCAZMwggE4oAMCAQIwggGTMII
5288   ="), c=>c.charCodeAt(0)),
5289   name: "alex.p.mueller@example.com",
5290   displayName: "Alex P. Miller",
5291   icon: "https://pics.example.com/00/p/aBjjjpqPb.png"
5292 },
5293
5294 // This Relying Party will accept either an ES256 or RS256 credential, but
5295 // prefers an ES256 credential.
5296 pubKeyCredParams: [
5297   {
5298     type: "public-key",
5299     alg: -7 // "ES256" as registered in the IANA COSE Algorithms registry
5300   },
5301   {
5302     type: "public-key",
5303     alg: -257 // Value registered by this specification for "RS256"
5304   }
5305 ],
5306
5307 timeout: 60000, // 1 minute
5308 excludeCredentials: [], // No exclude list of PKCredDescriptors
5309 extensions: {"loc": true} // Include location information
5310 // in attestation
5311 };
5312
5313 // Note: The following call will cause the authenticator to display UI.
5314 navigator.credentials.create({ publicKey })
5315 .then(function (newCredentialInfo) {
5316   // Send new credential info to server for verification and registration.
5317   }).catch(function (err) {
5318   // No acceptable authenticator or user refused consent. Handle appropriately
5319   });
5320
5321
5322 12.2. Registration Specifically with User Verifying Platform Authenticator
5323
5324 This is flow for when the Relying Party is specifically interested in
5325 creating a public key credential with a user-verifying platform
5326 authenticator.
5327 1. The user visits example.com and clicks on the login button, which
5328   redirects the user to login.example.com.
5329 2. The user enters a username and password to log in. After successful
5330   login, the user is redirected back to example.com.
5331 3. The Relying Party script runs the code snippet below.
5332 4. The user agent asks the user whether they are willing to register
5333   with the Relying Party using an available platform authenticator.
5334 5. If the user is not willing, terminate this flow.
5335 6. The user is shown appropriate UI and guided in creating a
5336   credential using one of the available platform authenticators. Upon
5337   successful credential creation, the RP script conveys the new
5338   credential to the server.
5339
5340 if (!window.PublicKeyCredential) { /* Platform not capable of the API. Handle er
5341   ror. */ }
5342
5343 PublicKeyCredential.isUserVerifyingPlatformAuthenticatorAvailable()
5344 .then(function (userIntent) {
5345
5346   // If the user has affirmed willingness to register with RP using an ava
5347   ilable platform authenticator
5348   if (userIntent) {
5349     var publicKeyOptions = { /* Public key credential creation options.
5350     */ };
5351
5352     // Create and register credentials.
5353     return navigator.credentials.create({ "publicKey": publicKeyOptions
5354

```

```

5313     } else {
5314
5315         // Record that the user does not intend to use a platform authentica
5316     tor
5317         // and default the user to a password-based flow in the future.
5318     }
5319
5320     }).then(function (newCredentialInfo) {
5321         // Send new credential info to server for verification and registration.
5322     }).catch( function(err) {
5323         // Something went wrong. Handle appropriately.
5324     });
5325
5326 12.3. Authentication
5327
5328 This is the flow when a user with an already registered credential
5329 visits a website and wants to authenticate using the credential.
5330 1. The user visits example.com, which serves up a script.
5331 2. The script asks the client platform for an Authentication
5332 Assertion, providing as much information as possible to narrow the
5333 choice of acceptable credentials for the user. This can be obtained
5334 from the data that was stored locally after registration, or by
5335 other means such as prompting the user for a username.
5336 3. The Relying Party script runs one of the code snippets below.
5337 4. The client platform searches for and locates the authenticator.
5338 5. The client platform connects to the authenticator, performing any
5339 pairing actions if necessary.
5340 6. The authenticator presents the user with a notification that their
5341 attention is needed. On opening the notification, the user is shown
5342 a friendly selection menu of acceptable credentials using the
5343 account information provided when creating the credentials, along
5344 with some information on the origin that is requesting these keys.
5345 7. The authenticator obtains a biometric or other authorization
5346 gesture from the user.
5347 8. The authenticator returns a response to the client platform, which
5348 in turn returns a response to the Relying Party script. If the user
5349 declined to select a credential or provide an authorization, an
5350 appropriate error is returned.
5351 9. If an assertion was successfully generated and returned,
5352 + The script sends the assertion to the server.
5353 + The server examines the assertion, extracts the credential ID,
5354 looks up the registered credential public key it is database,
5355 and verifies the assertion's authentication signature. If
5356 valid, it looks up the identity associated with the
5357 assertion's credential ID; that identity is now authenticated.
5358 If the credential ID is not recognized by the server (e.g., it
5359 has been deregistered due to inactivity) then the
5360 authentication has failed; each Relying Party will handle this
5361 in its own way.
5362 + The server now does whatever it would otherwise do upon
5363 successful authentication -- return a success page, set
5364 authentication cookies, etc.
5365
5366 If the Relying Party script does not have any hints available (e.g.,
5367 from locally stored data) to help it narrow the list of credentials,
5368 then the sample code for performing such an authentication might look
5369 like this:
5370 if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */ }
5371
5372 var options = {
5373     // The challenge must be produced by the server, see the Security
5374     y Considerations
5375     challenge: new Uint8Array([4,101,15 /* 29 more random bytes generated
5376     rated by the server */]),
5377     timeout: 60000, // 1 minute
5378     allowCredentials: [{ type: "public-key" }]
5379 };
5380
5381 navigator.credentials.get({ "publicKey": options })
5382 .then(function (assertion) {

```

```

5355     } else {
5356
5357         // Record that the user does not intend to use a platform authentica
5358     tor
5359         // and default the user to a password-based flow in the future.
5360     }
5361
5362     }).then(function (newCredentialInfo) {
5363         // Send new credential info to server for verification and registration.
5364     }).catch( function(err) {
5365         // Something went wrong. Handle appropriately.
5366     });
5367
5368 12.3. Authentication
5369
5370 This is the flow when a user with an already registered credential
5371 visits a website and wants to authenticate using the credential.
5372 1. The user visits example.com, which serves up a script.
5373 2. The script asks the client platform for an Authentication
5374 Assertion, providing as much information as possible to narrow the
5375 choice of acceptable credentials for the user. This can be obtained
5376 from the data that was stored locally after registration, or by
5377 other means such as prompting the user for a username.
5378 3. The Relying Party script runs one of the code snippets below.
5379 4. The client platform searches for and locates the authenticator.
5380 5. The client platform connects to the authenticator, performing any
5381 pairing actions if necessary.
5382 6. The authenticator presents the user with a notification that their
5383 attention is needed. On opening the notification, the user is shown
5384 a friendly selection menu of acceptable credentials using the
5385 account information provided when creating the credentials, along
5386 with some information on the origin that is requesting these keys.
5387 7. The authenticator obtains a biometric or other authorization
5388 gesture from the user.
5389 8. The authenticator returns a response to the client platform, which
5390 in turn returns a response to the Relying Party script. If the user
5391 declined to select a credential or provide an authorization, an
5392 appropriate error is returned.
5393 9. If an assertion was successfully generated and returned,
5394 + The script sends the assertion to the server.
5395 + The server examines the assertion, extracts the credential ID,
5396 looks up the registered credential public key it is database,
5397 and verifies the assertion's authentication signature. If
5398 valid, it looks up the identity associated with the
5399 assertion's credential ID; that identity is now authenticated.
5400 If the credential ID is not recognized by the server (e.g., it
5401 has been deregistered due to inactivity) then the
5402 authentication has failed; each Relying Party will handle this
5403 in its own way.
5404 + The server now does whatever it would otherwise do upon
5405 successful authentication -- return a success page, set
5406 authentication cookies, etc.
5407
5408 If the Relying Party script does not have any hints available (e.g.,
5409 from locally stored data) to help it narrow the list of credentials,
5410 then the sample code for performing such an authentication might look
5411 like this:
5412 if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */ }
5413
5414 var options = {
5415     // The challenge must be produced by the server, see the Security
5416     y Considerations
5417     challenge: new Uint8Array([4,101,15 /* 29 more random bytes generated
5418     rated by the server */]),
5419     timeout: 60000, // 1 minute
5420     allowCredentials: [{ type: "public-key" }]
5421 };
5422
5423 navigator.credentials.get({ "publicKey": options })
5424 .then(function (assertion) {

```

```

5383 // Send assertion to server for verification
5384 }).catch(function (err) {
5385 // No acceptable credential or user refused consent. Handle appropriately.
5386 });
5387
5388 On the other hand, if the Relying Party script has some hints to help
5389 it narrow the list of credentials, then the sample code for performing
5390 such an authentication might look like the following. Note that this
5391 sample also demonstrates how to use the extension for transaction
5392 authorization.
5393 if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */}
5394
5395 var encoder = new TextEncoder();
5396 var acceptableCredential1 = {
5397   type: "public-key",
5398   id: encoder.encode("!!!!!!hi there!!!!!!\n")
5399 };
5400 var acceptableCredential2 = {
5401   type: "public-key",
5402   id: encoder.encode("roses are red, violets are blue\n")
5403 };
5404
5405 var options = {
5406   // The challenge must be produced by the server, see the Security
5407   // Considerations
5408   challenge: new Uint8Array([8,18,33 /* 29 more random bytes generated
5409   by the server */]),
5410   timeout: 60000, // 1 minute
5411   allowCredentials: [acceptableCredential1, acceptableCredential2]
5412 ,
5413   extensions: { 'txAuthSimple':
5414     "Wave your hands in the air like you just don't care" }
5415 };
5416
5417 navigator.credentials.get({ "publicKey": options })
5418   .then(function (assertion) {
5419     // Send assertion to server for verification
5420   }).catch(function (err) {
5421     // No acceptable credential or user refused consent. Handle appropriately.
5422   });
5423
5424 12.4. Aborting Authentication Operations
5425
5426 The below example shows how a developer may use the AbortSignal
5427 parameter to abort a credential registration operation. A similar
5428 procedure applies to an authentication operation.
5429 const authAbortController = new AbortController();
5430 const authAbortSignal = authAbortController.signal;
5431
5432 authAbortSignal.onabort = function () {
5433   // Once the page knows the abort started, inform user it is attempting to abort.
5434 }
5435 }
5436
5437 var options = {
5438   // A list of options.
5439 }
5440
5441 navigator.credentials.create({
5442   publicKey: options,
5443   signal: authAbortSignal})
5444   .then(function (attestation) {
5445     // Register the user.
5446   }).catch(function (error) {
5447     if (error == "AbortError") {
5448       // Inform user the credential hasn't been created.
5449       // Let the server know a key hasn't been created.
5450     }
5451   });
5452

```

```

5425 // Send assertion to server for verification
5426 }).catch(function (err) {
5427 // No acceptable credential or user refused consent. Handle appropriately.
5428 });
5429
5430 On the other hand, if the Relying Party script has some hints to help
5431 it narrow the list of credentials, then the sample code for performing
5432 such an authentication might look like the following. Note that this
5433 sample also demonstrates how to use the extension for transaction
5434 authorization.
5435 if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */}
5436
5437 var encoder = new TextEncoder();
5438 var acceptableCredential1 = {
5439   type: "public-key",
5440   id: encoder.encode("!!!!!!hi there!!!!!!\n")
5441 };
5442 var acceptableCredential2 = {
5443   type: "public-key",
5444   id: encoder.encode("roses are red, violets are blue\n")
5445 };
5446
5447 var options = {
5448   // The challenge must be produced by the server, see the Security
5449   // Considerations
5450   challenge: new Uint8Array([8,18,33 /* 29 more random bytes generated
5451   by the server */]),
5452   timeout: 60000, // 1 minute
5453   allowCredentials: [acceptableCredential1, acceptableCredential2]
5454 ,
5455   extensions: { 'txAuthSimple':
5456     "Wave your hands in the air like you just don't care" }
5457 };
5458
5459 navigator.credentials.get({ "publicKey": options })
5460   .then(function (assertion) {
5461     // Send assertion to server for verification
5462   }).catch(function (err) {
5463     // No acceptable credential or user refused consent. Handle appropriately.
5464   });
5465
5466 12.4. Aborting Authentication Operations
5467
5468 The below example shows how a developer may use the AbortSignal
5469 parameter to abort a credential registration operation. A similar
5470 procedure applies to an authentication operation.
5471 const authAbortController = new AbortController();
5472 const authAbortSignal = authAbortController.signal;
5473
5474 authAbortSignal.onabort = function () {
5475   // Once the page knows the abort started, inform user it is attempting to abort.
5476 }
5477 }
5478
5479 var options = {
5480   // A list of options.
5481 }
5482
5483 navigator.credentials.create({
5484   publicKey: options,
5485   signal: authAbortSignal})
5486   .then(function (attestation) {
5487     // Register the user.
5488   }).catch(function (error) {
5489     if (error == "AbortError") {
5490       // Inform user the credential hasn't been created.
5491       // Let the server know a key hasn't been created.
5492     }
5493   });
5494

```

```

5453 // Assume widget shows up whenever authentication occurs.
5454 if (widget == "disappear") {
5455   authAbortController.abort();
5456 }

```

### 12.5. Decommissioning

The following are possible situations in which decommissioning a credential might be desired. Note that all of these are handled on the server side and do not need support from the API specified here.

- \* Possibility #1 -- user reports the credential as lost.
  - + User goes to server.example.net, authenticates and follows a link to report a lost/stolen device.
  - + Server returns a page showing the list of registered credentials with friendly names as configured during registration.
  - + User selects a credential and the server deletes it from its database.
  - + In future, the Relying Party script does not specify this credential in any list of acceptable credentials, and assertions signed by this credential are rejected.
- \* Possibility #2 -- server deregisters the credential due to inactivity.
  - + Server deletes credential from its database during maintenance activity.
  - + In the future, the Relying Party script does not specify this credential in any list of acceptable credentials, and assertions signed by this credential are rejected.
- \* Possibility #3 -- user deletes the credential from the device.
  - + User employs a device-specific method (e.g., device settings UI) to delete a credential from their device.
  - + From this point on, this credential will not appear in any selection prompts, and no assertions can be generated with it.
  - + Sometime later, the server deregisters this credential due to inactivity.

### 13. Security Considerations

This specification defines a Web API and a cryptographic peer-entity authentication protocol. The Web Authentication API allows Web developers (i.e., "authors") to utilize the Web Authentication protocol in their registration and authentication ceremonies. The entities comprising the Web Authentication protocol endpoints are user-controlled authenticators and a Relying Party's computing environment hosting the Relying Party's web application. In this model, the user agent, together with the WebAuthn Client, comprise an intermediary between authenticators and Relying Parties. Additionally, authenticators can attest to Relying Parties as to their provenance.

At this time, this specification does not feature detailed security considerations. However, the [FIDOSecRef] document provides a security analysis which is overall applicable to this specification. Also, the [FIDOAuthnrSecReqs] document suite defines authenticator security characteristics which are overall applicable for WebAuthn authenticators.

The below subsections comprise the current Web Authentication-specific security considerations.

#### 13.1. Cryptographic Challenges

As a cryptographic protocol, Web Authentication is dependent upon randomized challenges to avoid replay attacks. Therefore, both challenge's and challenge's value MUST be randomly generated by Relying Parties in an environment they trust (e.g., on the server-side), and the returned challenge value in the client's response MUST match what was generated. This SHOULD be done in a fashion that does not rely upon a client's behavior, e.g., the Relying Party SHOULD store the challenge temporarily until the operation is complete. Tolerating a mismatch will compromise the security of the protocol.

```

5495 // Assume widget shows up whenever authentication occurs.
5496 if (widget == "disappear") {
5497   authAbortController.abort();
5498 }

```

### 12.5. Decommissioning

The following are possible situations in which decommissioning a credential might be desired. Note that all of these are handled on the server side and do not need support from the API specified here.

- \* Possibility #1 -- user reports the credential as lost.
  - + User goes to server.example.net, authenticates and follows a link to report a lost/stolen device.
  - + Server returns a page showing the list of registered credentials with friendly names as configured during registration.
  - + User selects a credential and the server deletes it from its database.
  - + In future, the Relying Party script does not specify this credential in any list of acceptable credentials, and assertions signed by this credential are rejected.
- \* Possibility #2 -- server deregisters the credential due to inactivity.
  - + Server deletes credential from its database during maintenance activity.
  - + In the future, the Relying Party script does not specify this credential in any list of acceptable credentials, and assertions signed by this credential are rejected.
- \* Possibility #3 -- user deletes the credential from the device.
  - + User employs a device-specific method (e.g., device settings UI) to delete a credential from their device.
  - + From this point on, this credential will not appear in any selection prompts, and no assertions can be generated with it.
  - + Sometime later, the server deregisters this credential due to inactivity.

### 13. Security Considerations

This specification defines a Web API and a cryptographic peer-entity authentication protocol. The Web Authentication API allows Web developers (i.e., "authors") to utilize the Web Authentication protocol in their registration and authentication ceremonies. The entities comprising the Web Authentication protocol endpoints are user-controlled authenticators and a Relying Party's computing environment hosting the Relying Party's web application. In this model, the user agent, together with the WebAuthn Client, comprise an intermediary between authenticators and Relying Parties. Additionally, authenticators can attest to Relying Parties as to their provenance.

At this time, this specification does not feature detailed security considerations. However, the [FIDOSecRef] document provides a security analysis which is overall applicable to this specification. Also, the [FIDOAuthnrSecReqs] document suite defines authenticator security characteristics which are overall applicable for WebAuthn authenticators.

The below subsections comprise the current Web Authentication-specific security considerations.

#### 13.1. Cryptographic Challenges

As a cryptographic protocol, Web Authentication is dependent upon randomized challenges to avoid replay attacks. Therefore, both challenge's and challenge's value MUST be randomly generated by Relying Parties in an environment they trust (e.g., on the server-side), and the returned challenge value in the client's response MUST match what was generated. This SHOULD be done in a fashion that does not rely upon a client's behavior, e.g., the Relying Party SHOULD store the challenge temporarily until the operation is complete. Tolerating a mismatch will compromise the security of the protocol.



5523 In order to prevent replay attacks, the challenges MUST contain enough  
5524 entropy to make guessing them infeasible. Challenges SHOULD therefore  
5525 be at least 16 bytes long.

### 5526 13.2. Attestation Security Considerations

#### 5527 13.2.1. Attestation Certificate Hierarchy

5528 A 3-tier hierarchy for attestation certificates is RECOMMENDED (i.e.,  
5529 Attestation Root, Attestation Issuing CA, Attestation Certificate). It  
5530 is also RECOMMENDED that for each WebAuthn Authenticator device line  
5531 (i.e., model), a separate issuing CA is used to help facilitate  
5532 isolating problems with a specific version of a device.

5533 If the attestation root certificate is not dedicated to a single  
5534 WebAuthn Authenticator device line (i.e., AAGUID), the AAGUID SHOULD be  
5535 specified in the attestation certificate itself, so that it can be  
5536 verified against the authenticator data.

#### 5537 13.2.2. Attestation Certificate and Attestation Certificate CA Compromise

5538 When an intermediate CA or a root CA used for issuing attestation  
5539 certificates is compromised, WebAuthn authenticator attestation keys  
5540 are still safe although their certificates can no longer be trusted. A  
5541 WebAuthn Authenticator manufacturer that has recorded the public  
5542 attestation keys for their devices can issue new attestation  
5543 certificates for these keys from a new intermediate CA or from a new  
5544 root CA. If the root CA changes, the Relying Parties MUST update their  
5545 trusted root certificates accordingly.

5546 A WebAuthn Authenticator attestation certificate MUST be revoked by the  
5547 issuing CA if its key has been compromised. A WebAuthn Authenticator  
5548 manufacturer may need to ship a firmware update and inject new  
5549 attestation keys and certificates into already manufactured WebAuthn  
5550 Authenticators, if the exposure was due to a firmware flaw. (The  
5551 process by which this happens is out of scope for this specification.)  
5552 If the WebAuthn Authenticator manufacturer does not have this  
5553 capability, then it may not be possible for Relying Parties to trust  
5554 any further attestation statements from the affected WebAuthn  
5555 Authenticators.

5556 If attestation certificate validation fails due to a revoked  
5557 intermediate attestation CA certificate, and the Relying Party's policy  
5558 requires rejecting the registration/authentication request in these  
5559 situations, then it is RECOMMENDED that the Relying Party also  
5560 un-registers (or marks with a trust level equivalent to "self  
5561 attestation") public key credentials that were registered after the CA  
5562 compromise date using an attestation certificate chaining up to the  
5563 same intermediate CA. It is thus RECOMMENDED that Relying Parties  
5564 remember intermediate attestation CA certificates during Authenticator  
5565 registration in order to un-register related public key credentials if  
5566 the registration was performed after revocation of such certificates.

5567 If an ECDAAtest key has been compromised, it can be added to  
5568 the RogueList (i.e., the list of revoked authenticators) maintained by  
5569 the related ECDAAtest-Issuer. The Relying Party SHOULD verify whether an  
5570 authenticator belongs to the RogueList when performing ECDAAtest-Verify  
5571 (see section 3.6 in [FIDOEcdaaAlgorithm]). For example, the FIDO  
5572 Metadata Service [FIDOMetadataService] provides one way to access such  
5573 information.

### 5574 13.3. Security Benefits for Relying Parties

5575 The main benefits offered to Relying Parties by this specification  
5576 include:

- 5577 1. Users and accounts can be secured using widely compatible,  
5578 easy-to-use multi-factor authentication.
- 5579 2. The Relying Party does not need to provision authenticator hardware  
5580 to its users. Instead, each user can independently obtain any

5565 In order to prevent replay attacks, the challenges MUST contain enough  
5566 entropy to make guessing them infeasible. Challenges SHOULD therefore  
5567 be at least 16 bytes long.

### 5568 13.2. Attestation Security Considerations

#### 5569 13.2.1. Attestation Certificate Hierarchy

5570 A 3-tier hierarchy for attestation certificates is RECOMMENDED (i.e.,  
5571 Attestation Root, Attestation Issuing CA, Attestation Certificate). It  
5572 is also RECOMMENDED that for each WebAuthn Authenticator device line  
5573 (i.e., model), a separate issuing CA is used to help facilitate  
5574 isolating problems with a specific version of a device.

5575 If the attestation root certificate is not dedicated to a single  
5576 WebAuthn Authenticator device line (i.e., AAGUID), the AAGUID SHOULD be  
5577 specified in the attestation certificate itself, so that it can be  
5578 verified against the authenticator data.

#### 5579 13.2.2. Attestation Certificate and Attestation Certificate CA Compromise

5580 When an intermediate CA or a root CA used for issuing attestation  
5581 certificates is compromised, WebAuthn authenticator attestation keys  
5582 are still safe although their certificates can no longer be trusted. A  
5583 WebAuthn Authenticator manufacturer that has recorded the public  
5584 attestation keys for their devices can issue new attestation  
5585 certificates for these keys from a new intermediate CA or from a new  
5586 root CA. If the root CA changes, the Relying Parties MUST update their  
5587 trusted root certificates accordingly.

5588 A WebAuthn Authenticator attestation certificate MUST be revoked by the  
5589 issuing CA if its key has been compromised. A WebAuthn Authenticator  
5590 manufacturer may need to ship a firmware update and inject new  
5591 attestation keys and certificates into already manufactured WebAuthn  
5592 Authenticators, if the exposure was due to a firmware flaw. (The  
5593 process by which this happens is out of scope for this specification.)  
5594 If the WebAuthn Authenticator manufacturer does not have this  
5595 capability, then it may not be possible for Relying Parties to trust  
5596 any further attestation statements from the affected WebAuthn  
5597 Authenticators.

5598 If attestation certificate validation fails due to a revoked  
5599 intermediate attestation CA certificate, and the Relying Party's policy  
5600 requires rejecting the registration/authentication request in these  
5601 situations, then it is RECOMMENDED that the Relying Party also  
5602 un-registers (or marks with a trust level equivalent to "self  
5603 attestation") public key credentials that were registered after the CA  
5604 compromise date using an attestation certificate chaining up to the  
5605 same intermediate CA. It is thus RECOMMENDED that Relying Parties  
5606 remember intermediate attestation CA certificates during Authenticator  
5607 registration in order to un-register related public key credentials if  
5608 the registration was performed after revocation of such certificates.

5609 If an ECDAAtest key has been compromised, it can be added to  
5610 the RogueList (i.e., the list of revoked authenticators) maintained by  
5611 the related ECDAAtest-Issuer. The Relying Party SHOULD verify whether an  
5612 authenticator belongs to the RogueList when performing ECDAAtest-Verify  
5613 (see section 3.6 in [FIDOEcdaaAlgorithm]). For example, the FIDO  
5614 Metadata Service [FIDOMetadataService] provides one way to access such  
5615 information.

### 5616 13.3. Security Benefits for Relying Parties

5617 The main benefits offered to Relying Parties by this specification  
5618 include:

- 5619 1. Users and accounts can be secured using widely compatible,  
5620 easy-to-use multi-factor authentication.
- 5621 2. The Relying Party does not need to provision authenticator hardware  
5622 to its users. Instead, each user can independently obtain any

5593 conforming authenticator and use that same authenticator with any  
5594 number of Relying Parties. The Relying Party can optionally enforce  
5595 requirements on authenticators' security properties by inspecting  
5596 the attestation statements returned from the authenticators.  
5597 3. Registration and authentication ceremonies are resistant to  
5598 man-in-the-middle attacks.  
5599 4. The Relying Party can automatically support multiple types of user  
5600 verification - for example PIN, biometrics and/or future methods -  
5601 with little or no code change, and can let each user decide which  
5602 they prefer to use via their choice of authenticator.  
5603 5. The Relying Party does not need to store additional secrets in  
5604 order to gain the above benefits.

5605  
5606 As stated in the Conformance section, the Relying Party MUST behave as  
5607 described in 7 Relying Party Operations to obtain all of the above  
5608 security benefits. However, one notable use case that departs slightly  
5609 from this is described in the next section.

5610 13.3.1. Considerations for Self and None Attestation Types and Ignoring  
5611 Attestation

5612  
5613 When registering a new credential, the Relying Party MAY choose to  
5614 accept an attestation statement of type Self or None, or to not verify  
5615 the attestation statement. In all of these cases the Relying Party  
5616 loses much of benefit (3) listed above, but retains the other benefits.

5617  
5618 In these cases it is possible for a man-in-the-middle attacker - for  
5619 example, a malicious client or script - to replace the credential  
5620 public key to be registered, and subsequently tamper with future  
5621 authentication assertions bound for the same Relying Party and passing  
5622 through the same attacker. Accepting these types of attestation  
5623 statements therefore constitutes a leap of faith. In cases where  
5624 registration was accomplished securely, subsequent authentication  
5625 ceremonies remain resistant to man-in-the-middle attacks, i.e., benefit  
5626 (3) is retained. Note, however, that such an attack would be easy to  
5627 detect and very difficult to maintain, since any authentication  
5628 ceremony that the same attacker does not or cannot tamper with would  
5629 always fail.

5630  
5631 The Relying Party SHOULD consider the above in its threat model when  
5632 deciding its policy on what attestation types to accept or whether to  
5633 ignore attestation.

5634  
5635 Note: The default attestation type is None, since the above issues will  
5636 likely not be a major concern in most Relying Parties' threat models.  
5637 For example, the man-in-the-middle attack described above is more  
5638 difficult than a man-in-the-middle attack against a Relying Party that  
5639 only uses conventional password authentication.

5640 13.4. credentialId Unsigned

5641  
5642 The credential ID is not signed. This is not a problem because all that  
5643 would happen if an authenticator returns the wrong credential ID, or if  
5644 an attacker intercepts and manipulates the credential ID, is that the  
5645 Relying Party would not look up the correct credential public key with  
5646 which to verify the returned signed authenticator data (a.k.a.,  
5647 assertion), and thus the interaction would end in an error.

5648 13.5. Browser Permissions Framework and Extensions

5649  
5650 Web Authentication API implementations should leverage the browser  
5651 permissions framework as much as possible when obtaining user  
5652 permissions for certain extensions. An example is the location  
5653 extension (see 10.7 Location Extension (loc)), implementations of  
5654 which should make use of the existing browser permissions framework for  
5655 the Geolocation API.

5656 14. Privacy Considerations

5657  
5658 The privacy principles in [FIDO-Privacy-Principles] also apply to this

5659  
5660  
5661  
5662

5635 conforming authenticator and use that same authenticator with any  
5636 number of Relying Parties. The Relying Party can optionally enforce  
5637 requirements on authenticators' security properties by inspecting  
5638 the attestation statements returned from the authenticators.  
5639 3. Registration and authentication ceremonies are resistant to  
5640 man-in-the-middle attacks.  
5641 4. The Relying Party can automatically support multiple types of user  
5642 verification - for example PIN, biometrics and/or future methods -  
5643 with little or no code change, and can let each user decide which  
5644 they prefer to use via their choice of authenticator.  
5645 5. The Relying Party does not need to store additional secrets in  
5646 order to gain the above benefits.

5647  
5648 As stated in the Conformance section, the Relying Party MUST behave as  
5649 described in 7 Relying Party Operations to obtain all of the above  
5650 security benefits. However, one notable use case that departs slightly  
5651 from this is described in the next section.

5652 13.3.1. Considerations for Self and None Attestation Types and Ignoring  
5653 Attestation

5654  
5655 When registering a new credential, the Relying Party MAY choose to  
5656 accept an attestation statement of type Self or None, or to not verify  
5657 the attestation statement. In all of these cases the Relying Party  
5658 loses much of benefit (3) listed above, but retains the other benefits.

5659  
5660 In these cases it is possible for a man-in-the-middle attacker - for  
5661 example, a malicious client or script - to replace the credential  
5662 public key to be registered, and subsequently tamper with future  
5663 authentication assertions bound for the same Relying Party and passing  
5664 through the same attacker. Accepting these types of attestation  
5665 statements therefore constitutes a leap of faith. In cases where  
5666 registration was accomplished securely, subsequent authentication  
5667 ceremonies remain resistant to man-in-the-middle attacks, i.e., benefit  
5668 (3) is retained. Note, however, that such an attack would be easy to  
5669 detect and very difficult to maintain, since any authentication  
5670 ceremony that the same attacker does not or cannot tamper with would  
5671 always fail.

5672  
5673 The Relying Party SHOULD consider the above in its threat model when  
5674 deciding its policy on what attestation types to accept or whether to  
5675 ignore attestation.

5676  
5677 Note: The default attestation type is None, since the above issues will  
5678 likely not be a major concern in most Relying Parties' threat models.  
5679 For example, the man-in-the-middle attack described above is more  
5680 difficult than a man-in-the-middle attack against a Relying Party that  
5681 only uses conventional password authentication.

5682 13.4. credentialId Unsigned

5683  
5684 The credential ID is not signed. This is not a problem because all that  
5685 would happen if an authenticator returns the wrong credential ID, or if  
5686 an attacker intercepts and manipulates the credential ID, is that the  
5687 Relying Party would not look up the correct credential public key with  
5688 which to verify the returned signed authenticator data (a.k.a.,  
5689 assertion), and thus the interaction would end in an error.

5690 13.5. Browser Permissions Framework and Extensions

5691  
5692 Web Authentication API implementations should leverage the browser  
5693 permissions framework as much as possible when obtaining user  
5694 permissions for certain extensions. An example is the location  
5695 extension (see 10.7 Location Extension (loc)), implementations of  
5696 which should make use of the existing browser permissions framework for  
5697 the Geolocation API.

5700 14. Privacy Considerations

5701  
5702 The privacy principles in [FIDO-Privacy-Principles] also apply to this

5703  
5704

5663  
5664  
5665  
5666  
5667  
5668  
5669  
5670  
5671  
5672  
5673  
5674  
5675  
5676  
5677  
5678  
5679  
5680  
5681  
5682  
5683  
5684  
5685  
5686  
5687  
5688  
5689  
5690  
5691  
5692  
5693  
5694  
5695  
5696  
5697  
5698  
5699  
5700  
5701  
5702  
5703  
5704  
5705  
5706  
5707  
5708  
5709  
5710  
5711  
5712  
5713  
5714  
5715  
5716  
5717  
5718  
5719  
5720  
5721  
5722  
5723  
5724  
5725  
5726  
5727  
5728  
5729  
5730  
5731  
5732

specification.

#### 14.1. Attestation Privacy

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

- \* A WebAuthn authenticator manufacturer may choose to ship all of their devices with the same (or a fixed number of) attestation key(s) (called Basic Attestation). This will anonymize the user at the risk of not being able to revoke a particular attestation key if its private key is compromised. [UAFProtocol] requires that at least 100,000 devices share the same attestation certificate in order to produce sufficiently large groups. This may serve as guidance about suitable batch sizes.
- \* A WebAuthn authenticator may be capable of dynamically generating different attestation keys (and requesting related certificates) per-origin (similar to the Attestation CA approach). For example, an authenticator can ship with a master attestation key (and certificate), and combined with a cloud-operated Anonymization CA, can dynamically generate per-origin attestation keys and attestation certificates. Note: In various places outside this specification, the term "Privacy CA" is used to refer to what is termed here as an Anonymization CA. Because the Trusted Computing Group (TCG) also used the term "Privacy CA" to refer to what the TCG now refers to as an Attestation CA (ACA) [TCG-CMCPProfile-AIKCertEnroll], and the envisioned functionality of an Anonymization CA is not firmly established, we are using the term Anonymization CA here to try to mitigate confusion in the specific context of this specification.
- \* A WebAuthn Authenticator can implement Elliptic Curve based direct anonymous attestation (see [FIDOEcdaaAlgorithm]). Using this scheme, the authenticator generates a blinded attestation signature. This allows the Relying Party to verify the signature using the ECDAAs-Issuer public key, but the attestation signature does not serve as a global correlation handle.

#### 14.2. Registration Ceremony Privacy

In order to protect users from being identified without consent, implementations of the [[Create]](origin, options, sameOriginWithAncestors) method need to take care to not leak information that could enable a malicious Relying Party to distinguish between these cases, where "excluded" means that at least one of the credentials listed by the Relying Party in excludeCredentials is bound to the authenticator:

- \* No authenticators are present.
- \* At least one authenticator is present, and at least one present authenticator is excluded.

If the above cases are distinguishable, information is leaked by which a malicious Relying Party could identify the user by probing for which credentials are available. For example, one such information leak is if the client returns a failure response as soon as an excluded authenticator becomes available. In this case - especially if the excluded authenticator is a platform authenticator - the Relying Party could detect that the ceremony was canceled before the timeout and before the user could feasibly have canceled it manually, and thus conclude that at least one of the credentials listed in the excludeCredentials parameter is available to the user.

The above is not a concern, however, if the user has consented to create a new credential before a distinguishable error is returned, because in this case the user has confirmed intent to share the information that would be leaked.

#### 14.3. Authentication Ceremony Privacy

In order to protect users from being identified without consent, implementations of the [[DiscoverFromExternalSource]](origin, options,

5705  
5706  
5707  
5708  
5709  
5710  
5711  
5712  
5713  
5714  
5715  
5716  
5717  
5718  
5719  
5720  
5721  
5722  
5723  
5724  
5725  
5726  
5727  
5728  
5729  
5730  
5731  
5732  
5733  
5734  
5735  
5736  
5737  
5738  
5739  
5740  
5741  
5742  
5743  
5744  
5745  
5746  
5747  
5748  
5749  
5750  
5751  
5752  
5753  
5754  
5755  
5756  
5757  
5758  
5759  
5760  
5761  
5762  
5763  
5764  
5765  
5766  
5767  
5768  
5769  
5770  
5771  
5772  
5773  
5774

specification.

#### 14.1. Attestation Privacy

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

- \* A WebAuthn authenticator manufacturer may choose to ship all of their devices with the same (or a fixed number of) attestation key(s) (called Basic Attestation). This will anonymize the user at the risk of not being able to revoke a particular attestation key if its private key is compromised. [UAFProtocol] requires that at least 100,000 devices share the same attestation certificate in order to produce sufficiently large groups. This may serve as guidance about suitable batch sizes.
- \* A WebAuthn authenticator may be capable of dynamically generating different attestation keys (and requesting related certificates) per-origin (similar to the Attestation CA approach). For example, an authenticator can ship with a master attestation key (and certificate), and combined with a cloud-operated Anonymization CA, can dynamically generate per-origin attestation keys and attestation certificates. Note: In various places outside this specification, the term "Privacy CA" is used to refer to what is termed here as an Anonymization CA. Because the Trusted Computing Group (TCG) also used the term "Privacy CA" to refer to what the TCG now refers to as an Attestation CA (ACA) [TCG-CMCPProfile-AIKCertEnroll], and the envisioned functionality of an Anonymization CA is not firmly established, we are using the term Anonymization CA here to try to mitigate confusion in the specific context of this specification.
- \* A WebAuthn Authenticator can implement Elliptic Curve based direct anonymous attestation (see [FIDOEcdaaAlgorithm]). Using this scheme, the authenticator generates a blinded attestation signature. This allows the Relying Party to verify the signature using the ECDAAs-Issuer public key, but the attestation signature does not serve as a global correlation handle.

#### 14.2. Registration Ceremony Privacy

In order to protect users from being identified without consent, implementations of the [[Create]](origin, options, sameOriginWithAncestors) method need to take care to not leak information that could enable a malicious Relying Party to distinguish between these cases, where "excluded" means that at least one of the credentials listed by the Relying Party in excludeCredentials is bound to the authenticator:

- \* No authenticators are present.
- \* At least one authenticator is present, and at least one present authenticator is excluded.

If the above cases are distinguishable, information is leaked by which a malicious Relying Party could identify the user by probing for which credentials are available. For example, one such information leak is if the client returns a failure response as soon as an excluded authenticator becomes available. In this case - especially if the excluded authenticator is a platform authenticator - the Relying Party could detect that the ceremony was canceled before the timeout and before the user could feasibly have canceled it manually, and thus conclude that at least one of the credentials listed in the excludeCredentials parameter is available to the user.

The above is not a concern, however, if the user has consented to create a new credential before a distinguishable error is returned, because in this case the user has confirmed intent to share the information that would be leaked.

#### 14.3. Authentication Ceremony Privacy

In order to protect users from being identified without consent, implementations of the [[DiscoverFromExternalSource]](origin, options,

5733 sameOriginWithAncestors) method need to take care to not leak  
5734 information that could enable a malicious Relying Party to distinguish  
5735 between these cases, where "named" means that the credential is listed  
5736 by the Relying Party in allowCredentials:  
5737 \* A named credential is not available.  
5738 \* A named credential is available, but the user does not consent to  
5739 use it.

5741 If the above cases are distinguishable, information is leaked by which  
5742 a malicious Relying Party could identify the user by probing for which  
5743 credentials are available. For example, one such information leak is if  
5744 the client returns a failure response as soon as the user denies  
5745 consent to proceed with an authentication ceremony. In this case the  
5746 Relying Party could detect that the ceremony was canceled by the user  
5747 and not the timeout, and thus conclude that at least one of the  
5748 credentials listed in the allowCredentials parameter is available to  
5749 the user.

5751 15. Acknowledgements

5752 We thank the following people for their reviews of, and contributions  
5753 to, this specification: Yuriy Ackermann, James Barclay, Richard Barnes,  
5754 Dominic Battr, John Bradley, Domenic Denicola, Rahul Ghosh, Brad Hill,  
5755 Jing Jin, Wally Jones, Ian Kilpatrick, Axel Nennker, Yoshikazu Nojima,  
5756 Kimberly Paulhamus, Adam Powers, Yaron Sheffer, Anne van Kesteren,  
5757 Johan Verrept, and Boris Zbarsky.

5760 We thank Anthony Nadalin, John Fontana, and Richard Barnes for their  
5761 contributions as co-chairs of the Web Authentication Working Group.

5762 We also thank Wendy Seltzer, Samuel Weiler, and Harry Halpin for their  
5763 contributions as our W3C Team Contacts.

5765 Index

5766 Terms defined by this specification

- 5770 \* aaguid, in 6.3.1
- 5771 \* AAGUID, in 10.4
- 5772 \* alg, in 5.3
- 5773 \* allowCredentials, in 5.5
- 5774 \* Anonymization CA, in 14.1
- 5775 \* appid
  - 5776 + dict-member for AuthenticationExtensionsClientInputs, in 10.1
  - 5777 + dict-member for AuthenticationExtensionsClientOutputs, in
  - 5778 10.1
- 5779 \* AppID, in 10.1
- 5780 \* Assertion, in 4
- 5781 \* assertion signature, in 6
- 5782 \* attachment modality, in 5.4.5
- 5783 \* AttCA, in 6.3.3
- 5784 \* Attestation, in 4
- 5785 \* attestation, in 5.4
- 5786 \* Attestation CA, in 6.3.3
- 5787 \* Attestation Certificate, in 4
- 5788 \* Attestation Conveyance, in 5.4.6
- 5789 \* AttestationConveyancePreference, in 5.4.6
- 5790 \* attestationConveyancePreferenceOption, in 5.1.3
- 5791 \* attestation key pair, in 4
- 5792 \* attestationObject, in 5.2.1
- 5793 \* attestation object, in 6.3
- 5794 \* attestationObjectResult, in 5.1.3
- 5795 \* attestation private key, in 4
- 5796 \* attestation public key, in 4
- 5797 \* attestation signature, in 6
- 5798 \* attestation statement, in 6.3
- 5799 \* attestation statement format, in 6.3
- 5800 \* attestation statement format identifier, in 8.1
- 5801 \* attestation trust path, in 6.3.2
- 5802 \* attestation type, in 6.3

5775 sameOriginWithAncestors) method need to take care to not leak  
5776 information that could enable a malicious Relying Party to distinguish  
5777 between these cases, where "named" means that the credential is listed  
5778 by the Relying Party in allowCredentials:  
5779 \* A named credential is not available.  
5780 \* A named credential is available, but the user does not consent to  
5781 use it.

5782 If the above cases are distinguishable, information is leaked by which  
5783 a malicious Relying Party could identify the user by probing for which  
5784 credentials are available. For example, one such information leak is if  
5785 the client returns a failure response as soon as the user denies  
5786 consent to proceed with an authentication ceremony. In this case the  
5787 Relying Party could detect that the ceremony was canceled by the user  
5788 and not the timeout, and thus conclude that at least one of the  
5789 credentials listed in the allowCredentials parameter is available to  
5790 the user.

5792 15. Acknowledgements

5793 We thank the following people for their reviews of, and contributions  
5794 to, this specification: Yuriy Ackermann, James Barclay, Richard Barnes,  
5795 Dominic Battr, John Bradley, Domenic Denicola, Rahul Ghosh, Brad Hill,  
5796 Jing Jin, Wally Jones, Ian Kilpatrick, Axel Nennker, Yoshikazu Nojima,  
5797 Kimberly Paulhamus, Adam Powers, Yaron Sheffer, Anne van Kesteren,  
5798 Johan Verrept, and Boris Zbarsky.

5800 We thank Anthony Nadalin, John Fontana, and Richard Barnes for their  
5801 contributions as co-chairs of the Web Authentication Working Group.

5802 We also thank Wendy Seltzer, Samuel Weiler, and Harry Halpin for their  
5803 contributions as our W3C Team Contacts.

5805 Index

5806 Terms defined by this specification

- 5810 \* aaguid, in 6.3.1
- 5811 \* AAGUID, in 10.4
- 5812 \* alg, in 5.3
- 5813 \* allowCredentials, in 5.5
- 5814 \* Anonymization CA, in 14.1
- 5815 \* appid
  - 5816 + dict-member for AuthenticationExtensionsClientInputs, in 10.1
  - 5817 + dict-member for AuthenticationExtensionsClientOutputs, in
  - 5818 10.1
- 5819 \* AppID, in 10.1
- 5820 \* Assertion, in 4
- 5821 \* assertion signature, in 6
- 5822 \* attachment modality, in 5.4.5
- 5823 \* AttCA, in 6.3.3
- 5824 \* Attestation, in 4
- 5825 \* attestation, in 5.4
- 5826 \* Attestation CA, in 6.3.3
- 5827 \* Attestation Certificate, in 4
- 5828 \* Attestation Conveyance, in 5.4.6
- 5829 \* AttestationConveyancePreference, in 5.4.6
- 5830 \* attestationConveyancePreferenceOption, in 5.1.3
- 5831 \* attestation key pair, in 4
- 5832 \* attestationObject, in 5.2.1
- 5833 \* attestation object, in 6.3
- 5834 \* attestationObjectResult, in 5.1.3
- 5835 \* attestation private key, in 4
- 5836 \* attestation public key, in 4
- 5837 \* attestation signature, in 6
- 5838 \* attestation statement, in 6.3
- 5839 \* attestation statement format, in 6.3
- 5840 \* attestation statement format identifier, in 8.1
- 5841 \* attestation trust path, in 6.3.2
- 5842 \* attestation type, in 6.3

5803 \* Attested credential data, in 6.3.1  
 5804 \* attestedCredentialData, in 6.1  
 5805 \* authDataExtensions, in 6.1  
 5806 \* Authentication, in 4  
 5807 \* Authentication Assertion, in 4  
 5808 \* authentication extension, in 9  
 5809 \* AuthenticationExtensionsAuthenticatorInputs  
 5810 + (typedef), in 5.9  
 5811 + definition of, in 5.9  
 5812 \* AuthenticationExtensionsClientInputs  
 5813 + (dictionary), in 5.7  
 5814 + definition of, in 5.7  
 5815 \* AuthenticationExtensionsClientOutputs  
 5816 + (dictionary), in 5.8  
 5817 + definition of, in 5.8  
 5818 \* AuthenticationExtensionsSupported, in 10.5  
 5819 \* Authenticator, in 4  
 5820 \* AuthenticatorAssertionResponse, in 5.2.2  
 5821 \* AuthenticatorAttachment, in 5.4.5  
 5822 \* authenticatorAttachment, in 5.4.4  
 5823 \* AuthenticatorAttestationResponse, in 5.2.1  
 5824 \* authenticatorBiometricPerfBounds, in 10.9  
 5825 \* authenticatorCancel, in 6.2.4  
 5826 \* authenticator data, in 6.1  
 5827 \* authenticatorData, in 5.2.2  
 5828 \* authenticator data claimed to have been used for the attestation,  
 5829 in 6.3.2  
 5830 \* authenticator data for the attestation, in 6.3.2  
 5831 \* authenticatorDataResult, in 5.1.4.1  
 5832 \* authenticator extension, in 9  
 5833 \* authenticator extension input, in 9.3  
 5834 \* authenticator extension output, in 9.5  
 5835 \* Authenticator extension processing, in 9.5  
 5836 \* authenticatorGetAssertion, in 6.2.3  
 5837 \* authenticatorMakeCredential, in 6.2.2  
 5838 \* Authenticator Model, in 6  
 5839 \* Authenticator operations, in 6.2  
 5840 \* AuthenticatorResponse, in 5.2  
 5841 \* authenticatorSelection, in 5.4  
 5842 \* AuthenticatorSelectionCriteria, in 5.4.4  
 5843 \* AuthenticatorSelectionList, in 10.4  
 5844 \* authenticator session, in 6.2  
 5845 \* AuthenticatorTransport, in 5.10.4  
 5846 \* authnSel  
 5847 + dict-member for AuthenticationExtensionsClientInputs, in 10.4  
 5848 + dict-member for AuthenticationExtensionsClientOutputs, in  
 5849 10.4  
 5850 \* Authorization Gesture, in 4  
 5851 \* Base64url Encoding, in 3  
 5852 \* Basic, in 6.3.3  
 5853 \* Basic Attestation, in 6.3.3  
 5854 \* Biometric Authenticator, in 4  
 5855 \* Biometric Recognition, in 4  
 5856 \* ble, in 5.10.4  
 5857 \* CBOR, in 3  
 5858 \* Ceremony, in 4  
 5859 \* challenge  
 5860 + dict-member for CollectedClientData, in 5.10.1  
 5861 + dict-member for PublicKeyCredentialCreationOptions, in 5.4  
 5862 + dict-member for PublicKeyCredentialRequestOptions, in 5.5  
 5863 \* Client, in 4  
 5864 \* client data, in 5.10.1  
 5865 \* clientDataJSON, in 5.2  
 5866 \* clientDataJSONResult  
 5867 + dfn for assertionCreationData, in 5.1.4.1  
 5868 + dfn for credentialCreationData, in 5.1.3  
 5869 \* client extension, in 9  
 5870 \* client extension input, in 9.3  
 5871 \* client extension output, in 9.4  
 5872 \* Client extension processing, in 9.4

5845 \* Attested credential data, in 6.3.1  
 5846 \* attestedCredentialData, in 6.1  
 5847 \* authDataExtensions, in 6.1  
 5848 \* Authentication, in 4  
 5849 \* Authentication Assertion, in 4  
 5850 \* authentication extension, in 9  
 5851 \* AuthenticationExtensionsAuthenticatorInputs  
 5852 + (typedef), in 5.9  
 5853 + definition of, in 5.9  
 5854 \* AuthenticationExtensionsClientInputs  
 5855 + (dictionary), in 5.7  
 5856 + definition of, in 5.7  
 5857 \* AuthenticationExtensionsClientOutputs  
 5858 + (dictionary), in 5.8  
 5859 + definition of, in 5.8  
 5860 \* AuthenticationExtensionsSupported, in 10.5  
 5861 \* Authenticator, in 4  
 5862 \* AuthenticatorAssertionResponse, in 5.2.2  
 5863 \* AuthenticatorAttachment, in 5.4.5  
 5864 \* authenticatorAttachment, in 5.4.4  
 5865 \* AuthenticatorAttestationResponse, in 5.2.1  
 5866 \* authenticatorBiometricPerfBounds, in 10.9  
 5867 \* authenticatorCancel, in 6.2.4  
 5868 \* authenticator data, in 6.1  
 5869 \* authenticatorData, in 5.2.2  
 5870 \* authenticator data claimed to have been used for the attestation,  
 5871 in 6.3.2  
 5872 \* authenticator data for the attestation, in 6.3.2  
 5873 \* authenticatorDataResult, in 5.1.4.1  
 5874 \* authenticator extension, in 9  
 5875 \* authenticator extension input, in 9.3  
 5876 \* authenticator extension output, in 9.5  
 5877 \* Authenticator extension processing, in 9.5  
 5878 \* authenticatorGetAssertion, in 6.2.3  
 5879 \* authenticatorMakeCredential, in 6.2.2  
 5880 \* Authenticator Model, in 6  
 5881 \* Authenticator operations, in 6.2  
 5882 \* AuthenticatorResponse, in 5.2  
 5883 \* authenticatorSelection, in 5.4  
 5884 \* AuthenticatorSelectionCriteria, in 5.4.4  
 5885 \* AuthenticatorSelectionList, in 10.4  
 5886 \* authenticator session, in 6.2  
 5887 \* AuthenticatorTransport, in 5.10.4  
 5888 \* authnSel  
 5889 + dict-member for AuthenticationExtensionsClientInputs, in 10.4  
 5890 + dict-member for AuthenticationExtensionsClientOutputs, in  
 5891 10.4  
 5892 \* Authorization Gesture, in 4  
 5893 \* Base64url Encoding, in 3  
 5894 \* Basic, in 6.3.3  
 5895 \* Basic Attestation, in 6.3.3  
 5896 \* Biometric Authenticator, in 4  
 5897 \* Biometric Recognition, in 4  
 5898 \* ble, in 5.10.4  
 5899 \* CBOR, in 3  
 5900 \* Ceremony, in 4  
 5901 \* challenge  
 5902 + dict-member for CollectedClientData, in 5.10.1  
 5903 + dict-member for PublicKeyCredentialCreationOptions, in 5.4  
 5904 + dict-member for PublicKeyCredentialRequestOptions, in 5.5  
 5905 \* Client, in 4  
 5906 \* client data, in 5.10.1  
 5907 \* clientDataJSON, in 5.2  
 5908 \* clientDataJSONResult  
 5909 + dfn for assertionCreationData, in 5.1.4.1  
 5910 + dfn for credentialCreationData, in 5.1.3  
 5911 \* client extension, in 9  
 5912 \* client extension input, in 9.3  
 5913 \* client extension output, in 9.4  
 5914 \* Client extension processing, in 9.4

5873 \* clientExtensionResults  
 5874 + dfn for assertionCreationData, in 5.1.4.1  
 5875 + dfn for credentialCreationData, in 5.1.3  
 5876 \* [[clientExtensionsResults]], in 5.1  
 5877 \* Client-Side, in 4  
 5878 \* client-side credential private key storage, in 4  
 5879 \* Client-side-resident Credential Private Key, in 4  
 5880 \* CollectedClientData, in 5.10.1  
 5881 \* [[CollectFromCredentialStore]](origin, options,  
 5882 sameOriginWithAncestors), in 5.1.4  
 5883 \* Conforming User Agent, in 4  
 5884 \* content, in 10.3  
 5885 \* contentType, in 10.3  
 5886 \* COSEAlgorithmIdentifier  
 5887 + (typedef), in 5.10.5  
 5888 + definition of, in 5.10.5  
 5889 \* [[Create]](origin, options, sameOriginWithAncestors), in 5.1.3  
 5890 \* Credential ID, in 4  
 5891 \* credentialId, in 6.3.1  
 5892 \* credentialIdLength, in 6.3.1  
 5893 \* credentialIdResult, in 5.1.4.1  
 5894 \* credential key pair, in 4  
 5895 \* credential private key, in 4  
 5896 \* Credential Public Key, in 4  
 5897 \* credentialPublicKey, in 6.3.1  
 5898 \* credentials map, in 6  
 5899 \* "cross-platform", in 5.4.5  
 5900 \* cross-platform attached, in 5.4.5  
 5901 \* cross-platform attachment, in 5.4.5  
 5902 \* DAA, in 6.3.3  
 5903 \* direct, in 5.4.6  
 5904 \* "discouraged", in 5.10.6  
 5905 \* [[DiscoverFromExternalSource]](origin, options,  
 5906 sameOriginWithAncestors), in 5.1.4.1  
 5907 \* [[discovery]], in 5.1  
 5908 \* displayName, in 5.4.3  
 5909 \* ECDSA, in 6.3.3  
 5910 \* ECDSA-Issuer public key, in 8.2  
 5911 \* effective user verification requirement for assertion, in 5.1.4.1  
 5912 \* effective user verification requirement for credential creation, in  
 5913 5.1.3  
 5914 \* Elliptic Curve based Direct Anonymous Attestation, in 6.3.3  
 5915 \* excludeCredentials, in 5.4  
 5916 \* extension identifier, in 9.1  
 5917 \* extensions  
 5918 + dict-member for PublicKeyCredentialCreationOptions, in 5.4  
 5919 + dict-member for PublicKeyCredentialRequestOptions, in 5.5  
 5920 \* exts  
 5921 + dict-member for AuthenticationExtensionsClientInputs, in 10.5  
 5922 + dict-member for AuthenticationExtensionsClientOutputs, in  
 5923 10.5  
 5924 \* FAR, in 10.9  
 5925 \* flags, in 6.1  
 5926 \* FRR, in 10.9  
 5927 \* getClientExtensionResults(), in 5.1  
 5928 \* Hash of the serialized client data, in 5.10.1  
 5929 \* Human Palatability, in 4  
 5930 \* icon, in 5.4.1  
 5931 \* id  
 5932 + dfn for public key credential source, in 4  
 5933 + dict-member for PublicKeyCredentialDescriptor, in 5.10.3  
 5934 + dict-member for PublicKeyCredentialRpEntity, in 5.4.2  
 5935 + dict-member for PublicKeyCredentialUserEntity, in 5.4.3  
 5936 + dict-member for TokenBinding, in 5.10.1  
 5937 \* [[identifier]], in 5.1  
 5938 \* identifier of the ECDSA-Issuer public key, in 8.2  
 5939 \* indirect, in 5.4.6  
 5940 \* **internal, in 5.10.4**  
 5941 \* isUserVerifyingPlatformAuthenticatorAvailable(), in 5.1.7  
 5942 \* JSON-serialized client data, in 5.10.1

5915 \* clientExtensionResults  
 5916 + dfn for assertionCreationData, in 5.1.4.1  
 5917 + dfn for credentialCreationData, in 5.1.3  
 5918 \* [[clientExtensionsResults]], in 5.1  
 5919 \* Client-Side, in 4  
 5920 \* client-side credential private key storage, in 4  
 5921 \* Client-side-resident Credential Private Key, in 4  
 5922 \* CollectedClientData, in 5.10.1  
 5923 \* [[CollectFromCredentialStore]](origin, options,  
 5924 sameOriginWithAncestors), in 5.1.4  
 5925 \* Conforming User Agent, in 4  
 5926 \* content, in 10.3  
 5927 \* contentType, in 10.3  
 5928 \* COSEAlgorithmIdentifier  
 5929 + (typedef), in 5.10.5  
 5930 + definition of, in 5.10.5  
 5931 \* [[Create]](origin, options, sameOriginWithAncestors), in 5.1.3  
 5932 \* Credential ID, in 4  
 5933 \* credentialId, in 6.3.1  
 5934 \* credentialIdLength, in 6.3.1  
 5935 \* credentialIdResult, in 5.1.4.1  
 5936 \* credential key pair, in 4  
 5937 \* credential private key, in 4  
 5938 \* Credential Public Key, in 4  
 5939 \* credentialPublicKey, in 6.3.1  
 5940 \* credentials map, in 6  
 5941 \* "cross-platform", in 5.4.5  
 5942 \* cross-platform attached, in 5.4.5  
 5943 \* cross-platform attachment, in 5.4.5  
 5944 \* DAA, in 6.3.3  
 5945 \* direct, in 5.4.6  
 5946 \* "discouraged", in 5.10.6  
 5947 \* [[DiscoverFromExternalSource]](origin, options,  
 5948 sameOriginWithAncestors), in 5.1.4.1  
 5949 \* [[discovery]], in 5.1  
 5950 \* displayName, in 5.4.3  
 5951 \* ECDSA, in 6.3.3  
 5952 \* ECDSA-Issuer public key, in 8.2  
 5953 \* effective user verification requirement for assertion, in 5.1.4.1  
 5954 \* effective user verification requirement for credential creation, in  
 5955 5.1.3  
 5956 \* Elliptic Curve based Direct Anonymous Attestation, in 6.3.3  
 5957 \* excludeCredentials, in 5.4  
 5958 \* extension identifier, in 9.1  
 5959 \* extensions  
 5960 + dict-member for PublicKeyCredentialCreationOptions, in 5.4  
 5961 + dict-member for PublicKeyCredentialRequestOptions, in 5.5  
 5962 \* exts  
 5963 + dict-member for AuthenticationExtensionsClientInputs, in 10.5  
 5964 + dict-member for AuthenticationExtensionsClientOutputs, in  
 5965 10.5  
 5966 \* FAR, in 10.9  
 5967 \* flags, in 6.1  
 5968 \* FRR, in 10.9  
 5969 \* getClientExtensionResults(), in 5.1  
 5970 \* Hash of the serialized client data, in 5.10.1  
 5971 \* Human Palatability, in 4  
 5972 \* icon, in 5.4.1  
 5973 \* id  
 5974 + dfn for public key credential source, in 4  
 5975 + dict-member for PublicKeyCredentialDescriptor, in 5.10.3  
 5976 + dict-member for PublicKeyCredentialRpEntity, in 5.4.2  
 5977 + dict-member for PublicKeyCredentialUserEntity, in 5.4.3  
 5978 + dict-member for TokenBinding, in 5.10.1  
 5979 \* [[identifier]], in 5.1  
 5980 \* identifier of the ECDSA-Issuer public key, in 8.2  
 5981 \* indirect, in 5.4.6  
 5982 \* isUserVerifyingPlatformAuthenticatorAvailable(), in 5.1.7  
 5983 \* JSON-serialized client data, in 5.10.1

5943 \* loc  
 5944 + dict-member for AuthenticationExtensionsClientInputs, in 10.7  
 5945 + dict-member for AuthenticationExtensionsClientOutputs, in  
 5946 10.7  
 5947 \* looking up, in 6.2.1  
 5948 \* managing authenticator, in 4  
 5949 \* name, in 5.4.1  
 5950 \* nfc, in 5.10.4  
 5951 \* No attestation statement, in 6.3.3  
 5952 \* None, in 6.3.3  
 5953 \* none, in 5.4.6  
 5954 \* none attestation statement format, in 8.7  
 5955 \* "not-supported", in 5.10.1  
 5956 \* origin, in 5.10.1  
 5957 \* otherUI, in 4  
 5958 \* "platform", in 5.4.5  
 5959 \* platform attachment, in 5.4.5  
 5960 \* platform authenticators, in 5.4.5  
 5961 \* platform credential, in 5.4.5  
 5962 \* "preferred", in 5.10.6  
 5963 \* "present", in 5.10.1  
 5964 \* [[preventSilentAccess]](credential, sameOriginWithAncestors), in  
 5965 5.1.6  
 5966 \* privateKey, in 4  
 5967 \* pubKeyCredParams, in 5.4  
 5968 \* publicKey  
 5969 + dict-member for CredentialCreationOptions, in 5.1.1  
 5970 + dict-member for CredentialRequestOptions, in 5.1.2  
 5971 \* public-key, in 5.10.2  
 5972 \* Public Key Credential, in 4  
 5973 \* PublicKeyCredential, in 5.1  
 5974 \* PublicKeyCredentialCreationOptions, in 5.4  
 5975 \* PublicKeyCredentialDescriptor, in 5.10.3  
 5976 \* PublicKeyCredentialEntity, in 5.4.1  
 5977 \* PublicKeyCredentialParameters, in 5.3  
 5978 \* PublicKeyCredentialRequestOptions, in 5.5  
 5979 \* PublicKeyCredentialRpEntity, in 5.4.2  
 5980 \* Public Key Credential Source, in 4  
 5981 \* PublicKeyCredentialType, in 5.10.2  
 5982 \* PublicKeyCredentialUserEntity, in 5.4.3  
 5983 \* Rate Limiting, in 4  
 5984 \* rawId, in 5.1  
 5985 \* Registration, in 4  
 5986 \* registration extension, in 9  
 5987 \* Relying Party, in 4  
 5988 \* Relying Party Identifier, in 4  
 5989 \* "required", in 5.10.6  
 5990 \* requireResidentKey, in 5.4.4  
 5991 \* response, in 5.1  
 5992 \* roaming authenticators, in 5.4.5  
 5993 \* roaming credential, in 5.4.5  
 5994 \* rp, in 5.4  
 5995 \* rpId  
 5996 + dfn for public key credential source, in 4  
 5997 + dict-member for PublicKeyCredentialRequestOptions, in 5.5  
 5998 \* RP ID, in 4  
 5999 \* rpIdHash, in 6.1  
 6000 \* Self, in 6.3.3  
 6001 \* Self Attestation, in 6.3.3  
 6002 \* signature, in 5.2.2  
 6003 \* Signature Counter, in 6.1.1  
 6004 \* signatureResult, in 5.1.4.1  
 6005 \* signCount, in 6.1  
 6006 \* Signing procedure, in 6.3.2  
 6007 \* status, in 5.10.1  
 6008 \* [[Store]](credential, sameOriginWithAncestors), in 5.1.5  
 6009 \* "supported", in 5.10.1  
 6010 \* Test of User Presence, in 4  
 6011 \* timeout  
 6012 + dict-member for PublicKeyCredentialCreationOptions, in 5.4

5984 \* loc  
 5985 + dict-member for AuthenticationExtensionsClientInputs, in 10.7  
 5986 + dict-member for AuthenticationExtensionsClientOutputs, in  
 5987 10.7  
 5988 \* looking up, in 6.2.1  
 5989 \* managing authenticator, in 4  
 5990 \* name, in 5.4.1  
 5991 \* nfc, in 5.10.4  
 5992 \* No attestation statement, in 6.3.3  
 5993 \* None, in 6.3.3  
 5994 \* none, in 5.4.6  
 5995 \* none attestation statement format, in 8.7  
 5996 \* "not-supported", in 5.10.1  
 5997 \* origin, in 5.10.1  
 5998 \* otherUI, in 4  
 5999 \* "platform", in 5.4.5  
 6000 \* platform attachment, in 5.4.5  
 6001 \* platform authenticators, in 5.4.5  
 6002 \* platform credential, in 5.4.5  
 6003 \* "preferred", in 5.10.6  
 6004 \* "present", in 5.10.1  
 6005 \* [[preventSilentAccess]](credential, sameOriginWithAncestors), in  
 6006 5.1.6  
 6007 \* privateKey, in 4  
 6008 \* pubKeyCredParams, in 5.4  
 6009 \* publicKey  
 6010 + dict-member for CredentialCreationOptions, in 5.1.1  
 6011 + dict-member for CredentialRequestOptions, in 5.1.2  
 6012 \* public-key, in 5.10.2  
 6013 \* Public Key Credential, in 4  
 6014 \* PublicKeyCredential, in 5.1  
 6015 \* PublicKeyCredentialCreationOptions, in 5.4  
 6016 \* PublicKeyCredentialDescriptor, in 5.10.3  
 6017 \* PublicKeyCredentialEntity, in 5.4.1  
 6018 \* PublicKeyCredentialParameters, in 5.3  
 6019 \* PublicKeyCredentialRequestOptions, in 5.5  
 6020 \* PublicKeyCredentialRpEntity, in 5.4.2  
 6021 \* Public Key Credential Source, in 4  
 6022 \* PublicKeyCredentialType, in 5.10.2  
 6023 \* PublicKeyCredentialUserEntity, in 5.4.3  
 6024 \* Rate Limiting, in 4  
 6025 \* rawId, in 5.1  
 6026 \* Registration, in 4  
 6027 \* registration extension, in 9  
 6028 \* Relying Party, in 4  
 6029 \* Relying Party Identifier, in 4  
 6030 \* "required", in 5.10.6  
 6031 \* requireResidentKey, in 5.4.4  
 6032 \* response, in 5.1  
 6033 \* roaming authenticators, in 5.4.5  
 6034 \* roaming credential, in 5.4.5  
 6035 \* rp, in 5.4  
 6036 \* rpId  
 6037 + dfn for public key credential source, in 4  
 6038 + dict-member for PublicKeyCredentialRequestOptions, in 5.5  
 6039 \* RP ID, in 4  
 6040 \* rpIdHash, in 6.1  
 6041 \* Self, in 6.3.3  
 6042 \* Self Attestation, in 6.3.3  
 6043 \* signature, in 5.2.2  
 6044 \* Signature Counter, in 6.1.1  
 6045 \* signatureResult, in 5.1.4.1  
 6046 \* signCount, in 6.1  
 6047 \* Signing procedure, in 6.3.2  
 6048 \* status, in 5.10.1  
 6049 \* [[Store]](credential, sameOriginWithAncestors), in 5.1.5  
 6050 \* "supported", in 5.10.1  
 6051 \* Test of User Presence, in 4  
 6052 \* timeout  
 6053 + dict-member for PublicKeyCredentialCreationOptions, in 5.4

6013 + dict-member for PublicKeyCredentialRequestOptions, in 5.5  
6014 \* tokenBinding, in 5.10.1  
6015 \* TokenBinding, in 5.10.1  
6016 \* TokenBindingStatus, in 5.10.1  
6017 \* transports, in 5.10.3  
6018 \* txAuthGeneric  
6019 + dict-member for AuthenticationExtensionsClientInputs, in 10.3  
6020 + dict-member for AuthenticationExtensionsClientOutputs, in 10.3  
6021 \* txAuthGenericArg, in 10.3  
6022 \* txAuthSimple  
6023 + dict-member for AuthenticationExtensionsClientInputs, in 10.2  
6024 + dict-member for AuthenticationExtensionsClientOutputs, in 10.2  
6025 \* [[type]], in 5.1  
6026 \* type  
6027 + dfn for public key credential source, in 4  
6028 + dict-member for CollectedClientData, in 5.10.1  
6029 + dict-member for PublicKeyCredentialDescriptor, in 5.10.3  
6030 + dict-member for PublicKeyCredentialParameters, in 5.3  
6031 \* UP, in 4  
6032 \* usb, in 5.10.4  
6033 \* user, in 5.4  
6034 \* User Consent, in 4  
6035 \* userHandle  
6036 + attribute for AuthenticatorAssertionResponse, in 5.2.2  
6037 + dfn for public key credential source, in 4  
6038 \* User Handle, in 4  
6039 \* userHandleResult, in 5.1.4.1  
6040 \* User Present, in 4  
6041 \* User Public Key, in 4  
6042 \* userVerification  
6043 + dict-member for AuthenticatorSelectionCriteria, in 5.4.4  
6044 + dict-member for PublicKeyCredentialRequestOptions, in 5.5  
6045 \* User Verification, in 4  
6046 \* UserVerificationRequirement, in 5.10.6  
6047 \* User Verified, in 4  
6048 \* UV, in 4  
6049 \* uvi  
6050 + dict-member for AuthenticationExtensionsClientInputs, in 10.6  
6051 + dict-member for AuthenticationExtensionsClientOutputs, in 10.6  
6052 \* uvm  
6053 + dict-member for AuthenticationExtensionsClientInputs, in 10.8  
6054 + dict-member for AuthenticationExtensionsClientOutputs, in 10.8  
6055 \* UvmEntries, in 10.8  
6056 \* UvmEntry, in 10.8  
6057 \* Verification procedure, in 6.3.2  
6058 \* verification procedure inputs, in 6.3.2  
6059 \* Web Authentication API, in 5  
6060 \* WebAuthn Client, in 4  
6061  
6062 <https://w3c.github.io/webappsec-credential-management/#credentialReferenced> in:  
6063 \* 3. Dependencies  
6064 \* 5.1. PublicKeyCredential Interface (2) (3) (4) (5) (6)  
6065  
6066 <https://w3c.github.io/webappsec-credential-management/#dictdef-credentialCreationOptionsReferenced> in:  
6067 \* 5.1.1. CredentialCreationOptions Dictionary Extension (2)  
6068 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6069 [[Create]](origin, options, sameOriginWithAncestors) method  
6070  
6071 <https://w3c.github.io/webappsec-credential-management/#dictdef-credentialRequestOptionsReferenced> in:  
6072 \* 5.1.2. CredentialRequestOptions Dictionary Extension (2)  
6073 \* 5.1.4.1. PublicKeyCredential's  
6074 [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method

6054 + dict-member for PublicKeyCredentialRequestOptions, in 5.5  
6055 \* tokenBinding, in 5.10.1  
6056 \* TokenBinding, in 5.10.1  
6057 \* TokenBindingStatus, in 5.10.1  
6058 \* transports, in 5.10.3  
6059 \* txAuthGeneric  
6060 + dict-member for AuthenticationExtensionsClientInputs, in 10.3  
6061 + dict-member for AuthenticationExtensionsClientOutputs, in 10.3  
6062 \* txAuthGenericArg, in 10.3  
6063 \* txAuthSimple  
6064 + dict-member for AuthenticationExtensionsClientInputs, in 10.2  
6065 + dict-member for AuthenticationExtensionsClientOutputs, in 10.2  
6066 \* [[type]], in 5.1  
6067 \* type  
6068 + dfn for public key credential source, in 4  
6069 + dict-member for CollectedClientData, in 5.10.1  
6070 + dict-member for PublicKeyCredentialDescriptor, in 5.10.3  
6071 + dict-member for PublicKeyCredentialParameters, in 5.3  
6072 \* UP, in 4  
6073 \* usb, in 5.10.4  
6074 \* user, in 5.4  
6075 \* User Consent, in 4  
6076 \* userHandle  
6077 + attribute for AuthenticatorAssertionResponse, in 5.2.2  
6078 + dfn for public key credential source, in 4  
6079 \* User Handle, in 4  
6080 \* userHandleResult, in 5.1.4.1  
6081 \* User Present, in 4  
6082 \* User Public Key, in 4  
6083 \* userVerification  
6084 + dict-member for AuthenticatorSelectionCriteria, in 5.4.4  
6085 + dict-member for PublicKeyCredentialRequestOptions, in 5.5  
6086 \* User Verification, in 4  
6087 \* UserVerificationRequirement, in 5.10.6  
6088 \* User Verified, in 4  
6089 \* UV, in 4  
6090 \* uvi  
6091 + dict-member for AuthenticationExtensionsClientInputs, in 10.6  
6092 + dict-member for AuthenticationExtensionsClientOutputs, in 10.6  
6093 \* uvm  
6094 + dict-member for AuthenticationExtensionsClientInputs, in 10.8  
6095 + dict-member for AuthenticationExtensionsClientOutputs, in 10.8  
6096 \* UvmEntries, in 10.8  
6097 \* UvmEntry, in 10.8  
6098 \* Verification procedure, in 6.3.2  
6099 \* verification procedure inputs, in 6.3.2  
6100 \* Web Authentication API, in 5  
6101 \* WebAuthn Client, in 4  
6102  
6103 <https://w3c.github.io/webappsec-credential-management/#credentialReferenced> in:  
6104 \* 3. Dependencies  
6105 \* 5.1. PublicKeyCredential Interface (2) (3) (4) (5) (6)  
6106  
6107 <https://w3c.github.io/webappsec-credential-management/#dictdef-credentialCreationOptionsReferenced> in:  
6108 \* 5.1.1. CredentialCreationOptions Dictionary Extension (2)  
6109 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6110 [[Create]](origin, options, sameOriginWithAncestors) method  
6111  
6112 <https://w3c.github.io/webappsec-credential-management/#dictdef-credentialRequestOptionsReferenced> in:  
6113 \* 5.1.2. CredentialRequestOptions Dictionary Extension (2)  
6114 \* 5.1.4.1. PublicKeyCredential's  
6115 [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method



6083 https://w3c.github.io/webappsec-credential-management/#credentialscont  
6084 inerReferenced in:  
6085 \* 5.1.4.1. PublicKeyCredential's  
6086 [[DiscoverFromExternalSource]](origin, options,  
6087 sameOriginWithAncestors) method  
6088 \* 5.4. Options for Credential Creation (dictionary  
6089 PublicKeyCredentialCreationOptions)  
6090 \* 5.5. Options for Assertion Generation (dictionary  
6091 PublicKeyCredentialRequestOptions)  
6092  
6093 https://w3c.github.io/webappsec-credential-management/#abstract-opdef-r  
6094 equest-a-credentialReferenced in:  
6095 \* 5.1.4.1. PublicKeyCredential's  
6096 [[DiscoverFromExternalSource]](origin, options,  
6097 sameOriginWithAncestors) method  
6098  
6099 https://w3c.github.io/webappsec-credential-management/#collectfromcrede  
6100 ntialstore-origin-options-sameoriginwithancestorsReferenced in:  
6101 \* 5.1. PublicKeyCredential Interface  
6102 \* 5.1.4. Use an existing credential to make an assertion -  
6103 PublicKeyCredential's [[Get]](options) method  
6104  
6105 https://w3c.github.io/webappsec-credential-management/#create-origin-op  
6106 tions-sameoriginwithancestorsReferenced in:  
6107 \* 5.6. Abort operations with AbortSignal  
6108  
6109 https://w3c.github.io/webappsec-credential-management/#store-credential  
6110 -sameoriginwithancestorsReferenced in:  
6111 \* 5.1. PublicKeyCredential Interface  
6112  
6113 https://w3c.github.io/webappsec-credential-management/#dom-credential-d  
6114 iscovery-slotReferenced in:  
6115 \* 5.1. PublicKeyCredential Interface  
6116  
6117 https://w3c.github.io/webappsec-credential-management/#dom-credential-t  
6118 ype-slotReferenced in:  
6119 \* 5.1. PublicKeyCredential Interface  
6120  
6121 https://w3c.github.io/webappsec-credential-management/#dom-credentialsc  
6122 ontainer-createReferenced in:  
6123 \* 1. Introduction  
6124 \* 5.1. PublicKeyCredential Interface (2)  
6125 \* 5.1.1. CredentialCreationOptions Dictionary Extension  
6126 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6127 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6128 (4) (5)  
6129 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
6130 \* 5.4. Options for Credential Creation (dictionary  
6131 PublicKeyCredentialCreationOptions)  
6132 \* 5.4.4. Authenticator Selection Criteria (dictionary  
6133 AuthenticatorSelectionCriteria)  
6134 \* 5.10.3. Credential Descriptor (dictionary  
6135 PublicKeyCredentialDescriptor)  
6136 \* 7. Relying Party Operations  
6137 \* 7.1. Registering a new credential (2) (3)  
6138 \* 9. WebAuthn Extensions (2) (3)  
6139 \* 9.2. Defining extensions  
6140 \* 9.3. Extending request parameters (2)  
6141 \* 10.1. FIDO AppID Extension (appid)  
6142 \* 10.4. Authenticator Selection Extension (authnSel)  
6143 \* 10.9. Biometric Authenticator Performance Bounds Extension  
6144 (biometricPerfBounds)  
6145  
6146 https://w3c.github.io/webappsec-credential-management/#concept-credenti  
6147 alReferenced in:  
6148 \* 4. Terminology  
6149 \* 5.1.4. Use an existing credential to make an assertion -  
6150 PublicKeyCredential's [[Get]](options) method (2) (3)  
6151  
6152

6124 https://w3c.github.io/webappsec-credential-management/#credentialscont  
6125 inerReferenced in:  
6126 \* 5.1.4.1. PublicKeyCredential's  
6127 [[DiscoverFromExternalSource]](origin, options,  
6128 sameOriginWithAncestors) method  
6129 \* 5.4. Options for Credential Creation (dictionary  
6130 PublicKeyCredentialCreationOptions)  
6131 \* 5.5. Options for Assertion Generation (dictionary  
6132 PublicKeyCredentialRequestOptions)  
6133  
6134 https://w3c.github.io/webappsec-credential-management/#abstract-opdef-r  
6135 equest-a-credentialReferenced in:  
6136 \* 5.1.4.1. PublicKeyCredential's  
6137 [[DiscoverFromExternalSource]](origin, options,  
6138 sameOriginWithAncestors) method  
6139  
6140 https://w3c.github.io/webappsec-credential-management/#collectfromcrede  
6141 ntialstore-origin-options-sameoriginwithancestorsReferenced in:  
6142 \* 5.1. PublicKeyCredential Interface  
6143 \* 5.1.4. Use an existing credential to make an assertion -  
6144 PublicKeyCredential's [[Get]](options) method  
6145  
6146 https://w3c.github.io/webappsec-credential-management/#create-origin-op  
6147 tions-sameoriginwithancestorsReferenced in:  
6148 \* 5.6. Abort operations with AbortSignal  
6149  
6150 https://w3c.github.io/webappsec-credential-management/#store-credential  
6151 -sameoriginwithancestorsReferenced in:  
6152 \* 5.1. PublicKeyCredential Interface  
6153  
6154 https://w3c.github.io/webappsec-credential-management/#dom-credential-d  
6155 iscovery-slotReferenced in:  
6156 \* 5.1. PublicKeyCredential Interface  
6157  
6158 https://w3c.github.io/webappsec-credential-management/#dom-credential-t  
6159 ype-slotReferenced in:  
6160 \* 5.1. PublicKeyCredential Interface  
6161  
6162 https://w3c.github.io/webappsec-credential-management/#dom-credentialsc  
6163 ontainer-createReferenced in:  
6164 \* 1. Introduction  
6165 \* 5.1. PublicKeyCredential Interface (2)  
6166 \* 5.1.1. CredentialCreationOptions Dictionary Extension  
6167 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6168 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6169 (4) (5)  
6170 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
6171 \* 5.4. Options for Credential Creation (dictionary  
6172 PublicKeyCredentialCreationOptions)  
6173 \* 5.4.4. Authenticator Selection Criteria (dictionary  
6174 AuthenticatorSelectionCriteria)  
6175 \* 5.10.3. Credential Descriptor (dictionary  
6176 PublicKeyCredentialDescriptor)  
6177 \* 7. Relying Party Operations  
6178 \* 7.1. Registering a new credential (2) (3)  
6179 \* 9. WebAuthn Extensions (2) (3)  
6180 \* 9.2. Defining extensions  
6181 \* 9.3. Extending request parameters (2)  
6182 \* 10.1. FIDO AppID Extension (appid)  
6183 \* 10.4. Authenticator Selection Extension (authnSel)  
6184 \* 10.9. Biometric Authenticator Performance Bounds Extension  
6185 (biometricPerfBounds)  
6186  
6187 https://w3c.github.io/webappsec-credential-management/#concept-credenti  
6188 alReferenced in:  
6189 \* 4. Terminology  
6190 \* 5.1.4. Use an existing credential to make an assertion -  
6191 PublicKeyCredential's [[Get]](options) method (2) (3)  
6192  
6193

6153 <https://w3c.github.io/webappsec-credential-management/#credential-source>  
6154 eReferenced in:  
6155 \* 4. Terminology  
6156 \* 5.1.4. Use an existing credential to make an assertion -  
6157 PublicKeyCredential's `[[Get]](options)` method (2) (3) (4) (5)  
6158  
6159 <https://w3c.github.io/webappsec-credential-management/#dom-credential-scoped-credential-interface>  
6160 eReferenced in:  
6161 \* 1. Introduction  
6162 \* 4. Terminology  
6163 \* 5.1. PublicKeyCredential Interface (2)  
6164 \* 5.1.2. CredentialRequestOptions Dictionary Extension  
6165 \* 5.1.4. Use an existing credential to make an assertion -  
6166 PublicKeyCredential's `[[Get]](options)` method (2)  
6167 \* 5.1.4.1. PublicKeyCredential's  
6168 `[[DiscoverFromExternalSource]](origin, options,  
6169 sameOriginWithAncestors)` method (2) (3)  
6170 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
6171 \* 5.5. Options for Assertion Generation (dictionary  
6172 PublicKeyCredentialRequestOptions) (2)  
6173 \* 5.10.3. Credential Descriptor (dictionary  
6174 PublicKeyCredentialDescriptor)  
6175 \* 7. Relying Party Operations  
6176 \* 7.2. Verifying an authentication assertion (2)  
6177 \* 9. WebAuthn Extensions (2) (3)  
6178 \* 9.2. Defining extensions  
6179 \* 9.3. Extending request parameters (2)  
6180  
6181 <https://w3c.github.io/webappsec-credential-management/#dom-credential-interface>  
6182 eReferenced in:  
6183 \* 5.1. PublicKeyCredential Interface  
6184 \* 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6) (7)  
6185  
6186 <https://w3c.github.io/webappsec-credential-management/#dom-credential-discovery-remote>  
6187 eReferenced in:  
6188 \* 5.1. PublicKeyCredential Interface  
6189  
6190 <https://w3c.github.io/webappsec-credential-management/#same-origin-with-its-ancestors>  
6191 eReferenced in:  
6192 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6193 `[[Create]](origin, options, sameOriginWithAncestors)` method (2)  
6194 \* 5.1.4.1. PublicKeyCredential's  
6195 `[[DiscoverFromExternalSource]](origin, options,  
6196 sameOriginWithAncestors)` method (2)  
6197 \* 5.1.5. Store an existing credential - PublicKeyCredential's  
6198 `[[Store]](credential, sameOriginWithAncestors)` method  
6199  
6200 <https://w3c.github.io/webappsec-credential-management/#dom-credential-request-options>  
6201 eReferenced in:  
6202 \* 5.1.4.1. PublicKeyCredential's  
6203 `[[DiscoverFromExternalSource]](origin, options,  
6204 sameOriginWithAncestors)` method (2)  
6205  
6206 <https://w3c.github.io/webappsec-credential-management/#dom-credential-scoped-credential-interface-store>  
6207 eReferenced in:  
6208 \* 5.1.5. Store an existing credential - PublicKeyCredential's  
6209 `[[Store]](credential, sameOriginWithAncestors)` method  
6210  
6211 <https://w3c.github.io/webappsec-credential-management/#dom-credential-type>  
6212 eReferenced in:  
6213 \* 5.1. PublicKeyCredential Interface  
6214  
6215 <https://w3c.github.io/webappsec-credential-management/#user-mediated-credential-interface>  
6216 eReferenced in:  
6217 \* 5.1.4. Use an existing credential to make an assertion -  
6218 PublicKeyCredential's `[[Get]](options)` method  
6219  
6220 <https://dom.spec.whatwg.org/#abortcontroller>  
6221 eReferenced in:  
6222 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6223 `[[Create]](origin, options, sameOriginWithAncestors)` method

6194 <https://w3c.github.io/webappsec-credential-management/#credential-source>  
6195 eReferenced in:  
6196 \* 4. Terminology  
6197 \* 5.1.4. Use an existing credential to make an assertion -  
6198 PublicKeyCredential's `[[Get]](options)` method (2) (3) (4) (5)  
6199  
6200 <https://w3c.github.io/webappsec-credential-management/#dom-credential-scoped-credential-interface>  
6201 eReferenced in:  
6202 \* 1. Introduction  
6203 \* 4. Terminology  
6204 \* 5.1. PublicKeyCredential Interface (2)  
6205 \* 5.1.2. CredentialRequestOptions Dictionary Extension  
6206 \* 5.1.4. Use an existing credential to make an assertion -  
6207 PublicKeyCredential's `[[Get]](options)` method (2)  
6208 \* 5.1.4.1. PublicKeyCredential's  
6209 `[[DiscoverFromExternalSource]](origin, options,  
6210 sameOriginWithAncestors)` method (2) (3)  
6211 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
6212 \* 5.5. Options for Assertion Generation (dictionary  
6213 PublicKeyCredentialRequestOptions) (2)  
6214 \* 5.10.3. Credential Descriptor (dictionary  
6215 PublicKeyCredentialDescriptor)  
6216 \* 7. Relying Party Operations  
6217 \* 7.2. Verifying an authentication assertion (2)  
6218 \* 9. WebAuthn Extensions (2) (3)  
6219 \* 9.2. Defining extensions  
6220 \* 9.3. Extending request parameters (2)  
6221  
6222 <https://w3c.github.io/webappsec-credential-management/#dom-credential-interface>  
6223 eReferenced in:  
6224 \* 5.1. PublicKeyCredential Interface  
6225 \* 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6) (7)  
6226  
6227 <https://w3c.github.io/webappsec-credential-management/#dom-credential-discovery-remote>  
6228 eReferenced in:  
6229 \* 5.1. PublicKeyCredential Interface  
6230  
6231 <https://w3c.github.io/webappsec-credential-management/#same-origin-with-its-ancestors>  
6232 eReferenced in:  
6233 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6234 `[[Create]](origin, options, sameOriginWithAncestors)` method (2)  
6235 \* 5.1.4.1. PublicKeyCredential's  
6236 `[[DiscoverFromExternalSource]](origin, options,  
6237 sameOriginWithAncestors)` method (2)  
6238 \* 5.1.5. Store an existing credential - PublicKeyCredential's  
6239 `[[Store]](credential, sameOriginWithAncestors)` method  
6240  
6241 <https://w3c.github.io/webappsec-credential-management/#dom-credential-request-options>  
6242 eReferenced in:  
6243 \* 5.1.4.1. PublicKeyCredential's  
6244 `[[DiscoverFromExternalSource]](origin, options,  
6245 sameOriginWithAncestors)` method (2)  
6246  
6247 <https://w3c.github.io/webappsec-credential-management/#dom-credential-scoped-credential-interface-store>  
6248 eReferenced in:  
6249 \* 5.1.5. Store an existing credential - PublicKeyCredential's  
6250 `[[Store]](credential, sameOriginWithAncestors)` method  
6251  
6252 <https://w3c.github.io/webappsec-credential-management/#dom-credential-type>  
6253 eReferenced in:  
6254 \* 5.1. PublicKeyCredential Interface  
6255  
6256 <https://w3c.github.io/webappsec-credential-management/#user-mediated-credential-interface>  
6257 eReferenced in:  
6258 \* 5.1.4. Use an existing credential to make an assertion -  
6259 PublicKeyCredential's `[[Get]](options)` method  
6260  
6261 <https://dom.spec.whatwg.org/#abortcontroller>  
6262 eReferenced in:  
6263 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6264 `[[Create]](origin, options, sameOriginWithAncestors)` method

6223 \* 5.6. Abort operations with AbortSignal (2)  
6224  
6225 https://dom.spec.whatwg.org/#abortsignal-aborted-flagReferenced in:  
6226 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6227 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6228 \* 5.1.4.1. PublicKeyCredential's  
6229 [[DiscoverFromExternalSource]](origin, options,  
6230 sameOriginWithAncestors) method (2)  
6231 \* 5.6. Abort operations with AbortSignal (2)  
6232  
6233 https://dom.spec.whatwg.org/#concept-documentReferenced in:  
6234 \* 5.6. Abort operations with AbortSignal  
6235  
6236 https://tc39.github.io/ecma262/#sec-arraybuffer-constructorReferenced  
6237 in:  
6238 \* 3. Dependencies  
6239 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6240 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6241 \* 5.1.4.1. PublicKeyCredential's  
6242 [[DiscoverFromExternalSource]](origin, options,  
6243 sameOriginWithAncestors) method (2) (3) (4) (5) (6)  
6244 \* 9. WebAuthn Extensions (2)  
6245  
6246 https://tc39.github.io/ecma262/#sec-object-internal-methods-and-interna  
6247 l-slotsReferenced in:  
6248 \* 4. Terminology  
6249 \* 5.1. PublicKeyCredential Interface (2) (3) (4) (5)  
6250 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6251 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6252 \* 5.1.4. Use an existing credential to make an assertion -  
6253 PublicKeyCredential's [[Get]](options) method  
6254 \* 5.1.4.1. PublicKeyCredential's  
6255 [[DiscoverFromExternalSource]](origin, options,  
6256 sameOriginWithAncestors) method  
6257 \* 5.1.5. Store an existing credential - PublicKeyCredential's  
6258 [[Store]](credential, sameOriginWithAncestors) method  
6259 \* 5.1.6. Preventing silent access to an existing credential -  
6260 PublicKeyCredential's [[preventSilentAccess]](credential,  
6261 sameOriginWithAncestors) method  
6262  
6263 https://tc39.github.io/ecma262/#sec-object-internal-methods-and-interna  
6264 l-slotsReferenced in:  
6265 \* 4. Terminology  
6266 \* 5.1. PublicKeyCredential Interface (2) (3) (4) (5)  
6267 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6268 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6269 \* 5.1.4. Use an existing credential to make an assertion -  
6270 PublicKeyCredential's [[Get]](options) method  
6271 \* 5.1.4.1. PublicKeyCredential's  
6272 [[DiscoverFromExternalSource]](origin, options,  
6273 sameOriginWithAncestors) method  
6274 \* 5.1.5. Store an existing credential - PublicKeyCredential's  
6275 [[Store]](credential, sameOriginWithAncestors) method  
6276 \* 5.1.6. Preventing silent access to an existing credential -  
6277 PublicKeyCredential's [[preventSilentAccess]](credential,  
6278 sameOriginWithAncestors) method  
6279  
6280 https://tc39.github.io/ecma262/#sec-json.stringifyReferenced in:  
6281 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
6282 CollectedClientData)  
6283  
6284 https://encoding.spec.whatwg.org/#utf-8-decodeReferenced in:  
6285 \* 7.1. Registering a new credential (2) (3)  
6286 \* 7.2. Verifying an authentication assertion (2) (3)  
6287  
6288 https://encoding.spec.whatwg.org/#utf-8-encodeReferenced in:  
6289 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
6290 CollectedClientData)  
6291 \* 8.5. Android SafetyNet Attestation Statement Format  
6292 \* 10.5. Supported Extensions Extension (exts)

6264 \* 5.6. Abort operations with AbortSignal (2)  
6265  
6266 https://dom.spec.whatwg.org/#abortsignal-aborted-flagReferenced in:  
6267 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6268 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6269 \* 5.1.4.1. PublicKeyCredential's  
6270 [[DiscoverFromExternalSource]](origin, options,  
6271 sameOriginWithAncestors) method (2)  
6272 \* 5.6. Abort operations with AbortSignal (2)  
6273  
6274 https://dom.spec.whatwg.org/#concept-documentReferenced in:  
6275 \* 5.6. Abort operations with AbortSignal  
6276  
6277 https://tc39.github.io/ecma262/#sec-arraybuffer-constructorReferenced  
6278 in:  
6279 \* 3. Dependencies  
6280 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6281 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6282 \* 5.1.4.1. PublicKeyCredential's  
6283 [[DiscoverFromExternalSource]](origin, options,  
6284 sameOriginWithAncestors) method (2) (3) (4) (5) (6)  
6285 \* 9. WebAuthn Extensions (2)  
6286  
6287 https://tc39.github.io/ecma262/#sec-object-internal-methods-and-interna  
6288 l-slotsReferenced in:  
6289 \* 4. Terminology  
6290 \* 5.1. PublicKeyCredential Interface (2) (3) (4) (5)  
6291 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6292 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6293 \* 5.1.4. Use an existing credential to make an assertion -  
6294 PublicKeyCredential's [[Get]](options) method  
6295 \* 5.1.4.1. PublicKeyCredential's  
6296 [[DiscoverFromExternalSource]](origin, options,  
6297 sameOriginWithAncestors) method  
6298 \* 5.1.5. Store an existing credential - PublicKeyCredential's  
6299 [[Store]](credential, sameOriginWithAncestors) method  
6300 \* 5.1.6. Preventing silent access to an existing credential -  
6301 PublicKeyCredential's [[preventSilentAccess]](credential,  
6302 sameOriginWithAncestors) method  
6303  
6304 https://tc39.github.io/ecma262/#sec-object-internal-methods-and-interna  
6305 l-slotsReferenced in:  
6306 \* 4. Terminology  
6307 \* 5.1. PublicKeyCredential Interface (2) (3) (4) (5)  
6308 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6309 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6310 \* 5.1.4. Use an existing credential to make an assertion -  
6311 PublicKeyCredential's [[Get]](options) method  
6312 \* 5.1.4.1. PublicKeyCredential's  
6313 [[DiscoverFromExternalSource]](origin, options,  
6314 sameOriginWithAncestors) method  
6315 \* 5.1.5. Store an existing credential - PublicKeyCredential's  
6316 [[Store]](credential, sameOriginWithAncestors) method  
6317 \* 5.1.6. Preventing silent access to an existing credential -  
6318 PublicKeyCredential's [[preventSilentAccess]](credential,  
6319 sameOriginWithAncestors) method  
6320  
6321 https://tc39.github.io/ecma262/#sec-json.stringifyReferenced in:  
6322 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
6323 CollectedClientData)  
6324  
6325 https://encoding.spec.whatwg.org/#utf-8-decodeReferenced in:  
6326 \* 7.1. Registering a new credential (2) (3)  
6327 \* 7.2. Verifying an authentication assertion (2) (3)  
6328  
6329 https://encoding.spec.whatwg.org/#utf-8-encodeReferenced in:  
6330 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
6331 CollectedClientData)  
6332 \* 8.5. Android SafetyNet Attestation Statement Format  
6333 \* 10.5. Supported Extensions Extension (exts)

6293 https://fetch.spec.whatwg.org/#concept-request-windowReferenced in:  
6294 \* 5.6. Abort operations with AbortSignal (2)  
6295  
6296 https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-appid-and  
6297 -facets-v1.2-ps-20170411.html#determining-if-a-caller-s-facetid-is-auth  
6298 orized-for-an-appidReferenced in:  
6299 \* 3. Dependencies  
6300 \* 10.1. FIDO AppID Extension (appid)  
6301  
6302 https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-appid-and  
6303 -facets-v1.2-ps-20170411.html#determining-the-facetid-of-a-calling-appl  
6304 icationReferenced in:  
6305 \* 3. Dependencies  
6306 \* 10.1. FIDO AppID Extension (appid)  
6307  
6308 https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-client-to-aut  
6309 henticator-protocol-v2.0-ps-20170927.html#ctap2-canonical-cbor-encoding  
6310 -formReferenced in:  
6311 \* 2.4. All Conformance Classes (2)  
6312 \* 3. Dependencies  
6313 \* 6.3.1. Attested credential data  
6314 \* 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE\_Key  
6315 format  
6316 \* 9. WebAuthn Extensions  
6317  
6318 https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0  
6319 -ps-20170927.html#user-verification-methodsReferenced in:  
6320 \* 10.8. User Verification Method Extension (uvm)  
6321  
6322 https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0  
6323 -ps-20170927.html#key-protection-typesReferenced in:  
6324 \* 10.8. User Verification Method Extension (uvm)  
6325  
6326 https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0  
6327 -ps-20170927.html#matcher-protection-typesReferenced in:  
6328 \* 10.8. User Verification Method Extension (uvm)  
6329  
6330 https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0  
6331 -ps-20170927.html#public-key-representation-formatsReferenced in:  
6332 \* 8.6. FIDO U2F Attestation Statement Format  
6333  
6334 https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-m  
6335 essage-formats-v1.1-id-20160915.html#authentication-request-message---u  
6336 2f\_authenticateReferenced in:  
6337 \* 6.1.2. FIDO U2F signature format compatibility  
6338  
6339 https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-m  
6340 essage-formats-v1.1-id-20160915.html#registration-response-message-succ  
6341 essReferenced in:  
6342 \* 8.6. FIDO U2F Attestation Statement Format (2)  
6343  
6344 https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-m  
6345 essage-formats-v1.1-id-20160915.html#authentication-response-message-su  
6346 ccessReferenced in:  
6347 \* 6.1.2. FIDO U2F signature format compatibility  
6348  
6349 https://dev.w3.org/geo/api/spec-source.html#coordinates\_interfaceRefere  
6350 nced in:  
6351 \* 10.7. Location Extension (loc) (2) (3)  
6352  
6353 https://html.spec.whatwg.org/multipage/origin.html#ascii-serialisation-  
6354 of-an-originReferenced in:  
6355 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6356 [[Create]](origin, options, sameOriginWithAncestors) method  
6357 \* 5.1.4.1. PublicKeyCredential's  
6358 [[DiscoverFromExternalSource]](origin, options,  
6359 sameOriginWithAncestors) method  
6360  
6361 https://html.spec.whatwg.org/multipage/origin.html#concept-origin-effec  
6362

6334 https://fetch.spec.whatwg.org/#concept-request-windowReferenced in:  
6335 \* 5.6. Abort operations with AbortSignal (2)  
6336  
6337 https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-appid-and  
6338 -facets-v1.2-ps-20170411.html#determining-if-a-caller-s-facetid-is-auth  
6339 orized-for-an-appidReferenced in:  
6340 \* 3. Dependencies  
6341 \* 10.1. FIDO AppID Extension (appid)  
6342  
6343 https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-appid-and  
6344 -facets-v1.2-ps-20170411.html#determining-the-facetid-of-a-calling-appl  
6345 icationReferenced in:  
6346 \* 3. Dependencies  
6347 \* 10.1. FIDO AppID Extension (appid)  
6348  
6349 https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-client-to-aut  
6350 henticator-protocol-v2.0-ps-20170927.html#ctap2-canonical-cbor-encoding  
6351 -formReferenced in:  
6352 \* 2.4. All Conformance Classes (2)  
6353 \* 3. Dependencies  
6354 \* 6.3.1. Attested credential data  
6355 \* 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE\_Key  
6356 format  
6357 \* 9. WebAuthn Extensions  
6358  
6359 https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0  
6360 -ps-20170927.html#user-verification-methodsReferenced in:  
6361 \* 10.8. User Verification Method Extension (uvm)  
6362  
6363 https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0  
6364 -ps-20170927.html#key-protection-typesReferenced in:  
6365 \* 10.8. User Verification Method Extension (uvm)  
6366  
6367 https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0  
6368 -ps-20170927.html#matcher-protection-typesReferenced in:  
6369 \* 10.8. User Verification Method Extension (uvm)  
6370  
6371 https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0  
6372 -ps-20170927.html#public-key-representation-formatsReferenced in:  
6373 \* 8.6. FIDO U2F Attestation Statement Format  
6374  
6375 https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-m  
6376 essage-formats-v1.1-id-20160915.html#authentication-request-message---u  
6377 2f\_authenticateReferenced in:  
6378 \* 6.1.2. FIDO U2F signature format compatibility  
6379  
6380 https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-m  
6381 essage-formats-v1.1-id-20160915.html#registration-response-message-succ  
6382 essReferenced in:  
6383 \* 8.6. FIDO U2F Attestation Statement Format (2)  
6384  
6385 https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-m  
6386 essage-formats-v1.1-id-20160915.html#authentication-response-message-su  
6387 ccessReferenced in:  
6388 \* 6.1.2. FIDO U2F signature format compatibility  
6389  
6390 https://dev.w3.org/geo/api/spec-source.html#coordinates\_interfaceRefere  
6391 nced in:  
6392 \* 10.7. Location Extension (loc) (2) (3)  
6393  
6394 https://html.spec.whatwg.org/multipage/origin.html#ascii-serialisation-  
6395 of-an-originReferenced in:  
6396 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6397 [[Create]](origin, options, sameOriginWithAncestors) method  
6398 \* 5.1.4.1. PublicKeyCredential's  
6399 [[DiscoverFromExternalSource]](origin, options,  
6400 sameOriginWithAncestors) method  
6401  
6402 https://html.spec.whatwg.org/multipage/origin.html#concept-origin-effec  
6403

6363 tive-domainReferenced in:  
6364 \* 4. Terminology (2) (3) (4)  
6365 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6366 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6367 (4)  
6368 \* 5.1.4.1. PublicKeyCredential's  
6369 [[DiscoverFromExternalSource]](origin, options,  
6370 sameOriginWithAncestors) method (2) (3) (4)  
6371 \* 5.4. Options for Credential Creation (dictionary  
6372 PublicKeyCredentialCreationOptions)  
6373 \* 5.5. Options for Assertion Generation (dictionary  
6374 PublicKeyCredentialRequestOptions)  
6375 \* 6.1. Authenticator data (2)  
6376  
6377 <https://html.spec.whatwg.org/multipage/webappapis.html#environment-sett>  
6378 [ings-object](#)Referenced in:  
6379 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6380 [[Create]](origin, options, sameOriginWithAncestors) method  
6381 \* 5.1.4.1. PublicKeyCredential's  
6382 [[DiscoverFromExternalSource]](origin, options,  
6383 sameOriginWithAncestors) method  
6384 \* 5.1.5. Store an existing credential - PublicKeyCredential's  
6385 [[Store]](credential, sameOriginWithAncestors) method  
6386  
6387 <https://html.spec.whatwg.org/multipage/webappapis.html#concept-settings>  
6388 [-object-global](#)Referenced in:  
6389 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6390 [[Create]](origin, options, sameOriginWithAncestors) method  
6391 \* 5.1.4.1. PublicKeyCredential's  
6392 [[DiscoverFromExternalSource]](origin, options,  
6393 sameOriginWithAncestors) method  
6394  
6395 <https://html.spec.whatwg.org/multipage/origin.html#is-a-registrable-dom>  
6396 [ain-suffix-of-or-is-equal-to](#)Referenced in:  
6397 \* 3. Dependencies  
6398 \* 4. Terminology  
6399 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6400 [[Create]](origin, options, sameOriginWithAncestors) method  
6401 \* 5.1.4.1. PublicKeyCredential's  
6402 [[DiscoverFromExternalSource]](origin, options,  
6403 sameOriginWithAncestors) method  
6404 \* 6.1. Authenticator data  
6405  
6406 <https://html.spec.whatwg.org/multipage/origin.html#is-a-registrable-dom>  
6407 [ain-suffix-of-or-is-equal-to](#)Referenced in:  
6408 \* 3. Dependencies  
6409 \* 4. Terminology  
6410 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6411 [[Create]](origin, options, sameOriginWithAncestors) method  
6412 \* 5.1.4.1. PublicKeyCredential's  
6413 [[DiscoverFromExternalSource]](origin, options,  
6414 sameOriginWithAncestors) method  
6415 \* 6.1. Authenticator data  
6416  
6417 <https://html.spec.whatwg.org/multipage/webappapis.html#concept-settings>  
6418 [-object-origin](#)Referenced in:  
6419 \* 4. Terminology (2)  
6420 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6421 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6422 \* 5.1.4.1. PublicKeyCredential's  
6423 [[DiscoverFromExternalSource]](origin, options,  
6424 sameOriginWithAncestors) method (2)  
6425 \* 5.4. Options for Credential Creation (dictionary  
6426 PublicKeyCredentialCreationOptions)  
6427 \* 5.5. Options for Assertion Generation (dictionary  
6428 PublicKeyCredentialRequestOptions)  
6429  
6430 <https://html.spec.whatwg.org/multipage/webappapis.html#relevant-setting>  
6431 [s-object](#)Referenced in:  
6432 \* 3. Dependencies

6404 tive-domainReferenced in:  
6405 \* 4. Terminology (2) (3) (4)  
6406 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6407 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6408 (4)  
6409 \* 5.1.4.1. PublicKeyCredential's  
6410 [[DiscoverFromExternalSource]](origin, options,  
6411 sameOriginWithAncestors) method (2) (3) (4)  
6412 \* 5.4. Options for Credential Creation (dictionary  
6413 PublicKeyCredentialCreationOptions)  
6414 \* 5.5. Options for Assertion Generation (dictionary  
6415 PublicKeyCredentialRequestOptions)  
6416 \* 6.1. Authenticator data (2)  
6417  
6418 <https://html.spec.whatwg.org/multipage/webappapis.html#environment-sett>  
6419 [ings-object](#)Referenced in:  
6420 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6421 [[Create]](origin, options, sameOriginWithAncestors) method  
6422 \* 5.1.4.1. PublicKeyCredential's  
6423 [[DiscoverFromExternalSource]](origin, options,  
6424 sameOriginWithAncestors) method  
6425 \* 5.1.5. Store an existing credential - PublicKeyCredential's  
6426 [[Store]](credential, sameOriginWithAncestors) method  
6427  
6428 <https://html.spec.whatwg.org/multipage/webappapis.html#concept-settings>  
6429 [-object-global](#)Referenced in:  
6430 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6431 [[Create]](origin, options, sameOriginWithAncestors) method  
6432 \* 5.1.4.1. PublicKeyCredential's  
6433 [[DiscoverFromExternalSource]](origin, options,  
6434 sameOriginWithAncestors) method  
6435  
6436 <https://html.spec.whatwg.org/multipage/origin.html#is-a-registrable-dom>  
6437 [ain-suffix-of-or-is-equal-to](#)Referenced in:  
6438 \* 3. Dependencies  
6439 \* 4. Terminology  
6440 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6441 [[Create]](origin, options, sameOriginWithAncestors) method  
6442 \* 5.1.4.1. PublicKeyCredential's  
6443 [[DiscoverFromExternalSource]](origin, options,  
6444 sameOriginWithAncestors) method  
6445 \* 6.1. Authenticator data  
6446  
6447 <https://html.spec.whatwg.org/multipage/origin.html#is-a-registrable-dom>  
6448 [ain-suffix-of-or-is-equal-to](#)Referenced in:  
6449 \* 3. Dependencies  
6450 \* 4. Terminology  
6451 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6452 [[Create]](origin, options, sameOriginWithAncestors) method  
6453 \* 5.1.4.1. PublicKeyCredential's  
6454 [[DiscoverFromExternalSource]](origin, options,  
6455 sameOriginWithAncestors) method  
6456 \* 6.1. Authenticator data  
6457  
6458 <https://html.spec.whatwg.org/multipage/webappapis.html#concept-settings>  
6459 [-object-origin](#)Referenced in:  
6460 \* 4. Terminology (2)  
6461 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6462 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6463 \* 5.1.4.1. PublicKeyCredential's  
6464 [[DiscoverFromExternalSource]](origin, options,  
6465 sameOriginWithAncestors) method (2)  
6466 \* 5.4. Options for Credential Creation (dictionary  
6467 PublicKeyCredentialCreationOptions)  
6468 \* 5.5. Options for Assertion Generation (dictionary  
6469 PublicKeyCredentialRequestOptions)  
6470  
6471 <https://html.spec.whatwg.org/multipage/webappapis.html#relevant-setting>  
6472 [s-object](#)Referenced in:  
6473 \* 3. Dependencies

6433 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6434 [[Create]](origin, options, sameOriginWithAncestors) method  
6435 \* 5.1.4.1. PublicKeyCredential's  
6436 [[DiscoverFromExternalSource]](origin, options,  
6437 sameOriginWithAncestors) method  
6438 \* 5.4. Options for Credential Creation (dictionary  
6439 PublicKeyCredentialCreationOptions)  
6440 \* 5.5. Options for Assertion Generation (dictionary  
6441 PublicKeyCredentialRequestOptions)  
6442  
6443 <https://w3c.github.io/html/browsers.html#dom-document-domainReferenced>  
6444 in:  
6445 \* 4. Terminology  
6446  
6447 <https://w3c.github.io/html/browsers.html#opaque-originReferenced> in:  
6448 \* 3. Dependencies  
6449 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6450 [[Create]](origin, options, sameOriginWithAncestors) method  
6451 \* 5.1.4.1. PublicKeyCredential's  
6452 [[DiscoverFromExternalSource]](origin, options,  
6453 sameOriginWithAncestors) method  
6454  
6455 <https://w3c.github.io/html/browsers.html#concept-cross-originReferenced>  
6456 in:  
6457 \* 1. Introduction  
6458 \* 3. Dependencies  
6459 \* 4. Terminology (2) (3)  
6460 \* 5. Web Authentication API (2) (3) (4) (5) (6) (7)  
6461 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
6462 CollectedClientData)  
6463 \* 6.1. Authenticator data  
6464 \* 7.1. Registering a new credential  
6465 \* 7.2. Verifying an authentication assertion  
6466 \* 10.1. FIDO AppID Extension (appid)  
6467 \* 12.3. Authentication  
6468 \* 14.1. Attestation Privacy (2)  
6469  
6470 <https://infra.spec.whatwg.org/#set-appendReferenced> in:  
6471 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6472 [[Create]](origin, options, sameOriginWithAncestors) method  
6473 \* 5.1.4.1. PublicKeyCredential's  
6474 [[DiscoverFromExternalSource]](origin, options,  
6475 sameOriginWithAncestors) method (2)  
6476 \* 6.2.3. The authenticatorGetAssertion operation (2)  
6477  
6478 <https://infra.spec.whatwg.org/#byte-sequenceReferenced> in:  
6479 \* 4. Terminology (2)  
6480  
6481 <https://infra.spec.whatwg.org/#iteration-continueReferenced> in:  
6482 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6483 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6484 (4) (5) (6) (7) (8)  
6485 \* 5.1.4.1. PublicKeyCredential's  
6486 [[DiscoverFromExternalSource]](origin, options,  
6487 sameOriginWithAncestors) method (2) (3) (4) (5)  
6488  
6489 <https://infra.spec.whatwg.org/#map-iterateReferenced> in:  
6490 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6491 [[Create]](origin, options, sameOriginWithAncestors) method  
6492 \* 5.1.4.1. PublicKeyCredential's  
6493 [[DiscoverFromExternalSource]](origin, options,  
6494 sameOriginWithAncestors) method  
6495 \* 6.2.1. Lookup Credential Source by Credential ID algorithm  
6496 \* 6.2.2. The authenticatorMakeCredential operation  
6497 \* 6.2.3. The authenticatorGetAssertion operation (2)  
6498  
6499 <https://infra.spec.whatwg.org/#list-is-emptyReferenced> in:  
6500 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6501 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6502 \* 5.1.4.1. PublicKeyCredential's

6474 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6475 [[Create]](origin, options, sameOriginWithAncestors) method  
6476 \* 5.1.4.1. PublicKeyCredential's  
6477 [[DiscoverFromExternalSource]](origin, options,  
6478 sameOriginWithAncestors) method  
6479 \* 5.4. Options for Credential Creation (dictionary  
6480 PublicKeyCredentialCreationOptions)  
6481 \* 5.5. Options for Assertion Generation (dictionary  
6482 PublicKeyCredentialRequestOptions)  
6483  
6484 <https://w3c.github.io/html/browsers.html#dom-document-domainReferenced>  
6485 in:  
6486 \* 4. Terminology  
6487  
6488 <https://w3c.github.io/html/browsers.html#opaque-originReferenced> in:  
6489 \* 3. Dependencies  
6490 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6491 [[Create]](origin, options, sameOriginWithAncestors) method  
6492 \* 5.1.4.1. PublicKeyCredential's  
6493 [[DiscoverFromExternalSource]](origin, options,  
6494 sameOriginWithAncestors) method  
6495  
6496 <https://w3c.github.io/html/browsers.html#concept-cross-originReferenced>  
6497 in:  
6498 \* 1. Introduction  
6499 \* 3. Dependencies  
6500 \* 4. Terminology (2) (3)  
6501 \* 5. Web Authentication API (2) (3) (4) (5) (6) (7)  
6502 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
6503 CollectedClientData)  
6504 \* 6.1. Authenticator data  
6505 \* 7.1. Registering a new credential  
6506 \* 7.2. Verifying an authentication assertion  
6507 \* 10.1. FIDO AppID Extension (appid)  
6508 \* 12.3. Authentication  
6509 \* 14.1. Attestation Privacy (2)  
6510  
6511 <https://infra.spec.whatwg.org/#set-appendReferenced> in:  
6512 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6513 [[Create]](origin, options, sameOriginWithAncestors) method  
6514 \* 5.1.4.1. PublicKeyCredential's  
6515 [[DiscoverFromExternalSource]](origin, options,  
6516 sameOriginWithAncestors) method (2)  
6517 \* 6.2.3. The authenticatorGetAssertion operation (2)  
6518  
6519 <https://infra.spec.whatwg.org/#byte-sequenceReferenced> in:  
6520 \* 4. Terminology (2)  
6521  
6522 <https://infra.spec.whatwg.org/#iteration-continueReferenced> in:  
6523 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6524 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6525 (4) (5) (6) (7) (8)  
6526 \* 5.1.4.1. PublicKeyCredential's  
6527 [[DiscoverFromExternalSource]](origin, options,  
6528 sameOriginWithAncestors) method (2) (3) (4) (5)  
6529  
6530 <https://infra.spec.whatwg.org/#map-iterateReferenced> in:  
6531 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6532 [[Create]](origin, options, sameOriginWithAncestors) method  
6533 \* 5.1.4.1. PublicKeyCredential's  
6534 [[DiscoverFromExternalSource]](origin, options,  
6535 sameOriginWithAncestors) method  
6536 \* 6.2.1. Lookup Credential Source by Credential ID algorithm  
6537 \* 6.2.2. The authenticatorMakeCredential operation  
6538 \* 6.2.3. The authenticatorGetAssertion operation (2)  
6539  
6540 <https://infra.spec.whatwg.org/#list-is-emptyReferenced> in:  
6541 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6542 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6543 \* 5.1.4.1. PublicKeyCredential's

```

6503 [[DiscoverFromExternalSource]](origin, options,
6504 sameOriginWithAncestors) method (2) (3) (4) (5)
6505
6506 https://infra.spec.whatwg.org/#list-is-emptyReferenced in:
6507 * 5.1.3. Create a new credential - PublicKeyCredential's
6508 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
6509 * 5.1.4.1. PublicKeyCredential's
6510 [[DiscoverFromExternalSource]](origin, options,
6511 sameOriginWithAncestors) method (2) (3) (4) (5)
6512
6513 https://infra.spec.whatwg.org/#struct-itemReferenced in:
6514 * 4. Terminology (2)
6515 * 5.1.3. Create a new credential - PublicKeyCredential's
6516 [[Create]](origin, options, sameOriginWithAncestors) method
6517 * 5.1.4.1. PublicKeyCredential's
6518 [[DiscoverFromExternalSource]](origin, options,
6519 sameOriginWithAncestors) method
6520
6521 https://infra.spec.whatwg.org/#listReferenced in:
6522 * 5.1.3. Create a new credential - PublicKeyCredential's
6523 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6524 * 5.1.4.1. PublicKeyCredential's
6525 [[DiscoverFromExternalSource]](origin, options,
6526 sameOriginWithAncestors) method
6527 * 6.2.3. The authenticatorGetAssertion operation
6528
6529 https://infra.spec.whatwg.org/#ordered-mapReferenced in:
6530 * 5.1. PublicKeyCredential Interface
6531 * 5.1.3. Create a new credential - PublicKeyCredential's
6532 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6533 * 5.1.4.1. PublicKeyCredential's
6534 [[DiscoverFromExternalSource]](origin, options,
6535 sameOriginWithAncestors) method (2)
6536 * 6. WebAuthn Authenticator Model
6537 * 6.2.2. The authenticatorMakeCredential operation
6538 * 6.2.3. The authenticatorGetAssertion operation
6539 * 9.4. Client extension processing
6540
6541 https://infra.spec.whatwg.org/#ordered-setReferenced in:
6542 * 5.1.3. Create a new credential - PublicKeyCredential's
6543 [[Create]](origin, options, sameOriginWithAncestors) method
6544 * 5.1.4.1. PublicKeyCredential's
6545 [[DiscoverFromExternalSource]](origin, options,
6546 sameOriginWithAncestors) method (2) (3)
6547 * 6.2.3. The authenticatorGetAssertion operation
6548
6549 https://infra.spec.whatwg.org/#list-removeReferenced in:
6550 * 5.1.3. Create a new credential - PublicKeyCredential's
6551 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
6552 (4) (5) (6) (7) (8) (9)
6553 * 5.1.4.1. PublicKeyCredential's
6554 [[DiscoverFromExternalSource]](origin, options,
6555 sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7)
6556 * 6.2.3. The authenticatorGetAssertion operation
6557
6558 https://infra.spec.whatwg.org/#map-setReferenced in:
6559 * 5.1.3. Create a new credential - PublicKeyCredential's
6560 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6561 * 5.1.4.1. PublicKeyCredential's
6562 [[DiscoverFromExternalSource]](origin, options,
6563 sameOriginWithAncestors) method (2)
6564 * 6.2.2. The authenticatorMakeCredential operation
6565
6566 https://infra.spec.whatwg.org/#structReferenced in:
6567 * 4. Terminology
6568 * 5.1.3. Create a new credential - PublicKeyCredential's
6569 [[Create]](origin, options, sameOriginWithAncestors) method
6570 * 5.1.4.1. PublicKeyCredential's
6571 [[DiscoverFromExternalSource]](origin, options,
6572 sameOriginWithAncestors) method

```

```

6544 [[DiscoverFromExternalSource]](origin, options,
6545 sameOriginWithAncestors) method (2) (3) (4) (5)
6546
6547 https://infra.spec.whatwg.org/#list-is-emptyReferenced in:
6548 * 5.1.3. Create a new credential - PublicKeyCredential's
6549 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
6550 * 5.1.4.1. PublicKeyCredential's
6551 [[DiscoverFromExternalSource]](origin, options,
6552 sameOriginWithAncestors) method (2) (3) (4) (5)
6553
6554 https://infra.spec.whatwg.org/#struct-itemReferenced in:
6555 * 4. Terminology (2)
6556 * 5.1.3. Create a new credential - PublicKeyCredential's
6557 [[Create]](origin, options, sameOriginWithAncestors) method
6558 * 5.1.4.1. PublicKeyCredential's
6559 [[DiscoverFromExternalSource]](origin, options,
6560 sameOriginWithAncestors) method
6561
6562 https://infra.spec.whatwg.org/#listReferenced in:
6563 * 5.1.3. Create a new credential - PublicKeyCredential's
6564 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6565 * 5.1.4.1. PublicKeyCredential's
6566 [[DiscoverFromExternalSource]](origin, options,
6567 sameOriginWithAncestors) method
6568 * 6.2.3. The authenticatorGetAssertion operation
6569
6570 https://infra.spec.whatwg.org/#ordered-mapReferenced in:
6571 * 5.1. PublicKeyCredential Interface
6572 * 5.1.3. Create a new credential - PublicKeyCredential's
6573 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6574 * 5.1.4.1. PublicKeyCredential's
6575 [[DiscoverFromExternalSource]](origin, options,
6576 sameOriginWithAncestors) method (2) (3)
6577 * 6. WebAuthn Authenticator Model
6578 * 6.2.2. The authenticatorMakeCredential operation
6579 * 6.2.3. The authenticatorGetAssertion operation
6580 * 9.4. Client extension processing
6581
6582 https://infra.spec.whatwg.org/#ordered-setReferenced in:
6583 * 5.1.3. Create a new credential - PublicKeyCredential's
6584 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6585 * 5.1.4.1. PublicKeyCredential's
6586 [[DiscoverFromExternalSource]](origin, options,
6587 sameOriginWithAncestors) method (2) (3) (4)
6588 * 6.2.3. The authenticatorGetAssertion operation
6589
6590 https://infra.spec.whatwg.org/#list-removeReferenced in:
6591 * 5.1.3. Create a new credential - PublicKeyCredential's
6592 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
6593 (4) (5) (6) (7) (8) (9) (10)
6594 * 5.1.4.1. PublicKeyCredential's
6595 [[DiscoverFromExternalSource]](origin, options,
6596 sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7) (8) (9)
6597 * 6.2.3. The authenticatorGetAssertion operation
6598
6599 https://infra.spec.whatwg.org/#map-setReferenced in:
6600 * 5.1.3. Create a new credential - PublicKeyCredential's
6601 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6602 * 5.1.4.1. PublicKeyCredential's
6603 [[DiscoverFromExternalSource]](origin, options,
6604 sameOriginWithAncestors) method (2)
6605 * 6.2.2. The authenticatorMakeCredential operation
6606
6607 https://infra.spec.whatwg.org/#structReferenced in:
6608 * 4. Terminology
6609 * 5.1.3. Create a new credential - PublicKeyCredential's
6610 [[Create]](origin, options, sameOriginWithAncestors) method
6611 * 5.1.4.1. PublicKeyCredential's
6612 [[DiscoverFromExternalSource]](origin, options,
6613 sameOriginWithAncestors) method

```

6573  
6574  
6575  
6576  
6577  
6578  
6579  
6580  
6581  
6582  
6583  
6584  
6585  
6586  
6587  
6588  
6589  
6590  
6591  
6592  
6593  
6594  
6595  
6596  
6597  
6598  
6599  
6600  
6601  
6602  
6603  
6604  
6605  
6606  
6607  
6608  
6609  
6610  
6611  
6612  
6613  
6614  
6615  
6616  
6617  
6618  
6619  
6620  
6621  
6622  
6623  
6624  
6625  
6626  
6627  
6628  
6629  
6630  
6631  
6632  
6633  
6634  
6635  
6636  
6637  
6638  
6639  
6640  
6641  
6642

<https://infra.spec.whatwg.org/#iteration-whileReferenced in:>  
\* 5.1.3. Create a new credential - PublicKeyCredential's  
[[Create]](origin, options, sameOriginWithAncestors) method  
\* 5.1.4.1. PublicKeyCredential's  
[[DiscoverFromExternalSource]](origin, options,  
sameOriginWithAncestors) method  
<https://infra.spec.whatwg.org/#willful-violationReferenced in:>  
\* 4. Terminology  
<https://w3c.github.io/webappsec-mixed-content/#a-priori-authenticated-urReferenced in:>  
\* 5.4.1. Public Key Entity Description (dictionary  
PublicKeyCredentialEntity)  
<https://www.w3.org/TR/page-visibility/#visibility-statesReferenced in:>  
\* 5.6. Abort operations with AbortSignal  
<https://tools.ietf.org/html/rfc4949#page-182Referenced in:>  
\* 13.3.1. Considerations for Self and None Attestation Types and  
Ignoring Attestation  
<https://tools.ietf.org/html/rfc4949#page-186Referenced in:>  
\* 13.3. Security Benefits for Relying Parties  
\* 13.3.1. Considerations for Self and None Attestation Types and  
Ignoring Attestation (2) (3) (4)  
<https://tools.ietf.org/html/rfc8152#section-7Referenced in:>  
\* 6.3.1. Attested credential data  
\* 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE\_Key  
format  
\* 8.6. FIDO U2F Attestation Statement Format  
<https://w3c.github.io/webappsec-secure-contexts/#secure-contextsReferenced in:>  
\* 5. Web Authentication API  
\* 5.1.3. Create a new credential - PublicKeyCredential's  
[[Create]](origin, options, sameOriginWithAncestors) method  
\* 5.1.4.1. PublicKeyCredential's  
[[DiscoverFromExternalSource]](origin, options,  
sameOriginWithAncestors) method  
<https://tools.ietf.org/html/draft-ietf-tokbind-protocol#token-bindingReferenced in:>  
\* 5.1.3. Create a new credential - PublicKeyCredential's  
[[Create]](origin, options, sameOriginWithAncestors) method  
\* 5.1.4.1. PublicKeyCredential's  
[[DiscoverFromExternalSource]](origin, options,  
sameOriginWithAncestors) method  
\* 5.10.1. Client data used in WebAuthn signatures (dictionary  
CollectedClientData)  
\* 7.1. Registering a new credential (2)  
\* 7.2. Verifying an authentication assertion (2)  
<https://tools.ietf.org/html/draft-ietf-tokbind-protocol#section-3.2Referenced in:>  
\* 5.1.3. Create a new credential - PublicKeyCredential's  
[[Create]](origin, options, sameOriginWithAncestors) method  
\* 5.1.4.1. PublicKeyCredential's  
[[DiscoverFromExternalSource]](origin, options,  
sameOriginWithAncestors) method  
\* 5.10.1. Client data used in WebAuthn signatures (dictionary  
CollectedClientData)  
\* 7.1. Registering a new credential  
\* 7.2. Verifying an authentication assertion  
<https://url.spec.whatwg.org/#concept-domainReferenced in:>  
\* 5.1.3. Create a new credential - PublicKeyCredential's  
[[Create]](origin, options, sameOriginWithAncestors) method (2)

6614  
6615  
6616  
6617  
6618  
6619  
6620  
6621  
6622  
6623  
6624  
6625  
6626  
6627  
6628  
6629  
6630  
6631  
6632  
6633  
6634  
6635  
6636  
6637  
6638  
6639  
6640  
6641  
6642  
6643  
6644  
6645  
6646  
6647  
6648  
6649  
6650  
6651  
6652  
6653  
6654  
6655  
6656  
6657  
6658  
6659  
6660  
6661  
6662  
6663  
6664  
6665  
6666  
6667  
6668  
6669  
6670  
6671  
6672  
6673  
6674  
6675  
6676  
6677  
6678  
6679  
6680  
6681  
6682  
6683

<https://infra.spec.whatwg.org/#iteration-whileReferenced in:>  
\* 5.1.3. Create a new credential - PublicKeyCredential's  
[[Create]](origin, options, sameOriginWithAncestors) method  
\* 5.1.4.1. PublicKeyCredential's  
[[DiscoverFromExternalSource]](origin, options,  
sameOriginWithAncestors) method  
<https://infra.spec.whatwg.org/#willful-violationReferenced in:>  
\* 4. Terminology  
<https://w3c.github.io/webappsec-mixed-content/#a-priori-authenticated-urReferenced in:>  
\* 5.4.1. Public Key Entity Description (dictionary  
PublicKeyCredentialEntity)  
<https://www.w3.org/TR/page-visibility/#visibility-statesReferenced in:>  
\* 5.6. Abort operations with AbortSignal  
<https://tools.ietf.org/html/rfc4949#page-182Referenced in:>  
\* 13.3.1. Considerations for Self and None Attestation Types and  
Ignoring Attestation  
<https://tools.ietf.org/html/rfc4949#page-186Referenced in:>  
\* 13.3. Security Benefits for Relying Parties  
\* 13.3.1. Considerations for Self and None Attestation Types and  
Ignoring Attestation (2) (3) (4)  
<https://tools.ietf.org/html/rfc8152#section-7Referenced in:>  
\* 6.3.1. Attested credential data  
\* 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE\_Key  
format  
\* 8.6. FIDO U2F Attestation Statement Format  
<https://w3c.github.io/webappsec-secure-contexts/#secure-contextsReferenced in:>  
\* 5. Web Authentication API  
\* 5.1.3. Create a new credential - PublicKeyCredential's  
[[Create]](origin, options, sameOriginWithAncestors) method  
\* 5.1.4.1. PublicKeyCredential's  
[[DiscoverFromExternalSource]](origin, options,  
sameOriginWithAncestors) method  
<https://tools.ietf.org/html/draft-ietf-tokbind-protocol#token-bindingReferenced in:>  
\* 5.1.3. Create a new credential - PublicKeyCredential's  
[[Create]](origin, options, sameOriginWithAncestors) method  
\* 5.1.4.1. PublicKeyCredential's  
[[DiscoverFromExternalSource]](origin, options,  
sameOriginWithAncestors) method  
\* 5.10.1. Client data used in WebAuthn signatures (dictionary  
CollectedClientData)  
\* 7.1. Registering a new credential (2)  
\* 7.2. Verifying an authentication assertion (2)  
<https://tools.ietf.org/html/draft-ietf-tokbind-protocol#section-3.2Referenced in:>  
\* 5.1.3. Create a new credential - PublicKeyCredential's  
[[Create]](origin, options, sameOriginWithAncestors) method  
\* 5.1.4.1. PublicKeyCredential's  
[[DiscoverFromExternalSource]](origin, options,  
sameOriginWithAncestors) method  
\* 5.10.1. Client data used in WebAuthn signatures (dictionary  
CollectedClientData)  
\* 7.1. Registering a new credential  
\* 7.2. Verifying an authentication assertion  
<https://url.spec.whatwg.org/#concept-domainReferenced in:>  
\* 5.1.3. Create a new credential - PublicKeyCredential's  
[[Create]](origin, options, sameOriginWithAncestors) method (2)



6643 \* 5.1.4.1. PublicKeyCredential's  
6644 [[DiscoverFromExternalSource]](origin, options,  
6645 sameOriginWithAncestors) method (2)  
6646  
6647 https://url.spec.whatwg.org/#empty-hostReferenced in:  
6648 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6649 [[Create]](origin, options, sameOriginWithAncestors) method  
6650 \* 5.1.4.1. PublicKeyCredential's  
6651 [[DiscoverFromExternalSource]](origin, options,  
6652 sameOriginWithAncestors) method  
6653  
6654 https://url.spec.whatwg.org/#concept-url-hostReferenced in:  
6655 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6656 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6657 \* 5.1.4.1. PublicKeyCredential's  
6658 [[DiscoverFromExternalSource]](origin, options,  
6659 sameOriginWithAncestors) method (2)  
6660  
6661 https://url.spec.whatwg.org/#concept-ipv4Referenced in:  
6662 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6663 [[Create]](origin, options, sameOriginWithAncestors) method  
6664 \* 5.1.4.1. PublicKeyCredential's  
6665 [[DiscoverFromExternalSource]](origin, options,  
6666 sameOriginWithAncestors) method  
6667  
6668 https://url.spec.whatwg.org/#concept-ipv6Referenced in:  
6669 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6670 [[Create]](origin, options, sameOriginWithAncestors) method  
6671 \* 5.1.4.1. PublicKeyCredential's  
6672 [[DiscoverFromExternalSource]](origin, options,  
6673 sameOriginWithAncestors) method  
6674  
6675 https://url.spec.whatwg.org/#opaque-hostReferenced in:  
6676 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6677 [[Create]](origin, options, sameOriginWithAncestors) method  
6678 \* 5.1.4.1. PublicKeyCredential's  
6679 [[DiscoverFromExternalSource]](origin, options,  
6680 sameOriginWithAncestors) method  
6681  
6682 https://url.spec.whatwg.org/#concept-url-serializerReferenced in:  
6683 \* 5.4.1. Public Key Entity Description (dictionary  
6684 PublicKeyCredentialEntity)  
6685  
6686 https://url.spec.whatwg.org/#valid-domainReferenced in:  
6687 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6688 [[Create]](origin, options, sameOriginWithAncestors) method  
6689 \* 5.1.4.1. PublicKeyCredential's  
6690 [[DiscoverFromExternalSource]](origin, options,  
6691 sameOriginWithAncestors) method  
6692  
6693 https://url.spec.whatwg.org/#valid-domain-stringReferenced in:  
6694 \* 4. Terminology  
6695  
6696 https://heycam.github.io/webidl/#aborterrorReferenced in:  
6697 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6698 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6699 \* 5.1.4.1. PublicKeyCredential's  
6700 [[DiscoverFromExternalSource]](origin, options,  
6701 sameOriginWithAncestors) method (2)  
6702  
6703 https://heycam.github.io/webidl/#idl-ArrayBufferReferenced in:  
6704 \* 5.1. PublicKeyCredential Interface (2)  
6705 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6706 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6707 \* 5.1.4.1. PublicKeyCredential's  
6708 [[DiscoverFromExternalSource]](origin, options,  
6709 sameOriginWithAncestors) method (2) (3) (4) (5) (6)  
6710 \* 5.2. Authenticator Responses (interface AuthenticatorResponse) (2)  
6711 \* 5.2.1. Information about Public Key Credential (interface  
6712 AuthenticatorAttestationResponse) (2)

6684 \* 5.1.4.1. PublicKeyCredential's  
6685 [[DiscoverFromExternalSource]](origin, options,  
6686 sameOriginWithAncestors) method (2)  
6687  
6688 https://url.spec.whatwg.org/#empty-hostReferenced in:  
6689 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6690 [[Create]](origin, options, sameOriginWithAncestors) method  
6691 \* 5.1.4.1. PublicKeyCredential's  
6692 [[DiscoverFromExternalSource]](origin, options,  
6693 sameOriginWithAncestors) method  
6694  
6695 https://url.spec.whatwg.org/#concept-url-hostReferenced in:  
6696 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6697 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6698 \* 5.1.4.1. PublicKeyCredential's  
6699 [[DiscoverFromExternalSource]](origin, options,  
6700 sameOriginWithAncestors) method (2)  
6701  
6702 https://url.spec.whatwg.org/#concept-ipv4Referenced in:  
6703 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6704 [[Create]](origin, options, sameOriginWithAncestors) method  
6705 \* 5.1.4.1. PublicKeyCredential's  
6706 [[DiscoverFromExternalSource]](origin, options,  
6707 sameOriginWithAncestors) method  
6708  
6709 https://url.spec.whatwg.org/#concept-ipv6Referenced in:  
6710 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6711 [[Create]](origin, options, sameOriginWithAncestors) method  
6712 \* 5.1.4.1. PublicKeyCredential's  
6713 [[DiscoverFromExternalSource]](origin, options,  
6714 sameOriginWithAncestors) method  
6715  
6716 https://url.spec.whatwg.org/#opaque-hostReferenced in:  
6717 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6718 [[Create]](origin, options, sameOriginWithAncestors) method  
6719 \* 5.1.4.1. PublicKeyCredential's  
6720 [[DiscoverFromExternalSource]](origin, options,  
6721 sameOriginWithAncestors) method  
6722  
6723 https://url.spec.whatwg.org/#concept-url-serializerReferenced in:  
6724 \* 5.4.1. Public Key Entity Description (dictionary  
6725 PublicKeyCredentialEntity)  
6726  
6727 https://url.spec.whatwg.org/#valid-domainReferenced in:  
6728 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6729 [[Create]](origin, options, sameOriginWithAncestors) method  
6730 \* 5.1.4.1. PublicKeyCredential's  
6731 [[DiscoverFromExternalSource]](origin, options,  
6732 sameOriginWithAncestors) method  
6733  
6734 https://url.spec.whatwg.org/#valid-domain-stringReferenced in:  
6735 \* 4. Terminology  
6736  
6737 https://heycam.github.io/webidl/#aborterrorReferenced in:  
6738 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6739 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6740 \* 5.1.4.1. PublicKeyCredential's  
6741 [[DiscoverFromExternalSource]](origin, options,  
6742 sameOriginWithAncestors) method (2)  
6743  
6744 https://heycam.github.io/webidl/#idl-ArrayBufferReferenced in:  
6745 \* 5.1. PublicKeyCredential Interface (2)  
6746 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6747 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6748 \* 5.1.4.1. PublicKeyCredential's  
6749 [[DiscoverFromExternalSource]](origin, options,  
6750 sameOriginWithAncestors) method (2) (3) (4) (5) (6)  
6751 \* 5.2. Authenticator Responses (interface AuthenticatorResponse) (2)  
6752 \* 5.2.1. Information about Public Key Credential (interface  
6753 AuthenticatorAttestationResponse) (2)

6713 \* 5.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse) (2) (3) (4) (5) (6)

6714 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric) (2)

6715 \* 10.6. User Verification Index Extension (uvi)

6716

6717 https://heycam.github.io/webidl/#BufferSourceReferenced in:

6718 \* 5.4. Options for Credential Creation (dictionary PublicKeyCredentialCreationOptions) (2)

6719 \* 5.4.3. User Account Parameters for Credential Generation (dictionary PublicKeyCredentialUserEntity) (2)

6720 \* 5.5. Options for Assertion Generation (dictionary PublicKeyCredentialRequestOptions) (2)

6721 \* 5.10.3. Credential Descriptor (dictionary PublicKeyCredentialDescriptor)

6722 \* 10.4. Authenticator Selection Extension (authnSel)

6723

6724 https://heycam.github.io/webidl/#constrainterrorReferenced in:

6725 \* 6.2.2. The authenticatorMakeCredential operation (2)

6726

6727 https://heycam.github.io/webidl/#idl-DOMExceptionReferenced in:

6728 \* 3. Dependencies

6729 \* 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7) (8) (9)

6730 \* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7)

6731 \* 5.1.5. Store an existing credential - PublicKeyCredential's [[Store]](credential, sameOriginWithAncestors) method

6732 \* 10.1. FIDO AppID Extension (appid) (2)

6733

6734 https://heycam.github.io/webidl/#idl-DOMStringReferenced in:

6735 \* 5.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity) (2)

6736 \* 5.4.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRpEntity) (2)

6737 \* 5.4.3. User Account Parameters for Credential Generation (dictionary PublicKeyCredentialUserEntity) (2)

6738 \* 5.9. Authentication Extensions Authenticator Inputs (typedef AuthenticationExtensionsAuthenticatorInputs) (2)

6739 \* 5.10.1. Client data used in WebAuthn signatures (dictionary CollectedClientData) (2) (3) (4)

6740

6741 https://heycam.github.io/webidl/#ExposedReferenced in:

6742 \* 5.1. PublicKeyCredential Interface

6743 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)

6744 \* 5.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse)

6745 \* 5.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse)

6746

6747 https://heycam.github.io/webidl/#invalidstateerrorReferenced in:

6748 \* 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)

6749 \* 6.2.2. The authenticatorMakeCredential operation

6750

6751 https://heycam.github.io/webidl/#notallowederrorReferenced in:

6752 \* 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)

6753 \* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method (2) (3)

6754 \* 6.2.2. The authenticatorMakeCredential operation (2)

6755 \* 6.2.3. The authenticatorGetAssertion operation (2)

6756

6757 https://heycam.github.io/webidl/#notsupportederrorReferenced in:

6758 \* 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method

6759 \* 5.1.5. Store an existing credential - PublicKeyCredential's

6754 \* 5.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse) (2) (3) (4) (5) (6)

6755 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric) (2)

6756 \* 10.6. User Verification Index Extension (uvi)

6757

6758 https://heycam.github.io/webidl/#BufferSourceReferenced in:

6759 \* 5.4. Options for Credential Creation (dictionary PublicKeyCredentialCreationOptions) (2)

6760 \* 5.4.3. User Account Parameters for Credential Generation (dictionary PublicKeyCredentialUserEntity) (2)

6761 \* 5.5. Options for Assertion Generation (dictionary PublicKeyCredentialRequestOptions) (2)

6762 \* 5.10.3. Credential Descriptor (dictionary PublicKeyCredentialDescriptor)

6763 \* 10.4. Authenticator Selection Extension (authnSel)

6764

6765 https://heycam.github.io/webidl/#constrainterrorReferenced in:

6766 \* 6.2.2. The authenticatorMakeCredential operation (2)

6767

6768 https://heycam.github.io/webidl/#idl-DOMExceptionReferenced in:

6769 \* 3. Dependencies

6770 \* 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7) (8) (9)

6771 \* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7) (8) (9)

6772 \* 5.1.5. Store an existing credential - PublicKeyCredential's [[Store]](credential, sameOriginWithAncestors) method

6773 \* 10.1. FIDO AppID Extension (appid) (2)

6774

6775 https://heycam.github.io/webidl/#idl-DOMStringReferenced in:

6776 \* 5.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity) (2)

6777 \* 5.4.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRpEntity) (2)

6778 \* 5.4.3. User Account Parameters for Credential Generation (dictionary PublicKeyCredentialUserEntity) (2)

6779 \* 5.9. Authentication Extensions Authenticator Inputs (typedef AuthenticationExtensionsAuthenticatorInputs) (2)

6780 \* 5.10.1. Client data used in WebAuthn signatures (dictionary CollectedClientData) (2) (3) (4)

6781

6782 https://heycam.github.io/webidl/#ExposedReferenced in:

6783 \* 5.1. PublicKeyCredential Interface

6784 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)

6785 \* 5.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse)

6786 \* 5.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse)

6787

6788 https://heycam.github.io/webidl/#invalidstateerrorReferenced in:

6789 \* 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)

6790 \* 6.2.2. The authenticatorMakeCredential operation

6791

6792 https://heycam.github.io/webidl/#notallowederrorReferenced in:

6793 \* 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)

6794 \* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method (2) (3) (4) (5)

6795 \* 6.2.2. The authenticatorMakeCredential operation (2)

6796 \* 6.2.3. The authenticatorGetAssertion operation (2)

6797

6798 https://heycam.github.io/webidl/#notsupportederrorReferenced in:

6799 \* 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method

6800 \* 5.1.5. Store an existing credential - PublicKeyCredential's

6783 [[Store]](credential, sameOriginWithAncestors) method  
6784 \* 6.2.2. The authenticatorMakeCredential operation  
6785 \* 10.1. FIDO AppID Extension (appid)  
6786  
6787 https://heycam.github.io/webidl/#idl-promiseReferenced in:  
6788 \* 3. Dependencies  
6789 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6790 [[Create]](origin, options, sameOriginWithAncestors) method  
6791 \* 5.1.4.1. PublicKeyCredential's  
6792 [[DiscoverFromExternalSource]](origin, options,  
6793 sameOriginWithAncestors) method  
6794 \* 5.1.5. Store an existing credential - PublicKeyCredential's  
6795 [[Store]](credential, sameOriginWithAncestors) method  
6796  
6797 https://heycam.github.io/webidl/#SameObjectReferenced in:  
6798 \* 5.1. PublicKeyCredential Interface (2)  
6799 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
6800 \* 5.2.1. Information about Public Key Credential (interface  
6801 AuthenticatorAttestationResponse)  
6802 \* 5.2.2. Web Authentication Assertion (interface  
6803 AuthenticatorAssertionResponse) (2) (3)  
6804  
6805 https://heycam.github.io/webidl/#SecureContextReferenced in:  
6806 \* 5.1. PublicKeyCredential Interface  
6807 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
6808 \* 5.2.1. Information about Public Key Credential (interface  
6809 AuthenticatorAttestationResponse)  
6810 \* 5.2.2. Web Authentication Assertion (interface  
6811 AuthenticatorAssertionResponse)  
6812  
6813 https://heycam.github.io/webidl/#securityerrorReferenced in:  
6814 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6815 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6816 \* 5.1.4.1. PublicKeyCredential's  
6817 [[DiscoverFromExternalSource]](origin, options,  
6818 sameOriginWithAncestors) method (2)  
6819 \* 10.1. FIDO AppID Extension (appid)  
6820  
6821 https://heycam.github.io/webidl/#idl-USVStringReferenced in:  
6822 \* 5.4.1. Public Key Entity Description (dictionary  
6823 PublicKeyCredentialEntity) (2)  
6824 \* 5.5. Options for Assertion Generation (dictionary  
6825 PublicKeyCredentialRequestOptions) (2)  
6826 \* 10.1. FIDO AppID Extension (appid)  
6827 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple) (2)  
6828 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
6829 \* 10.5. Supported Extensions Extension (exts)  
6830  
6831 https://heycam.github.io/webidl/#unknownerrorReferenced in:  
6832 \* 6.2.2. The authenticatorMakeCredential operation (2)  
6833 \* 6.2.3. The authenticatorGetAssertion operation (2)  
6834  
6835 https://heycam.github.io/webidl/#idl-booleanReferenced in:  
6836 \* 5.1.7. Availability of User-Verifying Platform Authenticator -  
6837 PublicKeyCredential's  
6838 isUserVerifyingPlatformAuthenticatorAvailable() method  
6839 \* 5.4.4. Authenticator Selection Criteria (dictionary  
6840 AuthenticatorSelectionCriteria) (2)  
6841 \* 10.1. FIDO AppID Extension (appid)  
6842 \* 10.4. Authenticator Selection Extension (authnSel)  
6843 \* 10.5. Supported Extensions Extension (exts)  
6844 \* 10.6. User Verification Index Extension (uvi)  
6845 \* 10.7. Location Extension (loc)  
6846 \* 10.8. User Verification Method Extension (uvm)  
6847  
6848 https://heycam.github.io/webidl/#idl-floatReferenced in:  
6849 \* 10.9. Biometric Authenticator Performance Bounds Extension  
6850 (biometricPerfBounds) (2)  
6851  
6852 https://heycam.github.io/webidl/#dfn-interface-objectReferenced in:

6824 [[Store]](credential, sameOriginWithAncestors) method  
6825 \* 6.2.2. The authenticatorMakeCredential operation  
6826 \* 10.1. FIDO AppID Extension (appid)  
6827  
6828 https://heycam.github.io/webidl/#idl-promiseReferenced in:  
6829 \* 3. Dependencies  
6830 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6831 [[Create]](origin, options, sameOriginWithAncestors) method  
6832 \* 5.1.4.1. PublicKeyCredential's  
6833 [[DiscoverFromExternalSource]](origin, options,  
6834 sameOriginWithAncestors) method  
6835 \* 5.1.5. Store an existing credential - PublicKeyCredential's  
6836 [[Store]](credential, sameOriginWithAncestors) method  
6837  
6838 https://heycam.github.io/webidl/#SameObjectReferenced in:  
6839 \* 5.1. PublicKeyCredential Interface (2)  
6840 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
6841 \* 5.2.1. Information about Public Key Credential (interface  
6842 AuthenticatorAttestationResponse)  
6843 \* 5.2.2. Web Authentication Assertion (interface  
6844 AuthenticatorAssertionResponse) (2) (3)  
6845  
6846 https://heycam.github.io/webidl/#SecureContextReferenced in:  
6847 \* 5.1. PublicKeyCredential Interface  
6848 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
6849 \* 5.2.1. Information about Public Key Credential (interface  
6850 AuthenticatorAttestationResponse)  
6851 \* 5.2.2. Web Authentication Assertion (interface  
6852 AuthenticatorAssertionResponse)  
6853  
6854 https://heycam.github.io/webidl/#securityerrorReferenced in:  
6855 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6856 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6857 \* 5.1.4.1. PublicKeyCredential's  
6858 [[DiscoverFromExternalSource]](origin, options,  
6859 sameOriginWithAncestors) method (2)  
6860 \* 10.1. FIDO AppID Extension (appid)  
6861  
6862 https://heycam.github.io/webidl/#idl-USVStringReferenced in:  
6863 \* 5.4.1. Public Key Entity Description (dictionary  
6864 PublicKeyCredentialEntity) (2)  
6865 \* 5.5. Options for Assertion Generation (dictionary  
6866 PublicKeyCredentialRequestOptions) (2)  
6867 \* 10.1. FIDO AppID Extension (appid)  
6868 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple) (2)  
6869 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
6870 \* 10.5. Supported Extensions Extension (exts)  
6871  
6872 https://heycam.github.io/webidl/#unknownerrorReferenced in:  
6873 \* 6.2.2. The authenticatorMakeCredential operation (2)  
6874 \* 6.2.3. The authenticatorGetAssertion operation (2)  
6875  
6876 https://heycam.github.io/webidl/#idl-booleanReferenced in:  
6877 \* 5.1.7. Availability of User-Verifying Platform Authenticator -  
6878 PublicKeyCredential's  
6879 isUserVerifyingPlatformAuthenticatorAvailable() method  
6880 \* 5.4.4. Authenticator Selection Criteria (dictionary  
6881 AuthenticatorSelectionCriteria) (2)  
6882 \* 10.1. FIDO AppID Extension (appid)  
6883 \* 10.4. Authenticator Selection Extension (authnSel)  
6884 \* 10.5. Supported Extensions Extension (exts)  
6885 \* 10.6. User Verification Index Extension (uvi)  
6886 \* 10.7. Location Extension (loc)  
6887 \* 10.8. User Verification Method Extension (uvm)  
6888  
6889 https://heycam.github.io/webidl/#idl-floatReferenced in:  
6890 \* 10.9. Biometric Authenticator Performance Bounds Extension  
6891 (biometricPerfBounds) (2)  
6892  
6893 https://heycam.github.io/webidl/#dfn-interface-objectReferenced in:

6853 \* 5.1. PublicKeyCredential Interface (2) (3)  
6854 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6855 [[Create]](origin, options, sameOriginWithAncestors) method  
6856  
6857 https://heycam.github.io/webidl/#idl-longReferenced in:  
6858 \* 5.10.5. Cryptographic Algorithm Identifier (typedef  
6859 COSEAlgorithmIdentifier)  
6860  
6861 https://heycam.github.io/webidl/#dfn-presentReferenced in:  
6862 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6863 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6864 (4) (5) (6) (7) (8) (9) (10)  
6865 \* 5.1.4.1. PublicKeyCredential's  
6866 [[DiscoverFromExternalSource]](origin, options,  
6867 sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7)  
6868 \* 5.4.4. Authenticator Selection Criteria (dictionary  
6869 AuthenticatorSelectionCriteria)  
6870  
6871 https://heycam.github.io/webidl/#idl-unsigned-longReferenced in:  
6872 \* 5.4. Options for Credential Creation (dictionary  
6873 PublicKeyCredentialCreationOptions) (2)  
6874 \* 5.5. Options for Assertion Generation (dictionary  
6875 PublicKeyCredentialRequestOptions) (2)  
6876 \* 10.8. User Verification Method Extension (uvm)  
6877  
6878 https://html.spec.whatwg.org/#focusReferenced in:  
6879 \* 5.6. Abort operations with AbortSignal  
6880  
6881 https://html.spec.whatwg.org/#attr-fe-autocomplete-usernameReferenced  
6882 in:  
6883 \* 5.4.1. Public Key Entity Description (dictionary  
6884 PublicKeyCredentialEntity)  
6885  
6886 Terms defined by reference  
6887  
6888 \* [CREDENTIAL-MANAGEMENT-1] defines the following terms:  
6889 + Credential  
6890 + CredentialCreationOptions  
6891 + CredentialRequestOptions  
6892 + CredentialsContainer  
6893 + Request a Credential  
6894 + [[CollectFromCredentialStore]](origin, options,  
6895 sameOriginWithAncestors)  
6896 + [[Create]](origin, options, sameOriginWithAncestors)  
6897 + [[Store]](credential, sameOriginWithAncestors)  
6898 + [[discovery]]  
6899 + [[type]]  
6900 + create()  
6901 + credential  
6902 + credential source  
6903 + get()  
6904 + id  
6905 + remote  
6906 + same-origin with its ancestors  
6907 + signal (for CredentialRequestOptions)  
6908 + store()  
6909 + type  
6910 + user mediation  
6911 \* [DOM4] defines the following terms:  
6912 + AbortController  
6913 + aborted flag  
6914 + document  
6915 \* [ECMAScript] defines the following terms:  
6916 + %arraybuffer%  
6917 + internal method  
6918 + internal slot  
6919 + stringify  
6920 \* [ENCODING] defines the following terms:  
6921 + utf-8 decode  
6922 + utf-8 encode

6894 \* 5.1. PublicKeyCredential Interface (2) (3)  
6895 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6896 [[Create]](origin, options, sameOriginWithAncestors) method  
6897  
6898 https://heycam.github.io/webidl/#idl-longReferenced in:  
6899 \* 5.10.5. Cryptographic Algorithm Identifier (typedef  
6900 COSEAlgorithmIdentifier)  
6901  
6902 https://heycam.github.io/webidl/#dfn-presentReferenced in:  
6903 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6904 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6905 (4) (5) (6) (7) (8) (9) (10)  
6906 \* 5.1.4.1. PublicKeyCredential's  
6907 [[DiscoverFromExternalSource]](origin, options,  
6908 sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7)  
6909 \* 5.4.4. Authenticator Selection Criteria (dictionary  
6910 AuthenticatorSelectionCriteria)  
6911  
6912 https://heycam.github.io/webidl/#idl-unsigned-longReferenced in:  
6913 \* 5.4. Options for Credential Creation (dictionary  
6914 PublicKeyCredentialCreationOptions) (2)  
6915 \* 5.5. Options for Assertion Generation (dictionary  
6916 PublicKeyCredentialRequestOptions) (2)  
6917 \* 10.8. User Verification Method Extension (uvm)  
6918  
6919 https://html.spec.whatwg.org/#focusReferenced in:  
6920 \* 5.6. Abort operations with AbortSignal  
6921  
6922 https://html.spec.whatwg.org/#attr-fe-autocomplete-usernameReferenced  
6923 in:  
6924 \* 5.4.1. Public Key Entity Description (dictionary  
6925 PublicKeyCredentialEntity)  
6926  
6927 Terms defined by reference  
6928  
6929 \* [CREDENTIAL-MANAGEMENT-1] defines the following terms:  
6930 + Credential  
6931 + CredentialCreationOptions  
6932 + CredentialRequestOptions  
6933 + CredentialsContainer  
6934 + Request a Credential  
6935 + [[CollectFromCredentialStore]](origin, options,  
6936 sameOriginWithAncestors)  
6937 + [[Create]](origin, options, sameOriginWithAncestors)  
6938 + [[Store]](credential, sameOriginWithAncestors)  
6939 + [[discovery]]  
6940 + [[type]]  
6941 + create()  
6942 + credential  
6943 + credential source  
6944 + get()  
6945 + id  
6946 + remote  
6947 + same-origin with its ancestors  
6948 + signal (for CredentialRequestOptions)  
6949 + store()  
6950 + type  
6951 + user mediation  
6952 \* [DOM4] defines the following terms:  
6953 + AbortController  
6954 + aborted flag  
6955 + document  
6956 \* [ECMAScript] defines the following terms:  
6957 + %arraybuffer%  
6958 + internal method  
6959 + internal slot  
6960 + stringify  
6961 \* [ENCODING] defines the following terms:  
6962 + utf-8 decode  
6963 + utf-8 encode

6923 \* [FETCH] defines the following terms:  
 6924 + window  
 6925 \* [FIDO-APPID] defines the following terms:  
 6926 + determining if a caller's facetid is authorized for an appid  
 6927 + determining the facetid of a calling application  
 6928 \* [FIDO-CTAP] defines the following terms:  
 6929 + ctap2 canonical cbor encoding form  
 6930 \* [FIDO-Registry] defines the following terms:  
 6931 + section 3.1 user verification methods  
 6932 + section 3.2 key protection types  
 6933 + section 3.3 matcher protection types  
 6934 + section 3.6.2 public key representation formats  
 6935 \* [FIDO-U2F-Message-Formats] defines the following terms:  
 6936 + application parameter  
 6937 + section 4.3  
 6938 + section 5.4  
 6939 \* [Geolocation-API] defines the following terms:  
 6940 + Coordinates  
 6941 \* [HTML] defines the following terms:  
 6942 + ascii serialization of an origin  
 6943 + effective domain  
 6944 + environment settings object  
 6945 + global object  
 6946 + is a registrable domain suffix of or is equal to  
 6947 + is not a registrable domain suffix of and is not equal to  
 6948 + origin  
 6949 + relevant settings object  
 6950 \* [HTML52] defines the following terms:  
 6951 + document.domain  
 6952 + opaque origin  
 6953 + origin  
 6954 \* [INFRA] defines the following terms:  
 6955 + append (for set)  
 6956 + byte sequence  
 6957 + continue  
 6958 + for each (for map)  
 6959 + is empty  
 6960 + is not empty  
 6961 + item (for struct)  
 6962 + list  
 6963 + map  
 6964 + ordered set  
 6965 + remove  
 6966 + set (for map)  
 6967 + struct  
 6968 + while  
 6969 + willful violation  
 6970 \* [mixed-content] defines the following terms:  
 6971 + a priori authenticated url  
 6972 \* [page-visibility] defines the following terms:  
 6973 + visibility states  
 6974 \* [RFC4949] defines the following terms:  
 6975 + leap of faith  
 6976 + man-in-the-middle attack  
 6977 \* [RFC8152] defines the following terms:  
 6978 + section 7  
 6979 \* [secure-contexts] defines the following terms:  
 6980 + secure contexts  
 6981 \* [TokenBinding] defines the following terms:  
 6982 + token binding  
 6983 + token binding id  
 6984 \* [URL] defines the following terms:  
 6985 + domain  
 6986 + empty host  
 6987 + host  
 6988 + ipv4 address  
 6989 + ipv6 address  
 6990 + opaque host  
 6991 + url serializer  
 6992 + valid domain

6964 \* [FETCH] defines the following terms:  
 6965 + window  
 6966 \* [FIDO-APPID] defines the following terms:  
 6967 + determining if a caller's facetid is authorized for an appid  
 6968 + determining the facetid of a calling application  
 6969 \* [FIDO-CTAP] defines the following terms:  
 6970 + ctap2 canonical cbor encoding form  
 6971 \* [FIDO-Registry] defines the following terms:  
 6972 + section 3.1 user verification methods  
 6973 + section 3.2 key protection types  
 6974 + section 3.3 matcher protection types  
 6975 + section 3.6.2 public key representation formats  
 6976 \* [FIDO-U2F-Message-Formats] defines the following terms:  
 6977 + application parameter  
 6978 + section 4.3  
 6979 + section 5.4  
 6980 \* [Geolocation-API] defines the following terms:  
 6981 + Coordinates  
 6982 \* [HTML] defines the following terms:  
 6983 + ascii serialization of an origin  
 6984 + effective domain  
 6985 + environment settings object  
 6986 + global object  
 6987 + is a registrable domain suffix of or is equal to  
 6988 + is not a registrable domain suffix of and is not equal to  
 6989 + origin  
 6990 + relevant settings object  
 6991 \* [HTML52] defines the following terms:  
 6992 + document.domain  
 6993 + opaque origin  
 6994 + origin  
 6995 \* [INFRA] defines the following terms:  
 6996 + append (for set)  
 6997 + byte sequence  
 6998 + continue  
 6999 + for each (for map)  
 7000 + is empty  
 7001 + is not empty  
 7002 + item (for struct)  
 7003 + list  
 7004 + map  
 7005 + ordered set  
 7006 + remove  
 7007 + set (for map)  
 7008 + struct  
 7009 + while  
 7010 + willful violation  
 7011 \* [mixed-content] defines the following terms:  
 7012 + a priori authenticated url  
 7013 \* [page-visibility] defines the following terms:  
 7014 + visibility states  
 7015 \* [RFC4949] defines the following terms:  
 7016 + leap of faith  
 7017 + man-in-the-middle attack  
 7018 \* [RFC8152] defines the following terms:  
 7019 + section 7  
 7020 \* [secure-contexts] defines the following terms:  
 7021 + secure contexts  
 7022 \* [TokenBinding] defines the following terms:  
 7023 + token binding  
 7024 + token binding id  
 7025 \* [URL] defines the following terms:  
 7026 + domain  
 7027 + empty host  
 7028 + host  
 7029 + ipv4 address  
 7030 + ipv6 address  
 7031 + opaque host  
 7032 + url serializer  
 7033 + valid domain

6993 + valid domain string  
6994 \* [WebIDL] defines the following terms:  
6995 + AbortError  
6996 + ArrayBuffer  
6997 + BufferSource  
6998 + ConstraintError  
6999 + DOMException  
7000 + DOMString  
7001 + Exposed  
7002 + InvalidStateError  
7003 + NotAllowedError  
7004 + NotSupportedError  
7005 + Promise  
7006 + SameObject  
7007 + SecureContext  
7008 + SecurityError  
7009 + USVString  
7010 + UnknownError  
7011 + boolean  
7012 + float  
7013 + interface object  
7014 + long  
7015 + present  
7016 + unsigned long  
7017 \* [whatwg html] defines the following terms:  
7018 + focus  
7019 + username

References

Normative References

[CDDL]

C. Viano; H. Birkholz. CBOR data definition language (CDDL): a notational convention to express CBOR data structures. 21 September 2016. Internet Draft (work in progress). URL: <https://tools.ietf.org/html/draft-greevenbosch-appsawg-cbor-cddl>

[CREDENTIAL-MANAGEMENT-1]

Mike West. Credential Management Level 1. 4 August 2017. WD. URL: <https://www.w3.org/TR/credential-management-1/>

[DOM4]

Anne van Kesteren. DOM Standard. Living Standard. URL: <https://dom.spec.whatwg.org/>

[ECMAScript]

ECMAScript Language Specification. URL: <https://tc39.github.io/ecma262/>

[ENCODING]

Anne van Kesteren. Encoding Standard. Living Standard. URL: <https://encoding.spec.whatwg.org/>

[FETCH]

Anne van Kesteren. Fetch Standard. Living Standard. URL: <https://fetch.spec.whatwg.org/>

[FIDO-APPID]

D. Balfanz; et al. FIDO AppID and Facets. FIDO Alliance Proposed Standard. URL: <https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-appid-and-facets-v2.0-ps-20170927.html>

[FIDO-CTAP]

R. Lindemann; et al. FIDO 2.0: Client to Authenticator Protocol. FIDO Alliance Proposed Standard. URL: <https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-client-to-authenticator-protocol-v2.0-ps-20170927.html>

7034 + valid domain string  
7035 \* [WebIDL] defines the following terms:  
7036 + AbortError  
7037 + ArrayBuffer  
7038 + BufferSource  
7039 + ConstraintError  
7040 + DOMException  
7041 + DOMString  
7042 + Exposed  
7043 + InvalidStateError  
7044 + NotAllowedError  
7045 + NotSupportedError  
7046 + Promise  
7047 + SameObject  
7048 + SecureContext  
7049 + SecurityError  
7050 + USVString  
7051 + UnknownError  
7052 + boolean  
7053 + float  
7054 + interface object  
7055 + long  
7056 + present  
7057 + unsigned long  
7058 \* [whatwg html] defines the following terms:  
7059 + focus  
7060 + username

References

Normative References

[CDDL]

C. Viano; H. Birkholz. CBOR data definition language (CDDL): a notational convention to express CBOR data structures. 21 September 2016. Internet Draft (work in progress). URL: <https://tools.ietf.org/html/draft-greevenbosch-appsawg-cbor-cddl>

[CREDENTIAL-MANAGEMENT-1]

Mike West. Credential Management Level 1. 4 August 2017. WD. URL: <https://www.w3.org/TR/credential-management-1/>

[DOM4]

Anne van Kesteren. DOM Standard. Living Standard. URL: <https://dom.spec.whatwg.org/>

[ECMAScript]

ECMAScript Language Specification. URL: <https://tc39.github.io/ecma262/>

[ENCODING]

Anne van Kesteren. Encoding Standard. Living Standard. URL: <https://encoding.spec.whatwg.org/>

[FETCH]

Anne van Kesteren. Fetch Standard. Living Standard. URL: <https://fetch.spec.whatwg.org/>

[FIDO-APPID]

D. Balfanz; et al. FIDO AppID and Facets. FIDO Alliance Proposed Standard. URL: <https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-appid-and-facets-v2.0-ps-20170927.html>

[FIDO-CTAP]

R. Lindemann; et al. FIDO 2.0: Client to Authenticator Protocol. FIDO Alliance Proposed Standard. URL: <https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-client-to-authenticator-protocol-v2.0-ps-20170927.html>

7063 [FIDO-Privacy-Principles]  
7064 FIDO Alliance. FIDO Privacy Principles. FIDO Alliance  
7065 Whitepaper. URL:  
7066 [https://fidoalliance.org/wp-content/uploads/2014/12/FIDO\\_Alliance\\_Whitepaper\\_Privacy\\_Principles.pdf](https://fidoalliance.org/wp-content/uploads/2014/12/FIDO_Alliance_Whitepaper_Privacy_Principles.pdf)  
7067  
7068  
7069 [FIDO-Registry]  
7070 R. Lindemann. FIDO Registry of Predefined Values. 27 September  
7071 2017. FIDO Alliance Proposed Standard. URL:  
7072 <https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0-ps-20170927.html>  
7073  
7074  
7075 [FIDO-U2F-Message-Formats]  
7076 D. Balfanz; J. Ehrensvar; J. Lang. FIDO U2F Raw Message  
7077 Formats. FIDO Alliance Implementation Draft. URL:  
7078 <https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-message-formats-v1.1-id-20160915.html>  
7079  
7080  
7081 [FIDOEcdaaAlgorithm]  
7082 R. Lindemann; et al. FIDO ECDA Algorithm. FIDO Alliance  
7083 Implementation Draft. URL:  
7084 <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ecdaa-algorithm-v1.1-id-20170202.html>  
7085  
7086  
7087 [Geolocation-API]  
7088 Andrei Popescu. Geolocation API Specification 2nd Edition. 8  
7089 November 2016. REC. URL: <https://www.w3.org/TR/geolocation-API/>  
7090  
7091 [HTML]  
7092 Anne van Kesteren; et al. HTML Standard. Living Standard. URL:  
7093 <https://html.spec.whatwg.org/multipage/>  
7094  
7095 [HTML52]  
7096 Steve Faulkner; et al. HTML 5.2. 14 December 2017. REC. URL:  
7097 <https://www.w3.org/TR/html52/>  
7098  
7099 [IANA-COSE-ALGS-REG]  
7100 IANA CBOR Object Signing and Encryption (COSE) Algorithms  
7101 Registry. URL:  
7102 <https://www.iana.org/assignments/cose/cose.xhtml#algorithms>  
7103  
7104 [INFRA]  
7105 Anne van Kesteren; Domenic Denicola. Infra Standard. Living  
7106 Standard. URL: <https://infra.spec.whatwg.org/>  
7107  
7108 [MIXED-CONTENT]  
7109 Mike West. Mixed Content. 2 August 2016. CR. URL:  
7110 <https://www.w3.org/TR/mixed-content/>  
7111  
7112 [PAGE-VISIBILITY]  
7113 Jatinder Mann; Arvind Jain. Page Visibility (Second Edition). 29  
7114 October 2013. REC. URL: <https://www.w3.org/TR/page-visibility/>  
7115  
7116 [RFC2119]  
7117 S. Bradner. Key words for use in RFCs to Indicate Requirement  
7118 Levels. March 1997. Best Current Practice. URL:  
7119 <https://tools.ietf.org/html/rfc2119>  
7120  
7121 [RFC4648]  
7122 S. Josefsson. The Base16, Base32, and Base64 Data Encodings.  
7123 October 2006. Proposed Standard. URL:  
7124 <https://tools.ietf.org/html/rfc4648>  
7125  
7126 [RFC4949]  
7127 R. Shirey. Internet Security Glossary, Version 2. August 2007.  
7128 Informational. URL: <https://tools.ietf.org/html/rfc4949>  
7129  
7130 [RFC5234]  
7131 D. Crocker, Ed.; P. Overell. Augmented BNF for Syntax  
7132 Specifications: ABNF. January 2008. Internet Standard. URL:

7104 [FIDO-Privacy-Principles]  
7105 FIDO Alliance. FIDO Privacy Principles. FIDO Alliance  
7106 Whitepaper. URL:  
7107 [https://fidoalliance.org/wp-content/uploads/2014/12/FIDO\\_Alliance\\_Whitepaper\\_Privacy\\_Principles.pdf](https://fidoalliance.org/wp-content/uploads/2014/12/FIDO_Alliance_Whitepaper_Privacy_Principles.pdf)  
7108  
7109  
7110 [FIDO-Registry]  
7111 R. Lindemann. FIDO Registry of Predefined Values. 27 September  
7112 2017. FIDO Alliance Proposed Standard. URL:  
7113 <https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-registry-v2.0-ps-20170927.html>  
7114  
7115  
7116 [FIDO-U2F-Message-Formats]  
7117 D. Balfanz; J. Ehrensvar; J. Lang. FIDO U2F Raw Message  
7118 Formats. FIDO Alliance Implementation Draft. URL:  
7119 <https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-message-formats-v1.1-id-20160915.html>  
7120  
7121  
7122 [FIDOEcdaaAlgorithm]  
7123 R. Lindemann; et al. FIDO ECDA Algorithm. FIDO Alliance  
7124 Implementation Draft. URL:  
7125 <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ecdaa-algorithm-v1.1-id-20170202.html>  
7126  
7127  
7128 [Geolocation-API]  
7129 Andrei Popescu. Geolocation API Specification 2nd Edition. 8  
7130 November 2016. REC. URL: <https://www.w3.org/TR/geolocation-API/>  
7131  
7132 [HTML]  
7133 Anne van Kesteren; et al. HTML Standard. Living Standard. URL:  
7134 <https://html.spec.whatwg.org/multipage/>  
7135  
7136 [HTML52]  
7137 Steve Faulkner; et al. HTML 5.2. 14 December 2017. REC. URL:  
7138 <https://www.w3.org/TR/html52/>  
7139  
7140 [IANA-COSE-ALGS-REG]  
7141 IANA CBOR Object Signing and Encryption (COSE) Algorithms  
7142 Registry. URL:  
7143 <https://www.iana.org/assignments/cose/cose.xhtml#algorithms>  
7144  
7145 [INFRA]  
7146 Anne van Kesteren; Domenic Denicola. Infra Standard. Living  
7147 Standard. URL: <https://infra.spec.whatwg.org/>  
7148  
7149 [MIXED-CONTENT]  
7150 Mike West. Mixed Content. 2 August 2016. CR. URL:  
7151 <https://www.w3.org/TR/mixed-content/>  
7152  
7153 [PAGE-VISIBILITY]  
7154 Jatinder Mann; Arvind Jain. Page Visibility (Second Edition). 29  
7155 October 2013. REC. URL: <https://www.w3.org/TR/page-visibility/>  
7156  
7157 [RFC2119]  
7158 S. Bradner. Key words for use in RFCs to Indicate Requirement  
7159 Levels. March 1997. Best Current Practice. URL:  
7160 <https://tools.ietf.org/html/rfc2119>  
7161  
7162 [RFC4648]  
7163 S. Josefsson. The Base16, Base32, and Base64 Data Encodings.  
7164 October 2006. Proposed Standard. URL:  
7165 <https://tools.ietf.org/html/rfc4648>  
7166  
7167 [RFC4949]  
7168 R. Shirey. Internet Security Glossary, Version 2. August 2007.  
7169 Informational. URL: <https://tools.ietf.org/html/rfc4949>  
7170  
7171 [RFC5234]  
7172 D. Crocker, Ed.; P. Overell. Augmented BNF for Syntax  
7173 Specifications: ABNF. January 2008. Internet Standard. URL:

7133 https://tools.ietf.org/html/rfc5234  
7134  
7135 [RFC5890]  
7136 J. Klensin. Internationalized Domain Names for Applications  
7137 (IDNA): Definitions and Document Framework. August 2010.  
7138 Proposed Standard. URL: https://tools.ietf.org/html/rfc5890  
7139  
7140 [RFC7049]  
7141 C. Bormann; P. Hoffman. Concise Binary Object Representation  
7142 (CBOR). October 2013. Proposed Standard. URL:  
7143 https://tools.ietf.org/html/rfc7049  
7144  
7145 [RFC8152]  
7146 J. Schaad. CBOR Object Signing and Encryption (COSE). July 2017.  
7147 Proposed Standard. URL: https://tools.ietf.org/html/rfc8152  
7148  
7149 [RFC8230]  
7150 M. Jones. Using RSA Algorithms with CBOR Object Signing and  
7151 Encryption (COSE) Messages. September 2017. Proposed Standard.  
7152 URL: https://tools.ietf.org/html/rfc8230  
7153  
7154 [SEC1]  
7155 SEC1: Elliptic Curve Cryptography, Version 2.0. URL:  
7156 http://www.secg.org/sec1-v2.pdf  
7157  
7158 [SECURE-CONTEXTS]  
7159 Mike West. Secure Contexts. 15 September 2016. CR. URL:  
7160 https://www.w3.org/TR/secure-contexts/  
7161  
7162 [TCG-CMCPProfile-AIKCertEnroll]  
7163 Scott Kelly; et al. TCG Infrastructure Working Group: A CMC  
7164 Profile for AIK Certificate Enrollment. 24 March 2011.  
7165 Published. URL:  
7166 https://trustedcomputinggroup.org/wp-content/uploads/IWG\_CMC\_Pro  
7167 file\_Cert\_Enrollment\_v1\_r7.pdf  
7168  
7169 [TokenBinding]  
7170 A. Popov; et al. The Token Binding Protocol Version 1.0.  
7171 February 16, 2017. Internet-Draft. URL:  
7172 https://tools.ietf.org/html/draft-ietf-tokbind-protocol  
7173  
7174 [URL]  
7175 Anne van Kesteren. URL Standard. Living Standard. URL:  
7176 https://url.spec.whatwg.org/  
7177  
7178 [WebAuthn-COSE-Algs]  
7179 Michael B. Jones. COSE Algorithms for Web Authentication  
7180 (WebAuthn). May 2018. Active Internet-Draft. URL:  
7181 https://tools.ietf.org/html/draft-jones-webauthn-cose-algorithms  
7182  
7183 [WebAuthn-Registries]  
7184 Jeff Hodges; Giridhar Mandyam; Michael B. Jones. Registries for  
7185 Web Authentication (WebAuthn). February 2018. Active  
7186 Internet-Draft. URL:  
7187 https://tools.ietf.org/html/draft-hodges-webauthn-registries  
7188  
7189 [WebIDL]  
7190 Cameron McCormack; Boris Zbarsky; Tobie Langel. Web IDL. 15  
7191 December 2016. ED. URL: https://heycam.github.io/webidl/  
7192  
7193 [WebIDL-1]  
7194 Cameron McCormack. WebIDL Level 1. 15 December 2016. REC. URL:  
7195 https://www.w3.org/TR/2016/REC-WebIDL-1-20161215/  
7196  
7197 Informative References  
7198  
7199 [Ceremony]  
7200 Carl Ellison. Ceremony Design and Analysis. 2007. URL:  
7201 https://eprint.iacr.org/2007/399.pdf  
7202

7174 https://tools.ietf.org/html/rfc5234  
7175  
7176 [RFC5890]  
7177 J. Klensin. Internationalized Domain Names for Applications  
7178 (IDNA): Definitions and Document Framework. August 2010.  
7179 Proposed Standard. URL: https://tools.ietf.org/html/rfc5890  
7180  
7181 [RFC7049]  
7182 C. Bormann; P. Hoffman. Concise Binary Object Representation  
7183 (CBOR). October 2013. Proposed Standard. URL:  
7184 https://tools.ietf.org/html/rfc7049  
7185  
7186 [RFC8152]  
7187 J. Schaad. CBOR Object Signing and Encryption (COSE). July 2017.  
7188 Proposed Standard. URL: https://tools.ietf.org/html/rfc8152  
7189  
7190 [RFC8230]  
7191 M. Jones. Using RSA Algorithms with CBOR Object Signing and  
7192 Encryption (COSE) Messages. September 2017. Proposed Standard.  
7193 URL: https://tools.ietf.org/html/rfc8230  
7194  
7195 [SEC1]  
7196 SEC1: Elliptic Curve Cryptography, Version 2.0. URL:  
7197 http://www.secg.org/sec1-v2.pdf  
7198  
7199 [SECURE-CONTEXTS]  
7200 Mike West. Secure Contexts. 15 September 2016. CR. URL:  
7201 https://www.w3.org/TR/secure-contexts/  
7202  
7203 [TCG-CMCPProfile-AIKCertEnroll]  
7204 Scott Kelly; et al. TCG Infrastructure Working Group: A CMC  
7205 Profile for AIK Certificate Enrollment. 24 March 2011.  
7206 Published. URL:  
7207 https://trustedcomputinggroup.org/wp-content/uploads/IWG\_CMC\_Pro  
7208 file\_Cert\_Enrollment\_v1\_r7.pdf  
7209  
7210 [TokenBinding]  
7211 A. Popov; et al. The Token Binding Protocol Version 1.0.  
7212 February 16, 2017. Internet-Draft. URL:  
7213 https://tools.ietf.org/html/draft-ietf-tokbind-protocol  
7214  
7215 [URL]  
7216 Anne van Kesteren. URL Standard. Living Standard. URL:  
7217 https://url.spec.whatwg.org/  
7218  
7219 [WebAuthn-COSE-Algs]  
7220 Michael B. Jones. COSE Algorithms for Web Authentication  
7221 (WebAuthn). May 2018. Active Internet-Draft. URL:  
7222 https://tools.ietf.org/html/draft-jones-webauthn-cose-algorithms  
7223  
7224 [WebAuthn-Registries]  
7225 Jeff Hodges; Giridhar Mandyam; Michael B. Jones. Registries for  
7226 Web Authentication (WebAuthn). February 2018. Active  
7227 Internet-Draft. URL:  
7228 https://tools.ietf.org/html/draft-hodges-webauthn-registries  
7229  
7230 [WebIDL]  
7231 Cameron McCormack; Boris Zbarsky; Tobie Langel. Web IDL. 15  
7232 December 2016. ED. URL: https://heycam.github.io/webidl/  
7233  
7234 [WebIDL-1]  
7235 Cameron McCormack. WebIDL Level 1. 15 December 2016. REC. URL:  
7236 https://www.w3.org/TR/2016/REC-WebIDL-1-20161215/  
7237  
7238 Informative References  
7239  
7240 [Ceremony]  
7241 Carl Ellison. Ceremony Design and Analysis. 2007. URL:  
7242 https://eprint.iacr.org/2007/399.pdf  
7243



7203 [EduPersonObjectClassSpec]  
7204 EduPerson Object Class Specification (200604a). May 15, 2007.  
7205 URL:  
7206 <https://www.internet2.edu/media/medialibrary/2013/09/04/internet>  
7207 [2-mace-dir-eduperson-200604.html](https://www.internet2.edu/media/medialibrary/2013/09/04/internet)  
7208  
7209 [Feature-Policy]  
7210 Feature Policy. Draft Community Group Report. URL:  
7211 <https://wicg.github.io/feature-policy/>  
7212  
7213 [FIDO-UAF-AUTHNR-CMDS]  
7214 R. Lindemann; J. Kemp. FIDO UAF Authenticator Commands. FIDO  
7215 Alliance Implementation Draft. URL:  
7216 <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ua>  
7217 [f-authnr-cmds-v1.1-id-20170202.html](https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ua)  
7218  
7219 [FIDOAuthnrSecReqs]  
7220 D. Biggs; et al. FIDO Authenticator Security Requirements. FIDO  
7221 Alliance Final Documents. URL:  
7222 <https://fidoalliance.org/specs/fido-security-requirements-v1.0-f>  
7223 [d-20170524/](https://fidoalliance.org/specs/fido-security-requirements-v1.0-f)  
7224  
7225 [FIDOMetadataService]  
7226 R. Lindemann; B. Hill; D. Baghdasaryan. FIDO Metadata Service  
7227 v1.0. FIDO Alliance Proposed Standard. URL:  
7228 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua>  
7229 [f-metadata-service-v1.0-ps-20141208.html](https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua)  
7230  
7231 [FIDOSecRef]  
7232 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO Security Reference.  
7233 FIDO Alliance Proposed Standard. URL:  
7234 <https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-se>  
7235 [curity-ref-v1.2-ps-20170411.html](https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-se)  
7236  
7237 [FIDOUAFAuthenticatorMetadataStatements]  
7238 B. Hill; D. Baghdasaryan; J. Kemp. FIDO UAF Authenticator  
7239 Metadata Statements v1.0. FIDO Alliance Proposed Standard. URL:  
7240 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua>  
7241 [f-authnr-metadata-v1.0-ps-20141208.html](https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua)  
7242  
7243 [ISOBiometricVocabulary]  
7244 ISO/IEC JTC1/SC37. Information technology -- Vocabulary --  
7245 Biometrics. 15 December 2012. International Standard: ISO/IEC  
7246 2382-37:2012(E) First Edition. URL:  
7247 <http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194>  
7248 [\\_ISOIEC\\_2382-37\\_2012.zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194)  
7249  
7250 [RFC3279]  
7251 L. Bassham; W. Polk; R. Housley. Algorithms and Identifiers for  
7252 the Internet X.509 Public Key Infrastructure Certificate and  
7253 Certificate Revocation List (CRL) Profile. April 2002. Proposed  
7254 Standard. URL: <https://tools.ietf.org/html/rfc3279>  
7255  
7256 [RFC5280]  
7257 D. Cooper; et al. Internet X.509 Public Key Infrastructure  
7258 Certificate and Certificate Revocation List (CRL) Profile. May  
7259 2008. Proposed Standard. URL:  
7260 <https://tools.ietf.org/html/rfc5280>  
7261  
7262 [RFC6265]  
7263 A. Barth. HTTP State Management Mechanism. April 2011. Proposed  
7264 Standard. URL: <https://tools.ietf.org/html/rfc6265>  
7265  
7266 [RFC6454]  
7267 A. Barth. The Web Origin Concept. December 2011. Proposed  
7268 Standard. URL: <https://tools.ietf.org/html/rfc6454>  
7269  
7270 [RFC7515]  
7271 M. Jones; J. Bradley; N. Sakimura. JSON Web Signature (JWS). May  
7272 2015. Proposed Standard. URL:

7244 [EduPersonObjectClassSpec]  
7245 EduPerson Object Class Specification (200604a). May 15, 2007.  
7246 URL:  
7247 <https://www.internet2.edu/media/medialibrary/2013/09/04/internet>  
7248 [2-mace-dir-eduperson-200604.html](https://www.internet2.edu/media/medialibrary/2013/09/04/internet)  
7249  
7250 [Feature-Policy]  
7251 Feature Policy. Draft Community Group Report. URL:  
7252 <https://wicg.github.io/feature-policy/>  
7253  
7254 [FIDO-UAF-AUTHNR-CMDS]  
7255 R. Lindemann; J. Kemp. FIDO UAF Authenticator Commands. FIDO  
7256 Alliance Implementation Draft. URL:  
7257 <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ua>  
7258 [f-authnr-cmds-v1.1-id-20170202.html](https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ua)  
7259  
7260 [FIDOAuthnrSecReqs]  
7261 D. Biggs; et al. FIDO Authenticator Security Requirements. FIDO  
7262 Alliance Final Documents. URL:  
7263 <https://fidoalliance.org/specs/fido-security-requirements-v1.0-f>  
7264 [d-20170524/](https://fidoalliance.org/specs/fido-security-requirements-v1.0-f)  
7265  
7266 [FIDOMetadataService]  
7267 R. Lindemann; B. Hill; D. Baghdasaryan. FIDO Metadata Service  
7268 v1.0. FIDO Alliance Proposed Standard. URL:  
7269 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua>  
7270 [f-metadata-service-v1.0-ps-20141208.html](https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua)  
7271  
7272 [FIDOSecRef]  
7273 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO Security Reference.  
7274 FIDO Alliance Proposed Standard. URL:  
7275 <https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-se>  
7276 [curity-ref-v1.2-ps-20170411.html](https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-se)  
7277  
7278 [FIDOUAFAuthenticatorMetadataStatements]  
7279 B. Hill; D. Baghdasaryan; J. Kemp. FIDO UAF Authenticator  
7280 Metadata Statements v1.0. FIDO Alliance Proposed Standard. URL:  
7281 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua>  
7282 [f-authnr-metadata-v1.0-ps-20141208.html](https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua)  
7283  
7284 [ISOBiometricVocabulary]  
7285 ISO/IEC JTC1/SC37. Information technology -- Vocabulary --  
7286 Biometrics. 15 December 2012. International Standard: ISO/IEC  
7287 2382-37:2012(E) First Edition. URL:  
7288 <http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194>  
7289 [\\_ISOIEC\\_2382-37\\_2012.zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194)  
7290  
7291 [RFC3279]  
7292 L. Bassham; W. Polk; R. Housley. Algorithms and Identifiers for  
7293 the Internet X.509 Public Key Infrastructure Certificate and  
7294 Certificate Revocation List (CRL) Profile. April 2002. Proposed  
7295 Standard. URL: <https://tools.ietf.org/html/rfc3279>  
7296  
7297 [RFC5280]  
7298 D. Cooper; et al. Internet X.509 Public Key Infrastructure  
7299 Certificate and Certificate Revocation List (CRL) Profile. May  
7300 2008. Proposed Standard. URL:  
7301 <https://tools.ietf.org/html/rfc5280>  
7302  
7303 [RFC6265]  
7304 A. Barth. HTTP State Management Mechanism. April 2011. Proposed  
7305 Standard. URL: <https://tools.ietf.org/html/rfc6265>  
7306  
7307 [RFC6454]  
7308 A. Barth. The Web Origin Concept. December 2011. Proposed  
7309 Standard. URL: <https://tools.ietf.org/html/rfc6454>  
7310  
7311 [RFC7515]  
7312 M. Jones; J. Bradley; N. Sakimura. JSON Web Signature (JWS). May  
7313 2015. Proposed Standard. URL:

7273 https://tools.ietf.org/html/rfc7515  
7274  
7275 [RFC8017]  
7276 K. Moriarty, Ed.; et al. PKCS #1: RSA Cryptography  
7277 Specifications Version 2.2. November 2016. Informational. URL:  
7278 https://tools.ietf.org/html/rfc8017  
7279  
7280 [TPMv2-EK-Profile]  
7281 TCG EK Credential Profile for TPM Family 2.0. URL:  
7282 http://www.trustedcomputinggroup.org/wp-content/uploads/Credenti  
7283 al\_Profile\_EK\_V2.0\_R14\_published.pdf  
7284  
7285 [TPMv2-Part1]  
7286 Trusted Platform Module Library, Part 1: Architecture. URL:  
7287 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-  
7288 2.0-Part-1-Architecture-01.38.pdf  
7289  
7290 [TPMv2-Part2]  
7291 Trusted Platform Module Library, Part 2: Structures. URL:  
7292 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-  
7293 2.0-Part-2-Structures-01.38.pdf  
7294  
7295 [TPMv2-Part3]  
7296 Trusted Platform Module Library, Part 3: Commands. URL:  
7297 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-  
7298 2.0-Part-3-Commands-01.38.pdf  
7299  
7300 [UAFProtocol]  
7301 R. Lindemann; et al. FIDO UAF Protocol Specification v1.0. FIDO  
7302 Alliance Proposed Standard. URL:  
7303 https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua  
7304 f-protocol-v1.0-ps-20141208.html  
7305  
7306 IDL Index  
7307  
7308 [SecureContext, Exposed=Window]  
7309 interface PublicKeyCredential : Credential {  
7310 [SameObject] readonly attribute ArrayBuffer rawId;  
7311 [SameObject] readonly attribute AuthenticatorResponse response;  
7312 AuthenticationExtensionsClientOutputs getClientExtensionResults();  
7313 };  
7314  
7315 partial dictionary CredentialCreationOptions {  
7316 PublicKeyCredentialCreationOptions publicKey;  
7317 };  
7318  
7319 partial dictionary CredentialRequestOptions {  
7320 PublicKeyCredentialRequestOptions publicKey;  
7321 };  
7322  
7323 partial interface PublicKeyCredential {  
7324 static Promise < boolean > isUserVerifyingPlatformAuthenticatorAvailable();  
7325 };  
7326  
7327 [SecureContext, Exposed=Window]  
7328 interface AuthenticatorResponse {  
7329 [SameObject] readonly attribute ArrayBuffer clientDataJSON;  
7330 };  
7331  
7332 [SecureContext, Exposed=Window]  
7333 interface AuthenticatorAttestationResponse : AuthenticatorResponse {  
7334 [SameObject] readonly attribute ArrayBuffer attestationObject;  
7335 };  
7336  
7337 [SecureContext, Exposed=Window]  
7338 interface AuthenticatorAssertionResponse : AuthenticatorResponse {  
7339 [SameObject] readonly attribute ArrayBuffer authenticatorData;  
7340 [SameObject] readonly attribute ArrayBuffer signature;  
7341 [SameObject] readonly attribute ArrayBuffer? userHandle;  
7342 };

7314 https://tools.ietf.org/html/rfc7515  
7315  
7316 [RFC8017]  
7317 K. Moriarty, Ed.; et al. PKCS #1: RSA Cryptography  
7318 Specifications Version 2.2. November 2016. Informational. URL:  
7319 https://tools.ietf.org/html/rfc8017  
7320  
7321 [TPMv2-EK-Profile]  
7322 TCG EK Credential Profile for TPM Family 2.0. URL:  
7323 http://www.trustedcomputinggroup.org/wp-content/uploads/Credenti  
7324 al\_Profile\_EK\_V2.0\_R14\_published.pdf  
7325  
7326 [TPMv2-Part1]  
7327 Trusted Platform Module Library, Part 1: Architecture. URL:  
7328 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-  
7329 2.0-Part-1-Architecture-01.38.pdf  
7330  
7331 [TPMv2-Part2]  
7332 Trusted Platform Module Library, Part 2: Structures. URL:  
7333 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-  
7334 2.0-Part-2-Structures-01.38.pdf  
7335  
7336 [TPMv2-Part3]  
7337 Trusted Platform Module Library, Part 3: Commands. URL:  
7338 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-  
7339 2.0-Part-3-Commands-01.38.pdf  
7340  
7341 [UAFProtocol]  
7342 R. Lindemann; et al. FIDO UAF Protocol Specification v1.0. FIDO  
7343 Alliance Proposed Standard. URL:  
7344 https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua  
7345 f-protocol-v1.0-ps-20141208.html  
7346  
7347 IDL Index  
7348  
7349 [SecureContext, Exposed=Window]  
7350 interface PublicKeyCredential : Credential {  
7351 [SameObject] readonly attribute ArrayBuffer rawId;  
7352 [SameObject] readonly attribute AuthenticatorResponse response;  
7353 AuthenticationExtensionsClientOutputs getClientExtensionResults();  
7354 };  
7355  
7356 partial dictionary CredentialCreationOptions {  
7357 PublicKeyCredentialCreationOptions publicKey;  
7358 };  
7359  
7360 partial dictionary CredentialRequestOptions {  
7361 PublicKeyCredentialRequestOptions publicKey;  
7362 };  
7363  
7364 partial interface PublicKeyCredential {  
7365 static Promise < boolean > isUserVerifyingPlatformAuthenticatorAvailable();  
7366 };  
7367  
7368 [SecureContext, Exposed=Window]  
7369 interface AuthenticatorResponse {  
7370 [SameObject] readonly attribute ArrayBuffer clientDataJSON;  
7371 };  
7372  
7373 [SecureContext, Exposed=Window]  
7374 interface AuthenticatorAttestationResponse : AuthenticatorResponse {  
7375 [SameObject] readonly attribute ArrayBuffer attestationObject;  
7376 };  
7377  
7378 [SecureContext, Exposed=Window]  
7379 interface AuthenticatorAssertionResponse : AuthenticatorResponse {  
7380 [SameObject] readonly attribute ArrayBuffer authenticatorData;  
7381 [SameObject] readonly attribute ArrayBuffer signature;  
7382 [SameObject] readonly attribute ArrayBuffer? userHandle;  
7383 };

```

7343 dictionary PublicKeyCredentialParameters {
7344     required PublicKeyCredentialType type;
7345     required COSEAlgorithmIdentifier alg;
7346 };
7347
7348 dictionary PublicKeyCredentialCreationOptions {
7349     required PublicKeyCredentialRpEntity rp;
7350     required PublicKeyCredentialUserEntity user;
7351
7352     required BufferSource challenge;
7353     required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
7354
7355     unsigned long timeout;
7356     sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
7357     AuthenticatorSelectionCriteria authenticatorSelection;
7358     AttestationConveyancePreference attestation = "none";
7359     AuthenticationExtensionsClientInputs extensions;
7360 };
7361
7362 dictionary PublicKeyCredentialEntity {
7363     required DOMString name;
7364     USVString icon;
7365 };
7366
7367 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
7368     DOMString id;
7369 };
7370
7371 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
7372     required BufferSource id;
7373     required DOMString displayName;
7374 };
7375
7376 dictionary AuthenticatorSelectionCriteria {
7377     AuthenticatorAttachment authenticatorAttachment;
7378     boolean requireResidentKey = false;
7379     UserVerificationRequirement userVerification = "preferred";
7380 };
7381
7382 enum AuthenticatorAttachment {
7383     "platform" // Platform attachment
7384     "cross-platform" // Cross-platform attachment
7385 };
7386
7387 enum AttestationConveyancePreference {
7388     "none",
7389     "indirect",
7390     "direct"
7391 };
7392
7393 dictionary PublicKeyCredentialRequestOptions {
7394     required BufferSource challenge;
7395     unsigned long timeout;
7396     USVString rpId;
7397     sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
7398     UserVerificationRequirement userVerification = "preferred";
7399     AuthenticationExtensionsClientInputs extensions;
7400 };
7401
7402 dictionary AuthenticationExtensionsClientInputs {
7403 };
7404
7405 dictionary AuthenticationExtensionsClientOutputs {
7406 };
7407
7408 typedef record<DOMString, DOMString> AuthenticationExtensionsAuthenticatorInputs
7409 ;
7410
7411 dictionary CollectedClientData {
7412

```

```

7384 dictionary PublicKeyCredentialParameters {
7385     required PublicKeyCredentialType type;
7386     required COSEAlgorithmIdentifier alg;
7387 };
7388
7389 dictionary PublicKeyCredentialCreationOptions {
7390     required PublicKeyCredentialRpEntity rp;
7391     required PublicKeyCredentialUserEntity user;
7392
7393     required BufferSource challenge;
7394     required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
7395
7396     unsigned long timeout;
7397     sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
7398     AuthenticatorSelectionCriteria authenticatorSelection;
7399     AttestationConveyancePreference attestation = "none";
7400     AuthenticationExtensionsClientInputs extensions;
7401 };
7402
7403 dictionary PublicKeyCredentialEntity {
7404     required DOMString name;
7405     USVString icon;
7406 };
7407
7408 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
7409     DOMString id;
7410 };
7411
7412 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
7413     required BufferSource id;
7414     required DOMString displayName;
7415 };
7416
7417 dictionary AuthenticatorSelectionCriteria {
7418     AuthenticatorAttachment authenticatorAttachment;
7419     boolean requireResidentKey = false;
7420     UserVerificationRequirement userVerification = "preferred";
7421 };
7422
7423 enum AuthenticatorAttachment {
7424     "platform" // Platform attachment
7425     "cross-platform" // Cross-platform attachment
7426 };
7427
7428 enum AttestationConveyancePreference {
7429     "none",
7430     "indirect",
7431     "direct"
7432 };
7433
7434 dictionary PublicKeyCredentialRequestOptions {
7435     required BufferSource challenge;
7436     unsigned long timeout;
7437     USVString rpId;
7438     sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
7439     UserVerificationRequirement userVerification = "preferred";
7440     AuthenticationExtensionsClientInputs extensions;
7441 };
7442
7443 dictionary AuthenticationExtensionsClientInputs {
7444 };
7445
7446 dictionary AuthenticationExtensionsClientOutputs {
7447 };
7448
7449 typedef record<DOMString, DOMString> AuthenticationExtensionsAuthenticatorInputs
7450 ;
7451
7452 dictionary CollectedClientData {
7453

```

```

7413 required DOMString type;
7414 required DOMString challenge;
7415 required DOMString origin;
7416 TokenBinding tokenBinding;
7417 };
7418
7419 dictionary TokenBinding {
7420 required TokenBindingStatus status;
7421 DOMString id;
7422 };
7423
7424 enum TokenBindingStatus { "present", "supported", "not-supported" };
7425
7426 enum PublicKeyCredentialType {
7427 "public-key"
7428 };
7429
7430 dictionary PublicKeyCredentialDescriptor {
7431 required PublicKeyCredentialType type;
7432 required BufferSource id;
7433 sequence<AuthenticatorTransport> transports;
7434 };
7435
7436 enum AuthenticatorTransport {
7437 "usb",
7438 "nfc",
7439 "ble",
7440 "internal"
7441 };
7442
7443 typedef long COSEAlgorithmIdentifier;
7444
7445 enum UserVerificationRequirement {
7446 "required",
7447 "preferred",
7448 "discouraged"
7449 };
7450
7451 partial dictionary AuthenticationExtensionsClientInputs {
7452 USVString appid;
7453 };
7454
7455 partial dictionary AuthenticationExtensionsClientOutputs {
7456 boolean appid;
7457 };
7458
7459 partial dictionary AuthenticationExtensionsClientInputs {
7460 USVString txAuthSimple;
7461 };
7462
7463 partial dictionary AuthenticationExtensionsClientOutputs {
7464 USVString txAuthSimple;
7465 };
7466
7467 dictionary txAuthGenericArg {
7468 required USVString contentType; // MIME-Type of the content, e.g., "image
7469 /png"
7470 required ArrayBuffer content;
7471 };
7472
7473 partial dictionary AuthenticationExtensionsClientInputs {
7474 txAuthGenericArg txAuthGeneric;
7475 };
7476
7477 partial dictionary AuthenticationExtensionsClientOutputs {
7478 ArrayBuffer txAuthGeneric;
7479 };
7480
7481 typedef sequence<AAGUID> AuthenticatorSelectionList;
7482

```

```

7454 required DOMString type;
7455 required DOMString challenge;
7456 required DOMString origin;
7457 TokenBinding tokenBinding;
7458 };
7459
7460 dictionary TokenBinding {
7461 required TokenBindingStatus status;
7462 DOMString id;
7463 };
7464
7465 enum TokenBindingStatus { "present", "supported", "not-supported" };
7466
7467 enum PublicKeyCredentialType {
7468 "public-key"
7469 };
7470
7471 dictionary PublicKeyCredentialDescriptor {
7472 required PublicKeyCredentialType type;
7473 required BufferSource id;
7474 sequence<AuthenticatorTransport> transports;
7475 };
7476
7477 enum AuthenticatorTransport {
7478 "usb",
7479 "nfc",
7480 "ble"
7481 };
7482
7483 typedef long COSEAlgorithmIdentifier;
7484
7485 enum UserVerificationRequirement {
7486 "required",
7487 "preferred",
7488 "discouraged"
7489 };
7490
7491 partial dictionary AuthenticationExtensionsClientInputs {
7492 USVString appid;
7493 };
7494
7495 partial dictionary AuthenticationExtensionsClientOutputs {
7496 boolean appid;
7497 };
7498
7499 partial dictionary AuthenticationExtensionsClientInputs {
7500 USVString txAuthSimple;
7501 };
7502
7503 partial dictionary AuthenticationExtensionsClientOutputs {
7504 USVString txAuthSimple;
7505 };
7506
7507 dictionary txAuthGenericArg {
7508 required USVString contentType; // MIME-Type of the content, e.g., "image
7509 /png"
7510 required ArrayBuffer content;
7511 };
7512
7513 partial dictionary AuthenticationExtensionsClientInputs {
7514 txAuthGenericArg txAuthGeneric;
7515 };
7516
7517 partial dictionary AuthenticationExtensionsClientOutputs {
7518 ArrayBuffer txAuthGeneric;
7519 };
7520
7521 typedef sequence<AAGUID> AuthenticatorSelectionList;
7522

```

```

7483 partial dictionary AuthenticationExtensionsClientInputs {
7484     AuthenticatorSelectionList authnSel;
7485 };
7486
7487 typedef BufferSource    AAGUID;
7488
7489 partial dictionary AuthenticationExtensionsClientOutputs {
7490     boolean authnSel;
7491 };
7492
7493 partial dictionary AuthenticationExtensionsClientInputs {
7494     boolean exts;
7495 };
7496
7497 typedef sequence<USVString> AuthenticationExtensionsSupported;
7498
7499 partial dictionary AuthenticationExtensionsClientOutputs {
7500     AuthenticationExtensionsSupported exts;
7501 };
7502
7503 partial dictionary AuthenticationExtensionsClientInputs {
7504     boolean uvi;
7505 };
7506
7507 partial dictionary AuthenticationExtensionsClientOutputs {
7508     ArrayBuffer uvi;
7509 };
7510
7511 partial dictionary AuthenticationExtensionsClientInputs {
7512     boolean loc;
7513 };
7514
7515 partial dictionary AuthenticationExtensionsClientOutputs {
7516     Coordinates loc;
7517 };
7518
7519 partial dictionary AuthenticationExtensionsClientInputs {
7520     boolean uvm;
7521 };
7522
7523 typedef sequence<unsigned long> UvmEntry;
7524 typedef sequence<UvmEntry> UvmEntries;
7525
7526 partial dictionary AuthenticationExtensionsClientOutputs {
7527     UvmEntries uvm;
7528 };
7529
7530 dictionary authenticatorBiometricPerfBounds{
7531     float FAR;
7532     float FRR;
7533 };

```

Issues Index

The definitions of "lifetime of" and "becomes available" are intended to represent how devices are hot-plugged into (USB) or discovered by (NFC) browsers, and are underspecified. Resolving this with good definitions or some other means will be addressed by resolving Issue #613. RET

@balfanz wishes to add to the "direct" case: If the authenticator violates the privacy requirements of the attestation type it is using, the client SHOULD terminate this algorithm with an "AttestationNotPrivateError". RET

The definitions of "lifetime of" and "becomes available" are intended to represent how devices are hot-plugged into (USB) or discovered by (NFC) browsers, and are underspecified. Resolving this with good definitions or some other means will be addressed by resolving Issue #613. RET

The foregoing step may be incorrect, in that we are attempting to

```

7523 partial dictionary AuthenticationExtensionsClientInputs {
7524     AuthenticatorSelectionList authnSel;
7525 };
7526
7527 typedef BufferSource    AAGUID;
7528
7529 partial dictionary AuthenticationExtensionsClientOutputs {
7530     boolean authnSel;
7531 };
7532
7533 partial dictionary AuthenticationExtensionsClientInputs {
7534     boolean exts;
7535 };
7536
7537 typedef sequence<USVString> AuthenticationExtensionsSupported;
7538
7539 partial dictionary AuthenticationExtensionsClientOutputs {
7540     AuthenticationExtensionsSupported exts;
7541 };
7542
7543 partial dictionary AuthenticationExtensionsClientInputs {
7544     boolean uvi;
7545 };
7546
7547 partial dictionary AuthenticationExtensionsClientOutputs {
7548     ArrayBuffer uvi;
7549 };
7550
7551 partial dictionary AuthenticationExtensionsClientInputs {
7552     boolean loc;
7553 };
7554
7555 partial dictionary AuthenticationExtensionsClientOutputs {
7556     Coordinates loc;
7557 };
7558
7559 partial dictionary AuthenticationExtensionsClientInputs {
7560     boolean uvm;
7561 };
7562
7563 typedef sequence<unsigned long> UvmEntry;
7564 typedef sequence<UvmEntry> UvmEntries;
7565
7566 partial dictionary AuthenticationExtensionsClientOutputs {
7567     UvmEntries uvm;
7568 };
7569
7570 dictionary authenticatorBiometricPerfBounds{
7571     float FAR;
7572     float FRR;
7573 };

```

Issues Index

@balfanz wishes to add to the "direct" case: If the authenticator violates the privacy requirements of the attestation type it is using, the client SHOULD terminate this algorithm with an "AttestationNotPrivateError". RET

7553 create savedCredentialId here and use it later below, and we do not  
7554 have a global in which to allocate a place for it. Perhaps this is good  
7555 enough? addendum: @jcjones feels the above step is likely good enough.  
7556 RET

7557 The WHATWG HTML WG is discussing whether to provide a hook when a  
7558 browsing context gains or loses focuses. If a hook is provided, the  
7559 above paragraph will be updated to include the hook. See WHATWG HTML WG  
7560 Issue #2711 for more details. RET

7561

7562 #base64url-encodingReferenced in:  
7563 \* 5.1. PublicKeyCredential Interface  
7564 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7565 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
7566 \* 5.1.4.1. PublicKeyCredential's  
7567 [[DiscoverFromExternalSource]](origin, options,  
7568 sameOriginWithAncestors) method (2)  
7569 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
7570 CollectedClientData)  
7571 \* 7.1. Registering a new credential  
7572 \* 7.2. Verifying an authentication assertion (2)  
7573

7574 #cborReferenced in:  
7575 \* 2.4. All Conformance Classes  
7576 \* 3. Dependencies  
7577 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7578 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
7579 \* 5.1.4.1. PublicKeyCredential's  
7580 [[DiscoverFromExternalSource]](origin, options,  
7581 sameOriginWithAncestors) method  
7582 \* 6.1. Authenticator data (2)  
7583 \* 6.2.2. The authenticatorMakeCredential operation  
7584 \* 6.2.3. The authenticatorGetAssertion operation  
7585 \* 9. WebAuthn Extensions (2) (3) (4) (5) (6) (7)  
7586 \* 9.2. Defining extensions (2)  
7587 \* 9.3. Extending request parameters  
7588 \* 9.4. Client extension processing (2)  
7589 \* 9.5. Authenticator extension processing (2)  
7590

7591 #assertionReferenced in:  
7592 \* 7.1. Registering a new credential  
7593 \* 10.1. FIDO AppID Extension (appid)  
7594 \* 13.4. credentialId Unsigned  
7595

7596 #attestationReferenced in:  
7597 \* 4. Terminology (2)  
7598 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
7599 AttestationConveyancePreference) (2)  
7600 \* 6. WebAuthn Authenticator Model (2)  
7601 \* 6.3. Attestation (2) (3) (4)  
7602 \* 8.2. Packed Attestation Statement Format  
7603 \* 11.1. WebAuthn Attestation Statement Format Identifier  
7604 Registrations  
7605 \* 13. Security Considerations  
7606 \* 13.3.1. Considerations for Self and None Attestation Types and  
7607 Ignoring Attestation  
7608

7609 #attestation-certificateReferenced in:  
7610 \* 4. Terminology (2)  
7611 \* 6.3.3. Attestation Types  
7612 \* 8.3.1. TPM attestation statement certificate requirements  
7613

7614 #attestation-key-pairReferenced in:  
7615 \* 4. Terminology (2)  
7616 \* 6.3. Attestation  
7617 \* 6.3.3. Attestation Types  
7618

7619 #attestation-private-keyReferenced in:  
7620 \* 6. WebAuthn Authenticator Model  
7621 \* 6.3. Attestation  
7622 \* 8.2. Packed Attestation Statement Format

7582 The WHATWG HTML WG is discussing whether to provide a hook when a  
7583 browsing context gains or loses focuses. If a hook is provided, the  
7584 above paragraph will be updated to include the hook. See WHATWG HTML WG  
7585 Issue #2711 for more details. RET

7586

7587 #base64url-encodingReferenced in:  
7588 \* 5.1. PublicKeyCredential Interface  
7589 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7590 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
7591 \* 5.1.4.1. PublicKeyCredential's  
7592 [[DiscoverFromExternalSource]](origin, options,  
7593 sameOriginWithAncestors) method (2)  
7594 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
7595 CollectedClientData)  
7596 \* 7.1. Registering a new credential  
7597 \* 7.2. Verifying an authentication assertion (2)  
7598

7599 #cborReferenced in:  
7600 \* 2.4. All Conformance Classes  
7601 \* 3. Dependencies  
7602 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7603 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
7604 \* 5.1.4.1. PublicKeyCredential's  
7605 [[DiscoverFromExternalSource]](origin, options,  
7606 sameOriginWithAncestors) method  
7607 \* 6.1. Authenticator data (2)  
7608 \* 6.2.2. The authenticatorMakeCredential operation  
7609 \* 6.2.3. The authenticatorGetAssertion operation  
7610 \* 9. WebAuthn Extensions (2) (3) (4) (5) (6) (7)  
7611 \* 9.2. Defining extensions (2)  
7612 \* 9.3. Extending request parameters  
7613 \* 9.4. Client extension processing (2)  
7614 \* 9.5. Authenticator extension processing (2)  
7615

7616 #assertionReferenced in:  
7617 \* 7.1. Registering a new credential  
7618 \* 10.1. FIDO AppID Extension (appid)  
7619 \* 13.4. credentialId Unsigned  
7620

7621 #attestationReferenced in:  
7622 \* 4. Terminology (2)  
7623 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
7624 AttestationConveyancePreference) (2)  
7625 \* 6. WebAuthn Authenticator Model (2)  
7626 \* 6.3. Attestation (2) (3) (4)  
7627 \* 8.2. Packed Attestation Statement Format  
7628 \* 11.1. WebAuthn Attestation Statement Format Identifier  
7629 Registrations  
7630 \* 13. Security Considerations  
7631 \* 13.3.1. Considerations for Self and None Attestation Types and  
7632 Ignoring Attestation  
7633

7634 #attestation-certificateReferenced in:  
7635 \* 4. Terminology (2)  
7636 \* 6.3.3. Attestation Types  
7637 \* 8.3.1. TPM attestation statement certificate requirements  
7638

7639 #attestation-key-pairReferenced in:  
7640 \* 4. Terminology (2)  
7641 \* 6.3. Attestation  
7642 \* 6.3.3. Attestation Types  
7643

7644 #attestation-private-keyReferenced in:  
7645 \* 6. WebAuthn Authenticator Model  
7646 \* 6.3. Attestation  
7647 \* 8.2. Packed Attestation Statement Format

7623 #attestation-public-keyReferenced in:  
7624 \* 6.3. Attestation  
7625 \* 8.2. Packed Attestation Statement Format  
7626  
7627 #authenticationReferenced in:  
7628 \* 1. Introduction (2)  
7629 \* 4. Terminology (2) (3) (4) (5) (6) (7)  
7630 \* 7.2. Verifying an authentication assertion (2) (3) (4)  
7631 \* 13. Security Considerations  
7632 \* 13.3. Security Benefits for Relying Parties  
7633 \* 13.3.1. Considerations for Self and None Attestation Types and  
7634 Ignoring Attestation (2)  
7635 \* 14.3. Authentication Ceremony Privacy  
7636  
7637 #authentication-assertionReferenced in:  
7638 \* 1. Introduction  
7639 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8)  
7640 \* 5.1. PublicKeyCredential Interface  
7641 \* 5.2.2. Web Authentication Assertion (interface  
7642 AuthenticatorAssertionResponse)  
7643 \* 5.5. Options for Assertion Generation (dictionary  
7644 PublicKeyCredentialRequestOptions)  
7645 \* 9. WebAuthn Extensions  
7646 \* 13.3.1. Considerations for Self and None Attestation Types and  
7647 Ignoring Attestation  
7648  
7649 #authenticatorReferenced in:  
7650 \* 1. Introduction (2) (3) (4)  
7651 \* 1.1. Use Cases  
7652 \* 2.2. Authenticators  
7653 \* 2.2.1. Backwards Compatibility with FIDO U2F  
7654 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)  
7655 (14) (15) (16) (17) (18) (19) (20) (21)  
7656 \* 5. Web Authentication API (2) (3)  
7657 \* 5.1. PublicKeyCredential Interface  
7658 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7659 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
7660 (4)  
7661 \* 5.1.4.1. PublicKeyCredential's  
7662 [[DiscoverFromExternalSource]](origin, options,  
7663 sameOriginWithAncestors) method (2) (3) (4) (5) (6)  
7664 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
7665 \* 5.2.1. Information about Public Key Credential (interface  
7666 AuthenticatorAttestationResponse) (2)  
7667 \* 5.2.2. Web Authentication Assertion (interface  
7668 AuthenticatorAssertionResponse)  
7669 \* 5.4.1. Public Key Entity Description (dictionary  
7670 PublicKeyCredentialEntity) (2)  
7671 \* 5.4.3. User Account Parameters for Credential Generation  
7672 (dictionary PublicKeyCredentialUserEntity)  
7673 \* 5.4.5. Authenticator Attachment enumeration (enum  
7674 AuthenticatorAttachment)  
7675 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
7676 AttestationConveyancePreference) (2)  
7677 \* 5.5. Options for Assertion Generation (dictionary  
7678 PublicKeyCredentialRequestOptions)  
7679 \* 5.10.4. Authenticator Transport enumeration (enum  
7680 AuthenticatorTransport)  
7681 \* 6. WebAuthn Authenticator Model (2) (3) (4) (5) (6)  
7682 \* 6.1. Authenticator data  
7683 \* 6.2.1. Lookup Credential Source by Credential ID algorithm  
7684 \* 6.2.2. The authenticatorMakeCredential operation (2)  
7685 \* 6.2.3. The authenticatorGetAssertion operation (2) (3) (4)  
7686 \* 6.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)  
7687 \* 6.3.2. Attestation Statement Formats  
7688 \* 6.3.3. Attestation Types (2) (3) (4)  
7689 \* 6.3.4. Generating an Attestation Object  
7690 \* 7.1. Registering a new credential (2)  
7691 \* 7.2. Verifying an authentication assertion  
7692

7648 #attestation-public-keyReferenced in:  
7649 \* 6.3. Attestation  
7650 \* 8.2. Packed Attestation Statement Format  
7651  
7652 #authenticationReferenced in:  
7653 \* 1. Introduction (2)  
7654 \* 4. Terminology (2) (3) (4) (5) (6) (7)  
7655 \* 7.2. Verifying an authentication assertion (2) (3) (4)  
7656 \* 13. Security Considerations  
7657 \* 13.3. Security Benefits for Relying Parties  
7658 \* 13.3.1. Considerations for Self and None Attestation Types and  
7659 Ignoring Attestation (2)  
7660 \* 14.3. Authentication Ceremony Privacy  
7661  
7662 #authentication-assertionReferenced in:  
7663 \* 1. Introduction  
7664 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8)  
7665 \* 5.1. PublicKeyCredential Interface  
7666 \* 5.2.2. Web Authentication Assertion (interface  
7667 AuthenticatorAssertionResponse)  
7668 \* 5.5. Options for Assertion Generation (dictionary  
7669 PublicKeyCredentialRequestOptions)  
7670 \* 9. WebAuthn Extensions  
7671 \* 13.3.1. Considerations for Self and None Attestation Types and  
7672 Ignoring Attestation  
7673  
7674 #authenticatorReferenced in:  
7675 \* 1. Introduction (2) (3) (4)  
7676 \* 1.1. Use Cases  
7677 \* 2.2. Authenticators  
7678 \* 2.2.1. Backwards Compatibility with FIDO U2F  
7679 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)  
7680 (14) (15) (16) (17) (18) (19) (20) (21)  
7681 \* 5. Web Authentication API (2) (3)  
7682 \* 5.1. PublicKeyCredential Interface  
7683 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7684 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
7685 (4) (5) (6) (7)  
7686 \* 5.1.4.1. PublicKeyCredential's  
7687 [[DiscoverFromExternalSource]](origin, options,  
7688 sameOriginWithAncestors) method (2) (3) (4) (5) (6) (7) (8)  
7689 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
7690 \* 5.2.1. Information about Public Key Credential (interface  
7691 AuthenticatorAttestationResponse) (2)  
7692 \* 5.2.2. Web Authentication Assertion (interface  
7693 AuthenticatorAssertionResponse)  
7694 \* 5.4.1. Public Key Entity Description (dictionary  
7695 PublicKeyCredentialEntity) (2)  
7696 \* 5.4.3. User Account Parameters for Credential Generation  
7697 (dictionary PublicKeyCredentialUserEntity)  
7698 \* 5.4.5. Authenticator Attachment enumeration (enum  
7699 AuthenticatorAttachment)  
7700 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
7701 AttestationConveyancePreference) (2)  
7702 \* 5.5. Options for Assertion Generation (dictionary  
7703 PublicKeyCredentialRequestOptions)  
7704  
7705 \* 6. WebAuthn Authenticator Model (2) (3) (4) (5) (6)  
7706 \* 6.1. Authenticator data  
7707 \* 6.2.1. Lookup Credential Source by Credential ID algorithm  
7708 \* 6.2.2. The authenticatorMakeCredential operation (2)  
7709 \* 6.2.3. The authenticatorGetAssertion operation (2) (3) (4)  
7710 \* 6.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)  
7711 \* 6.3.2. Attestation Statement Formats  
7712 \* 6.3.3. Attestation Types (2) (3) (4)  
7713 \* 6.3.4. Generating an Attestation Object  
7714 \* 7.1. Registering a new credential (2)  
7715 \* 7.2. Verifying an authentication assertion

7693 \* 8.2. Packed Attestation Statement Format  
7694 \* 8.4. Android Key Attestation Statement Format  
7695 \* 8.5. Android SafetyNet Attestation Statement Format  
7696 \* 8.7. None Attestation Statement Format  
7697 \* 10.5. Supported Extensions Extension (exts)  
7698 \* 10.6. User Verification Index Extension (uvi)  
7699 \* 10.8. User Verification Method Extension (uvm)  
7700 \* 12. Sample scenarios  
7701 \* 13. Security Considerations (2) (3) (4) (5)  
7702 \* 13.2.2. Attestation Certificate and Attestation Certificate CA  
7703 Compromise  
7704 \* 13.3. Security Benefits for Relying Parties (2) (3) (4) (5) (6)  
7705 \* 13.4. credentialId Unsigned  
7706 \* 14.1. Attestation Privacy (2) (3)  
7707 \* 14.2. Registration Ceremony Privacy (2) (3) (4) (5) (6)  
7708  
7709 #authorization-gestureReferenced in:  
7710 \* 1.1.1. Registration  
7711 \* 1.1.2. Authentication  
7712 \* 1.1.3. Other use cases and configurations  
7713 \* 4. Terminology (2) (3) (4) (5) (6)  
7714 \* 5.1.4. Use an existing credential to make an assertion -  
7715 PublicKeyCredential's [[Get]](options) method (2)  
7716 \* 5.1.6. Preventing silent access to an existing credential -  
7717 PublicKeyCredential's [[preventSilentAccess]](credential,  
7718 sameOriginWithAncestors) method  
7719  
7720 #biometric-recognitionReferenced in:  
7721 \* 4. Terminology (2) (3)  
7722  
7723 #biometric-authenticatorReferenced in:  
7724 \* 10.9. Biometric Authenticator Performance Bounds Extension  
7725 (biometricPerfBounds)  
7726  
7727 #ceremonyReferenced in:  
7728 \* 1. Introduction  
7729 \* 4. Terminology (2) (3) (4) (5) (6) (7)  
7730 \* 7.1. Registering a new credential (2)  
7731 \* 7.2. Verifying an authentication assertion (2)  
7732 \* 13. Security Considerations  
7733 \* 13.3. Security Benefits for Relying Parties  
7734 \* 13.3.1. Considerations for Self and None Attestation Types and  
7735 Ignoring Attestation (2)  
7736 \* 14.2. Registration Ceremony Privacy  
7737 \* 14.3. Authentication Ceremony Privacy (2)  
7738  
7739 #clientReferenced in:  
7740 \* 4. Terminology  
7741  
7742 \* 5.1.7. Availability of User-Verifying Platform Authenticator -  
7743 PublicKeyCredential's  
7744 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)  
7745 \* 5.4.5. Authenticator Attachment enumeration (enum  
7746 AuthenticatorAttachment) (2) (3)  
7747 \* 5.10.3. Credential Descriptor (dictionary  
7748 PublicKeyCredentialDescriptor)  
7749 \* 5.10.4. Authenticator Transport enumeration (enum  
7750 AuthenticatorTransport)  
7751 \* 7.1. Registering a new credential  
7752 \* 7.2. Verifying an authentication assertion  
7753 \* 13.3.1. Considerations for Self and None Attestation Types and  
7754 Ignoring Attestation  
7755 #client-side-resident-credential-private-keyReferenced in:  
7756 \* 4. Terminology (2)  
7757 \* 5.1.3. Create a new credential - PublicKeyCredential's

7716 \* 8.2. Packed Attestation Statement Format  
7717 \* 8.4. Android Key Attestation Statement Format  
7718 \* 8.5. Android SafetyNet Attestation Statement Format  
7719 \* 8.7. None Attestation Statement Format  
7720 \* 10.5. Supported Extensions Extension (exts)  
7721 \* 10.6. User Verification Index Extension (uvi)  
7722 \* 10.8. User Verification Method Extension (uvm)  
7723 \* 12. Sample scenarios  
7724 \* 13. Security Considerations (2) (3) (4) (5)  
7725 \* 13.2.2. Attestation Certificate and Attestation Certificate CA  
7726 Compromise  
7727 \* 13.3. Security Benefits for Relying Parties (2) (3) (4) (5) (6)  
7728 \* 13.4. credentialId Unsigned  
7729 \* 14.1. Attestation Privacy (2) (3)  
7730 \* 14.2. Registration Ceremony Privacy (2) (3) (4) (5) (6)  
7731  
7732 #authorization-gestureReferenced in:  
7733 \* 1.1.1. Registration  
7734 \* 1.1.2. Authentication  
7735 \* 1.1.3. Other use cases and configurations  
7736 \* 4. Terminology (2) (3) (4) (5) (6)  
7737 \* 5.1.4. Use an existing credential to make an assertion -  
7738 PublicKeyCredential's [[Get]](options) method (2)  
7739 \* 5.1.6. Preventing silent access to an existing credential -  
7740 PublicKeyCredential's [[preventSilentAccess]](credential,  
7741 sameOriginWithAncestors) method  
7742  
7743 #biometric-recognitionReferenced in:  
7744 \* 4. Terminology (2) (3)  
7745  
7746 #biometric-authenticatorReferenced in:  
7747 \* 10.9. Biometric Authenticator Performance Bounds Extension  
7748 (biometricPerfBounds)  
7749  
7750 #ceremonyReferenced in:  
7751 \* 1. Introduction  
7752 \* 4. Terminology (2) (3) (4) (5) (6) (7)  
7753 \* 7.1. Registering a new credential (2)  
7754 \* 7.2. Verifying an authentication assertion (2)  
7755 \* 13. Security Considerations  
7756 \* 13.3. Security Benefits for Relying Parties  
7757 \* 13.3.1. Considerations for Self and None Attestation Types and  
7758 Ignoring Attestation (2)  
7759 \* 14.2. Registration Ceremony Privacy  
7760 \* 14.3. Authentication Ceremony Privacy (2)  
7761  
7762 #clientReferenced in:  
7763 \* 4. Terminology  
7764 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7765 [[Create]](origin, options, sameOriginWithAncestors) method  
7766 \* 5.1.4.1. PublicKeyCredential's  
7767 [[DiscoverFromExternalSource]](origin, options,  
7768 sameOriginWithAncestors) method  
7769 \* 5.1.7. Availability of User-Verifying Platform Authenticator -  
7770 PublicKeyCredential's  
7771 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)  
7772 \* 5.4.5. Authenticator Attachment enumeration (enum  
7773 AuthenticatorAttachment) (2) (3)  
7774 \* 5.10.3. Credential Descriptor (dictionary  
7775 PublicKeyCredentialDescriptor)  
7776  
7777 \* 7.1. Registering a new credential  
7778 \* 7.2. Verifying an authentication assertion  
7779 \* 13.3.1. Considerations for Self and None Attestation Types and  
7780 Ignoring Attestation  
7781 #client-side-resident-credential-private-keyReferenced in:  
7782 \* 4. Terminology (2)  
7783 \* 5.1.3. Create a new credential - PublicKeyCredential's



775f [[Create]](origin, options, sameOriginWithAncestors) method  
775e \* 5.4.4. Authenticator Selection Criteria (dictionary  
776c AuthenticatorSelectionCriteria) (2)  
7761 \* 6.2.2. The authenticatorMakeCredential operation (2)  
7762  
7763 #conforming-user-agentReferenced in:  
7764 \* 1. Introduction  
7765 \* 2.1. User Agents  
7766 \* 2.2. Authenticators  
7767 \* 4. Terminology (2)  
7768  
7769 #credential-idReferenced in:  
7770 \* 4. Terminology (2) (3) (4)  
7771 \* 5.1. PublicKeyCredential Interface (2)  
7772 \* 5.1.4.1. PublicKeyCredential's  
7773 [[DiscoverFromExternalSource]](origin, options,  
7774 sameOriginWithAncestors) method  
7775 \* 5.2.1. Information about Public Key Credential (interface  
7776 AuthenticatorAttestationResponse)  
7777 \* 5.10.3. Credential Descriptor (dictionary  
7778 PublicKeyCredentialDescriptor)  
7779 \* 6.2.1. Lookup Credential Source by Credential ID algorithm  
7780 \* 6.2.2. The authenticatorMakeCredential operation  
7781 \* 6.2.3. The authenticatorGetAssertion operation  
7782 \* 6.3.1. Attested credential data  
7783 \* 7.1. Registering a new credential  
7784 \* 8.6. FIDO U2F Attestation Statement Format  
7785 \* 12.1. Registration  
7786 \* 12.3. Authentication (2) (3)  
7787 \* 13.4. credentialId Unsigned (2) (3)  
7788  
7789 #credential-public-keyReferenced in:  
779c \* 4. Terminology (2) (3) (4) (5) (6) (7) (8)  
7791 \* 5.2.1. Information about Public Key Credential (interface  
7792 AuthenticatorAttestationResponse)  
7793 \* 6. WebAuthn Authenticator Model  
7794 \* 6.3. Attestation (2) (3)  
7795 \* 6.3.1. Attested credential data (2) (3)  
7796 \* 12.1. Registration (2)  
7797 \* 13.3.1. Considerations for Self and None Attestation Types and  
7798 Ignoring Attestation  
7799 \* 13.4. credentialId Unsigned  
7800  
7801 #user-public-keyReferenced in:  
7802 \* 4. Terminology  
7803 \* 8.6. FIDO U2F Attestation Statement Format  
7804  
7805 #credential-key-pairReferenced in:  
7806 \* 4. Terminology (2) (3)  
7807  
7808 #credential-private-keyReferenced in:  
7809 \* 4. Terminology (2) (3) (4) (5) (6)  
781c \* 5.1. PublicKeyCredential Interface  
7811 \* 5.2.2. Web Authentication Assertion (interface  
7812 AuthenticatorAssertionResponse)  
7813 \* 6. WebAuthn Authenticator Model  
7814 \* 6.3. Attestation (2)  
7815 \* 7.2. Verifying an authentication assertion  
7816  
7817 #human-palatabilityReferenced in:  
7818 \* 4. Terminology  
7819 \* 5.4.1. Public Key Entity Description (dictionary  
782c PublicKeyCredentialEntity) (2)  
7821  
7822 #public-key-credential-sourceReferenced in:  
7823 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11)  
7824 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7825 [[Create]](origin, options, sameOriginWithAncestors) method  
7826 \* 6. WebAuthn Authenticator Model  
7827 \* 6.2.1. Lookup Credential Source by Credential ID algorithm (2)

7784 [[Create]](origin, options, sameOriginWithAncestors) method  
7785 \* 5.4.4. Authenticator Selection Criteria (dictionary  
7786 AuthenticatorSelectionCriteria) (2)  
7787 \* 6.2.2. The authenticatorMakeCredential operation (2)  
7788  
7789 #conforming-user-agentReferenced in:  
779c \* 1. Introduction  
7791 \* 2.1. User Agents  
7792 \* 2.2. Authenticators  
7793 \* 4. Terminology (2)  
7794  
7795 #credential-idReferenced in:  
7796 \* 4. Terminology (2) (3) (4)  
7797 \* 5.1. PublicKeyCredential Interface (2)  
7798 \* 5.1.4.1. PublicKeyCredential's  
7799 [[DiscoverFromExternalSource]](origin, options,  
780c sameOriginWithAncestors) method  
7801 \* 5.2.1. Information about Public Key Credential (interface  
7802 AuthenticatorAttestationResponse)  
7803 \* 5.10.3. Credential Descriptor (dictionary  
7804 PublicKeyCredentialDescriptor)  
7805 \* 6.2.1. Lookup Credential Source by Credential ID algorithm  
7806 \* 6.2.2. The authenticatorMakeCredential operation  
7807 \* 6.2.3. The authenticatorGetAssertion operation  
7808 \* 6.3.1. Attested credential data  
7809 \* 7.1. Registering a new credential  
781c \* 8.6. FIDO U2F Attestation Statement Format  
7811 \* 12.1. Registration  
7812 \* 12.3. Authentication (2) (3)  
7813 \* 13.4. credentialId Unsigned (2) (3)  
7814  
7815 #credential-public-keyReferenced in:  
7816 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8)  
7817 \* 5.2.1. Information about Public Key Credential (interface  
7818 AuthenticatorAttestationResponse)  
7819 \* 6. WebAuthn Authenticator Model  
782c \* 6.3. Attestation (2) (3)  
7821 \* 6.3.1. Attested credential data (2) (3)  
7822 \* 12.1. Registration (2)  
7823 \* 13.3.1. Considerations for Self and None Attestation Types and  
7824 Ignoring Attestation  
7825 \* 13.4. credentialId Unsigned  
7826  
7827 #user-public-keyReferenced in:  
7828 \* 4. Terminology  
7829 \* 8.6. FIDO U2F Attestation Statement Format  
783c  
7831 #credential-key-pairReferenced in:  
7832 \* 4. Terminology (2) (3)  
7833  
7834 #credential-private-keyReferenced in:  
7835 \* 4. Terminology (2) (3) (4) (5) (6)  
7836 \* 5.1. PublicKeyCredential Interface  
7837 \* 5.2.2. Web Authentication Assertion (interface  
7838 AuthenticatorAssertionResponse)  
7839 \* 6. WebAuthn Authenticator Model  
784c \* 6.3. Attestation (2)  
7841 \* 7.2. Verifying an authentication assertion  
7842  
7843 #human-palatabilityReferenced in:  
7844 \* 4. Terminology  
7845 \* 5.4.1. Public Key Entity Description (dictionary  
7846 PublicKeyCredentialEntity) (2)  
7847  
7848 #public-key-credential-sourceReferenced in:  
7849 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11)  
785c \* 5.1.3. Create a new credential - PublicKeyCredential's  
7851 [[Create]](origin, options, sameOriginWithAncestors) method  
7852 \* 6. WebAuthn Authenticator Model  
7853 \* 6.2.1. Lookup Credential Source by Credential ID algorithm (2)

7828 \* 6.2.2. The authenticatorMakeCredential operation  
7829 \* 6.2.3. The authenticatorGetAssertion operation (2)  
7830  
7831 #public-key-credential-source-typeReferenced in:  
7832 \* 6.2.2. The authenticatorMakeCredential operation (2)  
7833  
7834 #public-key-credential-source-idReferenced in:  
7835 \* 6.2.1. Lookup Credential Source by Credential ID algorithm (2)  
7836 \* 6.2.2. The authenticatorMakeCredential operation  
7837 \* 6.2.3. The authenticatorGetAssertion operation  
7838  
7839 #public-key-credential-source-privatekeyReferenced in:  
7840 \* 6.2.2. The authenticatorMakeCredential operation  
7841 \* 6.2.3. The authenticatorGetAssertion operation  
7842  
7843 #public-key-credential-source-rpidReferenced in:  
7844 \* 6. WebAuthn Authenticator Model  
7845 \* 6.2.2. The authenticatorMakeCredential operation  
7846 \* 6.2.3. The authenticatorGetAssertion operation  
7847  
7848 #public-key-credential-source-userhandleReferenced in:  
7849 \* 6. WebAuthn Authenticator Model  
7850 \* 6.2.2. The authenticatorMakeCredential operation  
7851 \* 6.2.3. The authenticatorGetAssertion operation (2)  
7852  
7853 #public-key-credential-source-otheruiReferenced in:  
7854 \* 6.2.2. The authenticatorMakeCredential operation  
7855  
7856 #public-key-credential-source-managing-authenticatorReferenced in:  
7857 \* 4. Terminology  
7858 \* 5.10.3. Credential Descriptor (dictionary  
7859 PublicKeyCredentialDescriptor)  
7860  
7861 #public-key-credentialReferenced in:  
7862 \* 1. Introduction (2) (3) (4) (5)  
7863 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8)  
7864 \* 5. Web Authentication API (2) (3) (4)  
7865 \* 5.1. PublicKeyCredential Interface  
7866 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7867 [[Create]](origin, options, sameOriginWithAncestors) method  
7868 \* 5.1.4. Use an existing credential to make an assertion -  
7869 PublicKeyCredential's [[Get]](options) method  
7870 \* 5.1.4.1. PublicKeyCredential's  
7871 [[DiscoverFromExternalSource]](origin, options,  
7872 sameOriginWithAncestors) method (2)  
7873 \* 5.2.1. Information about Public Key Credential (interface  
7874 AuthenticatorAttestationResponse)  
7875 \* 5.4.1. Public Key Entity Description (dictionary  
7876 PublicKeyCredentialEntity)  
7877 \* 5.4.4. Authenticator Selection Criteria (dictionary  
7878 AuthenticatorSelectionCriteria)  
7879 \* 5.4.5. Authenticator Attachment enumeration (enum  
7880 AuthenticatorAttachment) (2) (3)  
7881 \* 5.5. Options for Assertion Generation (dictionary  
7882 PublicKeyCredentialRequestOptions)  
7883 \* 5.10. Supporting Data Structures  
7884 \* 5.10.3. Credential Descriptor (dictionary  
7885 PublicKeyCredentialDescriptor) (2) (3) (4)  
7886 \* 6. WebAuthn Authenticator Model (2)  
7887 \* 6.2.3. The authenticatorGetAssertion operation  
7888 \* 6.3. Attestation (2)  
7889 \* 6.3.2. Attestation Statement Formats  
7890 \* 6.3.3. Attestation Types  
7891 \* 7.1. Registering a new credential  
7892 \* 7.2. Verifying an authentication assertion (2)  
7893 \* 9. WebAuthn Extensions (2)  
7894 \* 12. Sample scenarios  
7895 \* 13.2.2. Attestation Certificate and Attestation Certificate CA  
7896 Compromise (2)  
7897 \* 14.2. Registration Ceremony Privacy (2) (3)

7854 \* 6.2.2. The authenticatorMakeCredential operation  
7855 \* 6.2.3. The authenticatorGetAssertion operation (2)  
7856  
7857 #public-key-credential-source-typeReferenced in:  
7858 \* 6.2.2. The authenticatorMakeCredential operation (2)  
7859  
7860 #public-key-credential-source-idReferenced in:  
7861 \* 6.2.1. Lookup Credential Source by Credential ID algorithm (2)  
7862 \* 6.2.2. The authenticatorMakeCredential operation  
7863 \* 6.2.3. The authenticatorGetAssertion operation  
7864  
7865 #public-key-credential-source-privatekeyReferenced in:  
7866 \* 6.2.2. The authenticatorMakeCredential operation  
7867 \* 6.2.3. The authenticatorGetAssertion operation  
7868  
7869 #public-key-credential-source-rpidReferenced in:  
7870 \* 6. WebAuthn Authenticator Model  
7871 \* 6.2.2. The authenticatorMakeCredential operation  
7872 \* 6.2.3. The authenticatorGetAssertion operation  
7873  
7874 #public-key-credential-source-userhandleReferenced in:  
7875 \* 6. WebAuthn Authenticator Model  
7876 \* 6.2.2. The authenticatorMakeCredential operation  
7877 \* 6.2.3. The authenticatorGetAssertion operation (2)  
7878  
7879 #public-key-credential-source-otheruiReferenced in:  
7880 \* 6.2.2. The authenticatorMakeCredential operation  
7881  
7882 #public-key-credential-source-managing-authenticatorReferenced in:  
7883 \* 4. Terminology  
7884 \* 5.10.3. Credential Descriptor (dictionary  
7885 PublicKeyCredentialDescriptor)  
7886  
7887 #public-key-credentialReferenced in:  
7888 \* 1. Introduction (2) (3) (4) (5)  
7889 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8)  
7890 \* 5. Web Authentication API (2) (3) (4)  
7891 \* 5.1. PublicKeyCredential Interface  
7892 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7893 [[Create]](origin, options, sameOriginWithAncestors) method  
7894 \* 5.1.4. Use an existing credential to make an assertion -  
7895 PublicKeyCredential's [[Get]](options) method  
7896 \* 5.1.4.1. PublicKeyCredential's  
7897 [[DiscoverFromExternalSource]](origin, options,  
7898 sameOriginWithAncestors) method (2) (3)  
7899 \* 5.2.1. Information about Public Key Credential (interface  
7900 AuthenticatorAttestationResponse)  
7901 \* 5.4.1. Public Key Entity Description (dictionary  
7902 PublicKeyCredentialEntity)  
7903 \* 5.4.4. Authenticator Selection Criteria (dictionary  
7904 AuthenticatorSelectionCriteria)  
7905 \* 5.4.5. Authenticator Attachment enumeration (enum  
7906 AuthenticatorAttachment) (2) (3)  
7907 \* 5.5. Options for Assertion Generation (dictionary  
7908 PublicKeyCredentialRequestOptions)  
7909 \* 5.10. Supporting Data Structures  
7910 \* 5.10.3. Credential Descriptor (dictionary  
7911 PublicKeyCredentialDescriptor) (2) (3) (4)  
7912 \* 6. WebAuthn Authenticator Model (2)  
7913 \* 6.2.3. The authenticatorGetAssertion operation  
7914 \* 6.3. Attestation (2)  
7915 \* 6.3.2. Attestation Statement Formats  
7916 \* 6.3.3. Attestation Types  
7917 \* 7.1. Registering a new credential  
7918 \* 7.2. Verifying an authentication assertion (2)  
7919 \* 9. WebAuthn Extensions (2)  
7920 \* 12. Sample scenarios  
7921 \* 13.2.2. Attestation Certificate and Attestation Certificate CA  
7922 Compromise (2)  
7923 \* 14.2. Registration Ceremony Privacy (2) (3)

789E \* 14.3. Authentication Ceremony Privacy (2) (3) (4) (5)  
789F  
7900 #registrationReferenced in:  
7901 \* 1. Introduction (2)  
7902 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)  
7903 \* 7.1. Registering a new credential (2) (3)  
7904 \* 10.9. Biometric Authenticator Performance Bounds Extension  
7905 (biometricPerfBounds)  
7906 \* 13. Security Considerations  
7907 \* 13.3. Security Benefits for Relying Parties  
7908 \* 13.3.1. Considerations for Self and None Attestation Types and  
7909 Ignoring Attestation  
7910  
7911 #relying-partyReferenced in:  
7912 \* 1. Introduction (2) (3) (4) (5) (6) (7)  
7913 \* 1.1.3. Other use cases and configurations  
7914 \* 2.3. Relying Parties  
7915 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)  
7916 (14) (15) (16) (17) (18) (19) (20) (21) (22) (23) (24) (25) (26)  
7917 (27) (28) (29) (30)  
7918 \* 5. Web Authentication API (2) (3) (4) (5) (6) (7)  
7919 \* 5.1. PublicKeyCredential Interface (2)  
7920 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7921 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
7922 (4) (5)  
7923 \* 5.1.4. Use an existing credential to make an assertion -  
7924 PublicKeyCredential's [[Get]](options) method (2)  
7925 \* 5.1.4.1. PublicKeyCredential's  
7926 [[DiscoverFromExternalSource]](origin, options,  
7927 sameOriginWithAncestors) method (2) (3) (4)  
7928 \* 5.1.7. Availability of User-Verifying Platform Authenticator -  
7929 PublicKeyCredential's  
7930 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3)  
7931 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
7932 \* 5.2.1. Information about Public Key Credential (interface  
7933 AuthenticatorAttestationResponse) (2)  
7934 \* 5.2.2. Web Authentication Assertion (interface  
7935 AuthenticatorAssertionResponse)  
7936 \* 5.4. Options for Credential Creation (dictionary  
7937 PublicKeyCredentialCreationOptions) (2) (3) (4) (5)  
7938 \* 5.4.1. Public Key Entity Description (dictionary  
7939 PublicKeyCredentialEntity) (2) (3) (4) (5)  
7940 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
7941 PublicKeyCredentialRpEntity) (2)  
7942 \* 5.4.3. User Account Parameters for Credential Generation  
7943 (dictionary PublicKeyCredentialUserEntity)  
7944 \* 5.4.4. Authenticator Selection Criteria (dictionary  
7945 AuthenticatorSelectionCriteria) (2) (3)  
7946 \* 5.4.5. Authenticator Attachment enumeration (enum  
7947 AuthenticatorAttachment) (2) (3) (4) (5) (6)  
7948 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
7949 AttestationConveyancePreference) (2) (3) (4) (5) (6) (7)  
7950 \* 5.5. Options for Assertion Generation (dictionary  
7951 PublicKeyCredentialRequestOptions)  
7952 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
7953 CollectedClientData) (2) (3) (4)  
7954 \* 5.10.4. Authenticator Transport enumeration (enum  
7955 AuthenticatorTransport) (2)  
7956 \* 5.10.6. User Verification Requirement enumeration (enum  
7957 UserVerificationRequirement) (2) (3) (4)  
7958 \* 6. WebAuthn Authenticator Model (2) (3)  
7959 \* 6.1. Authenticator data (2)  
7960 \* 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6)  
7961 \* 6.2.2. The authenticatorMakeCredential operation (2) (3) (4) (5)  
7962 (6)  
7963 \* 6.2.3. The authenticatorGetAssertion operation (2) (3)  
7964 \* 6.3. Attestation (2) (3) (4) (5) (6)  
7965 \* 6.3.3. Attestation Types  
7966 \* 7. Relying Party Operations (2) (3) (4)  
7967 \* 7.1. Registering a new credential (2) (3) (4) (5) (6) (7) (8) (9)

7924 \* 14.3. Authentication Ceremony Privacy (2) (3) (4) (5)  
7925  
7926 #registrationReferenced in:  
7927 \* 1. Introduction (2)  
7928 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)  
7929 \* 7.1. Registering a new credential (2) (3)  
7930 \* 10.9. Biometric Authenticator Performance Bounds Extension  
7931 (biometricPerfBounds)  
7932 \* 13. Security Considerations  
7933 \* 13.3. Security Benefits for Relying Parties  
7934 \* 13.3.1. Considerations for Self and None Attestation Types and  
7935 Ignoring Attestation  
7936  
7937 #relying-partyReferenced in:  
7938 \* 1. Introduction (2) (3) (4) (5) (6) (7)  
7939 \* 1.1.3. Other use cases and configurations  
7940 \* 2.3. Relying Parties  
7941 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)  
7942 (14) (15) (16) (17) (18) (19) (20) (21) (22) (23) (24) (25) (26)  
7943 (27) (28) (29) (30)  
7944 \* 5. Web Authentication API (2) (3) (4) (5) (6) (7)  
7945 \* 5.1. PublicKeyCredential Interface (2)  
7946 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7947 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
7948 (4) (5)  
7949 \* 5.1.4. Use an existing credential to make an assertion -  
7950 PublicKeyCredential's [[Get]](options) method (2)  
7951 \* 5.1.4.1. PublicKeyCredential's  
7952 [[DiscoverFromExternalSource]](origin, options,  
7953 sameOriginWithAncestors) method (2) (3) (4)  
7954 \* 5.1.7. Availability of User-Verifying Platform Authenticator -  
7955 PublicKeyCredential's  
7956 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3)  
7957 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
7958 \* 5.2.1. Information about Public Key Credential (interface  
7959 AuthenticatorAttestationResponse) (2)  
7960 \* 5.2.2. Web Authentication Assertion (interface  
7961 AuthenticatorAssertionResponse)  
7962 \* 5.4. Options for Credential Creation (dictionary  
7963 PublicKeyCredentialCreationOptions) (2) (3) (4) (5)  
7964 \* 5.4.1. Public Key Entity Description (dictionary  
7965 PublicKeyCredentialEntity) (2) (3) (4) (5)  
7966 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
7967 PublicKeyCredentialRpEntity) (2)  
7968 \* 5.4.3. User Account Parameters for Credential Generation  
7969 (dictionary PublicKeyCredentialUserEntity)  
7970 \* 5.4.4. Authenticator Selection Criteria (dictionary  
7971 AuthenticatorSelectionCriteria) (2) (3)  
7972 \* 5.4.5. Authenticator Attachment enumeration (enum  
7973 AuthenticatorAttachment) (2) (3) (4) (5) (6)  
7974 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
7975 AttestationConveyancePreference) (2) (3) (4) (5) (6) (7)  
7976 \* 5.5. Options for Assertion Generation (dictionary  
7977 PublicKeyCredentialRequestOptions)  
7978 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
7979 CollectedClientData) (2) (3) (4)  
7980 \* 5.10.4. Authenticator Transport enumeration (enum  
7981 AuthenticatorTransport) (2)  
7982 \* 5.10.6. User Verification Requirement enumeration (enum  
7983 UserVerificationRequirement) (2) (3) (4)  
7984 \* 6. WebAuthn Authenticator Model (2) (3)  
7985 \* 6.1. Authenticator data (2)  
7986 \* 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6)  
7987 \* 6.2.2. The authenticatorMakeCredential operation (2) (3) (4) (5)  
7988 (6)  
7989 \* 6.2.3. The authenticatorGetAssertion operation (2) (3)  
7990 \* 6.3. Attestation (2) (3) (4) (5) (6)  
7991 \* 6.3.3. Attestation Types  
7992 \* 7. Relying Party Operations (2) (3) (4)  
7993 \* 7.1. Registering a new credential (2) (3) (4) (5) (6) (7) (8) (9)

796E (10) (11) (12) (13)  
796E \* 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6) (7)  
797C (8)  
7971 \* 8.4. Android Key Attestation Statement Format  
7972 \* 8.7. None Attestation Statement Format  
7973 \* 9. WebAuthn Extensions (2) (3) (4) (5)  
7974 \* 9.2. Defining extensions (2)  
797E \* 9.3. Extending request parameters (2) (3) (4)  
797E \* 10.1. FIDO AppID Extension (appid) (2)  
7977 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
797E \* 10.4. Authenticator Selection Extension (authnSel) (2) (3)  
797E \* 10.5. Supported Extensions Extension (exts) (2)  
798C \* 10.6. User Verification Index Extension (uvi)  
7981 \* 10.7. Location Extension (loc) (2)  
7982 \* 10.8. User Verification Method Extension (uvm)  
7983 \* 10.9. Biometric Authenticator Performance Bounds Extension  
7984 (biometricPerfBounds) (2) (3)  
7985 \* 11.2. WebAuthn Extension Identifier Registrations (2)  
798E \* 12.1. Registration (2) (3) (4) (5)  
7987 \* 12.2. Registration Specifically with User Verifying Platform  
798E Authenticator (2) (3)  
798E \* 12.3. Authentication (2) (3) (4) (5)  
799C \* 12.5. Decommissioning (2)  
7991 \* 13. Security Considerations (2) (3) (4)  
7992 \* 13.1. Cryptographic Challenges  
7993 \* 13.2.2. Attestation Certificate and Attestation Certificate CA  
7994 Compromise (2) (3) (4) (5) (6)  
799E \* 13.3. Security Benefits for Relying Parties (2) (3) (4) (5) (6) (7)  
799E \* 13.3.1. Considerations for Self and None Attestation Types and  
7997 Ignoring Attestation (2) (3) (4) (5) (6)  
799E \* 13.4. credentialId Unsigned  
799E \* 14.1. Attestation Privacy  
800C \* 14.2. Registration Ceremony Privacy (2) (3) (4)  
8001 \* 14.3. Authentication Ceremony Privacy (2) (3) (4)  
8002  
8003 #relying-party-identifierReferenced in:  
8004 \* 4. Terminology  
800E \* 5. Web Authentication API  
800E \* 5.4. Options for Credential Creation (dictionary  
8007 PublicKeyCredentialCreationOptions)  
800E \* 5.5. Options for Assertion Generation (dictionary  
800E PublicKeyCredentialRequestOptions)  
801C  
8011 #rp-idReferenced in:  
8012 \* 4. Terminology (2) (3) (4) (5)  
8013 \* 5. Web Authentication API (2) (3) (4) (5)  
8014 \* 5.1.3. Create a new credential - PublicKeyCredential's  
801E [[Create]](origin, options, sameOriginWithAncestors) method (2)  
801E \* 5.1.4.1. PublicKeyCredential's  
8017 [[DiscoverFromExternalSource]](origin, options,  
801E sameOriginWithAncestors) method (2)  
801E \* 5.4.2. RP Parameters for Credential Generation (dictionary  
802C PublicKeyCredentialRpEntity)  
8021 \* 6.1. Authenticator data (2) (3) (4) (5) (6)  
8022 \* 6.1.1. Signature Counter Considerations  
8023 \* 6.2.2. The authenticatorMakeCredential operation (2) (3)  
8024 \* 6.2.3. The authenticatorGetAssertion operation (2)  
802E \* 7.1. Registering a new credential (2)  
802E \* 7.2. Verifying an authentication assertion  
8027 \* 8.4. Android Key Attestation Statement Format  
802E \* 8.6. FIDO U2F Attestation Statement Format  
802E \* 10.1. FIDO AppID Extension (appid)  
803C  
8031 #test-of-user-presenceReferenced in:  
8032 \* 4. Terminology (2) (3) (4) (5) (6)  
8033 \* 6.2.2. The authenticatorMakeCredential operation (2)  
8034 \* 6.2.3. The authenticatorGetAssertion operation  
803E \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
803E \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
8037

7994 (10) (11) (12) (13)  
799E \* 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6) (7)  
799E (8)  
7997 \* 8.4. Android Key Attestation Statement Format  
799E \* 8.7. None Attestation Statement Format  
799E \* 9. WebAuthn Extensions (2) (3) (4) (5)  
800C \* 9.2. Defining extensions (2)  
8001 \* 9.3. Extending request parameters (2) (3) (4)  
8002 \* 10.1. FIDO AppID Extension (appid) (2)  
8003 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
8004 \* 10.4. Authenticator Selection Extension (authnSel) (2) (3)  
800E \* 10.5. Supported Extensions Extension (exts) (2)  
800E \* 10.6. User Verification Index Extension (uvi)  
8007 \* 10.7. Location Extension (loc) (2)  
800E \* 10.8. User Verification Method Extension (uvm)  
800E \* 10.9. Biometric Authenticator Performance Bounds Extension  
801C (biometricPerfBounds) (2) (3)  
8011 \* 11.2. WebAuthn Extension Identifier Registrations (2)  
8012 \* 12.1. Registration (2) (3) (4) (5)  
8013 \* 12.2. Registration Specifically with User Verifying Platform  
8014 Authenticator (2) (3)  
801E \* 12.3. Authentication (2) (3) (4) (5)  
801E \* 12.5. Decommissioning (2)  
8017 \* 13. Security Considerations (2) (3) (4)  
801E \* 13.1. Cryptographic Challenges  
801E \* 13.2.2. Attestation Certificate and Attestation Certificate CA  
802C Compromise (2) (3) (4) (5) (6)  
8021 \* 13.3. Security Benefits for Relying Parties (2) (3) (4) (5) (6) (7)  
8022 \* 13.3.1. Considerations for Self and None Attestation Types and  
802C Ignoring Attestation (2) (3) (4) (5) (6)  
8024 \* 13.4. credentialId Unsigned  
802E \* 14.1. Attestation Privacy  
802E \* 14.2. Registration Ceremony Privacy (2) (3) (4)  
8027 \* 14.3. Authentication Ceremony Privacy (2) (3) (4)  
802E  
802E #relying-party-identifierReferenced in:  
803C \* 4. Terminology  
8031 \* 5. Web Authentication API  
8032 \* 5.4. Options for Credential Creation (dictionary  
8033 PublicKeyCredentialCreationOptions)  
8034 \* 5.5. Options for Assertion Generation (dictionary  
803E PublicKeyCredentialRequestOptions)  
803E  
8037 #rp-idReferenced in:  
803E \* 4. Terminology (2) (3) (4) (5)  
803E \* 5. Web Authentication API (2) (3) (4) (5)  
803E \* 5.1.3. Create a new credential - PublicKeyCredential's  
8041 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
8042 \* 5.1.4.1. PublicKeyCredential's  
8043 [[DiscoverFromExternalSource]](origin, options,  
8044 sameOriginWithAncestors) method (2)  
804E \* 5.4.2. RP Parameters for Credential Generation (dictionary  
804E PublicKeyCredentialRpEntity)  
8047 \* 6.1. Authenticator data (2) (3) (4) (5) (6)  
804E \* 6.1.1. Signature Counter Considerations  
804E \* 6.2.2. The authenticatorMakeCredential operation (2) (3)  
805C \* 6.2.3. The authenticatorGetAssertion operation (2)  
8051 \* 7.1. Registering a new credential (2)  
8052 \* 7.2. Verifying an authentication assertion  
8053 \* 8.4. Android Key Attestation Statement Format  
8054 \* 8.6. FIDO U2F Attestation Statement Format  
805E \* 10.1. FIDO AppID Extension (appid)  
805E  
8057 #test-of-user-presenceReferenced in:  
805E \* 4. Terminology (2) (3) (4) (5) (6)  
805E \* 6.2.2. The authenticatorMakeCredential operation (2)  
806C \* 6.2.3. The authenticatorGetAssertion operation  
8061 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
8062 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
806C

```

803E #user-consentReferenced in:
803F * 1. Introduction (2)
8040 * 4. Terminology (2)
8041 * 5. Web Authentication API
8042 * 5.1.3. Create a new credential - PublicKeyCredential's
8043 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
8044 * 5.1.4. Use an existing credential to make an assertion -
8045 PublicKeyCredential's [[Get]](options) method
8046 * 5.1.4.1. PublicKeyCredential's
8047 [[DiscoverFromExternalSource]](origin, options,
8048 sameOriginWithAncestors) method
8049 * 5.2.2. Web Authentication Assertion (interface
8050 AuthenticatorAssertionResponse)
8051 * 5.4.6. Attestation Conveyance Preference enumeration (enum
8052 AttestationConveyancePreference)
8053 * 6. WebAuthn Authenticator Model (2) (3)
8054 * 6.2.2. The authenticatorMakeCredential operation (2) (3) (4) (5)
8055 (6) (7) (8)
8056 * 6.2.3. The authenticatorGetAssertion operation (2) (3) (4) (5)
8057 * 11.2. WebAuthn Extension Identifier Registrations
8058 * 14.2. Registration Ceremony Privacy (2)
8059 * 14.3. Authentication Ceremony Privacy (2) (3)
8060
8061 #user-handleReferenced in:
8062 * 2.2.1. Backwards Compatibility with FIDO U2F
8063 * 4. Terminology
8064 * 5.1.4.1. PublicKeyCredential's
8065 [[DiscoverFromExternalSource]](origin, options,
8066 sameOriginWithAncestors) method (2)
8067 * 5.2.2. Web Authentication Assertion (interface
8068 AuthenticatorAssertionResponse) (2)
8069 * 5.4.3. User Account Parameters for Credential Generation
8070 (dictionary PublicKeyCredentialUserEntity)
8071 * 6.2.2. The authenticatorMakeCredential operation
8072
8073 #user-verificationReferenced in:
8074 * 1. Introduction
8075 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
8076 * 5.1.3. Create a new credential - PublicKeyCredential's
8077 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
8078 * 5.1.4.1. PublicKeyCredential's
8079 [[DiscoverFromExternalSource]](origin, options,
8080 sameOriginWithAncestors) method (2) (3)
8081 * 5.1.7. Availability of User-Verifying Platform Authenticator -
8082 PublicKeyCredential's
8083 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)
8084 (5)
8085 * 5.4.4. Authenticator Selection Criteria (dictionary
8086 AuthenticatorSelectionCriteria)
8087 * 5.5. Options for Assertion Generation (dictionary
8088 PublicKeyCredentialRequestOptions)
8089 * 5.10.6. User Verification Requirement enumeration (enum
8090 UserVerificationRequirement) (2) (3) (4)
8091 * 6.2.2. The authenticatorMakeCredential operation (2) (3)
8092 * 6.2.3. The authenticatorGetAssertion operation
8093 * 7.1. Registering a new credential
8094 * 7.2. Verifying an authentication assertion
8095 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
8096 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
8097 * 12.2. Registration Specifically with User Verifying Platform
8098 Authenticator
8099 * 13.3. Security Benefits for Relying Parties
8100
8101 #concept-user-presentReferenced in:
8102 * 4. Terminology
8103 * 6.1. Authenticator data (2) (3)
8104 * 7.1. Registering a new credential
8105 * 7.2. Verifying an authentication assertion
8106
8107 #upReferenced in:

```

```

8064 #user-consentReferenced in:
8065 * 1. Introduction (2)
8066 * 4. Terminology (2)
8067 * 5. Web Authentication API
8068 * 5.1.3. Create a new credential - PublicKeyCredential's
8069 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
8070 * 5.1.4. Use an existing credential to make an assertion -
8071 PublicKeyCredential's [[Get]](options) method
8072 * 5.1.4.1. PublicKeyCredential's
8073 [[DiscoverFromExternalSource]](origin, options,
8074 sameOriginWithAncestors) method
8075 * 5.2.2. Web Authentication Assertion (interface
8076 AuthenticatorAssertionResponse)
8077 * 5.4.6. Attestation Conveyance Preference enumeration (enum
8078 AttestationConveyancePreference)
8079 * 6. WebAuthn Authenticator Model (2) (3)
8080 * 6.2.2. The authenticatorMakeCredential operation (2) (3) (4) (5)
8081 (6) (7) (8)
8082 * 6.2.3. The authenticatorGetAssertion operation (2) (3) (4) (5)
8083 * 11.2. WebAuthn Extension Identifier Registrations
8084 * 14.2. Registration Ceremony Privacy (2)
8085 * 14.3. Authentication Ceremony Privacy (2) (3)
8086
8087 #user-handleReferenced in:
8088 * 2.2.1. Backwards Compatibility with FIDO U2F
8089 * 4. Terminology
8090 * 5.1.4.1. PublicKeyCredential's
8091 [[DiscoverFromExternalSource]](origin, options,
8092 sameOriginWithAncestors) method (2)
8093 * 5.2.2. Web Authentication Assertion (interface
8094 AuthenticatorAssertionResponse) (2)
8095 * 5.4.3. User Account Parameters for Credential Generation
8096 (dictionary PublicKeyCredentialUserEntity)
8097 * 6.2.2. The authenticatorMakeCredential operation
8098
8099 #user-verificationReferenced in:
8100 * 1. Introduction
8101 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
8102 * 5.1.3. Create a new credential - PublicKeyCredential's
8103 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
8104 * 5.1.4.1. PublicKeyCredential's
8105 [[DiscoverFromExternalSource]](origin, options,
8106 sameOriginWithAncestors) method (2) (3)
8107 * 5.1.7. Availability of User-Verifying Platform Authenticator -
8108 PublicKeyCredential's
8109 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)
8110 (5)
8111 * 5.4.4. Authenticator Selection Criteria (dictionary
8112 AuthenticatorSelectionCriteria)
8113 * 5.5. Options for Assertion Generation (dictionary
8114 PublicKeyCredentialRequestOptions)
8115 * 5.10.6. User Verification Requirement enumeration (enum
8116 UserVerificationRequirement) (2) (3) (4)
8117 * 6.2.2. The authenticatorMakeCredential operation (2) (3)
8118 * 6.2.3. The authenticatorGetAssertion operation
8119 * 7.1. Registering a new credential
8120 * 7.2. Verifying an authentication assertion
8121 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
8122 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
8123 * 12.2. Registration Specifically with User Verifying Platform
8124 Authenticator
8125 * 13.3. Security Benefits for Relying Parties
8126
8127 #concept-user-presentReferenced in:
8128 * 4. Terminology
8129 * 6.1. Authenticator data (2) (3)
8130 * 7.1. Registering a new credential
8131 * 7.2. Verifying an authentication assertion
8132
8133 #upReferenced in:

```

810E \* 6.1. Authenticator data  
810F \* 6.1.2. FIDO U2F signature format compatibility  
8110  
8111 #concept-user-verifiedReferenced in:  
8112 \* 4. Terminology  
8113 \* 6.1. Authenticator data (2) (3)  
8114 \* 7.1. Registering a new credential  
8115 \* 7.2. Verifying an authentication assertion  
8116  
8117 #uvReferenced in:  
8118 \* 5.10.6. User Verification Requirement enumeration (enum  
8119 UserVerificationRequirement) (2)  
8120 \* 6.1. Authenticator data  
8121  
8122 #webauthn-clientReferenced in:  
8123 \* 4. Terminology (2) (3) (4)  
8124 \* 6.2. Authenticator operations  
8125 \* 6.2.2. The authenticatorMakeCredential operation  
8126 \* 6.2.3. The authenticatorGetAssertion operation  
8127 \* 13. Security Considerations  
8128  
8129 #web-authentication-apiReferenced in:  
8130 \* 1. Introduction (2) (3)  
8131 \* 4. Terminology (2) (3) (4)  
8132 \* 13. Security Considerations  
8133  
8134 #publickeycredentialReferenced in:  
8135 \* 1. Introduction  
8136 \* 5.1. PublicKeyCredential Interface (2) (3) (4) (5) (6) (7) (8)  
8137 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8138 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
8139 \* 5.1.4.1. PublicKeyCredential's  
8140 [[DiscoverFromExternalSource]](origin, options,  
8141 sameOriginWithAncestors) method  
8142 \* 5.1.5. Store an existing credential - PublicKeyCredential's  
8143 [[Store]](credential, sameOriginWithAncestors) method (2)  
8144 \* 5.1.7. Availability of User-Verifying Platform Authenticator -  
8145 PublicKeyCredential's  
8146 isUserVerifyingPlatformAuthenticatorAvailable() method  
8147 \* 5.10.3. Credential Descriptor (dictionary  
8148 PublicKeyCredentialDescriptor)  
8149 \* 7. Relying Party Operations  
8150 \* 7.2. Verifying an authentication assertion  
8151  
8152 #dom-publickeycredential-rawidReferenced in:  
8153 \* 5.1. PublicKeyCredential Interface  
8154 \* 7.2. Verifying an authentication assertion  
8155  
8156 #dom-publickeycredential-getclientextensionresultsReferenced in:  
8157 \* 5.1. PublicKeyCredential Interface  
8158 \* 9.4. Client extension processing  
8159  
8160 #dom-publickeycredential-responseReferenced in:  
8161 \* 5.1. PublicKeyCredential Interface  
8162 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8163 [[Create]](origin, options, sameOriginWithAncestors) method  
8164 \* 5.1.4.1. PublicKeyCredential's  
8165 [[DiscoverFromExternalSource]](origin, options,  
8166 sameOriginWithAncestors) method  
8167 \* 7.2. Verifying an authentication assertion (2)  
8168  
8169 #dom-publickeycredential-identifier-slotReferenced in:  
8170 \* 5.1. PublicKeyCredential Interface (2)  
8171 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8172 [[Create]](origin, options, sameOriginWithAncestors) method  
8173 \* 5.1.4.1. PublicKeyCredential's  
8174 [[DiscoverFromExternalSource]](origin, options,  
8175 sameOriginWithAncestors) method  
8176  
8177 #dom-publickeycredential-clientextensionsresults-slotReferenced in:

8134 \* 6.1. Authenticator data  
8135 \* 6.1.2. FIDO U2F signature format compatibility  
8136  
8137 #concept-user-verifiedReferenced in:  
8138 \* 4. Terminology  
8139 \* 6.1. Authenticator data (2) (3)  
8140 \* 7.1. Registering a new credential  
8141 \* 7.2. Verifying an authentication assertion  
8142  
8143 #uvReferenced in:  
8144 \* 5.10.6. User Verification Requirement enumeration (enum  
8145 UserVerificationRequirement) (2)  
8146 \* 6.1. Authenticator data  
8147  
8148 #webauthn-clientReferenced in:  
8149 \* 4. Terminology (2) (3) (4)  
8150 \* 6.2. Authenticator operations  
8151 \* 6.2.2. The authenticatorMakeCredential operation  
8152 \* 6.2.3. The authenticatorGetAssertion operation  
8153 \* 13. Security Considerations  
8154  
8155 #web-authentication-apiReferenced in:  
8156 \* 1. Introduction (2) (3)  
8157 \* 4. Terminology (2) (3) (4)  
8158 \* 13. Security Considerations  
8159  
8160 #publickeycredentialReferenced in:  
8161 \* 1. Introduction  
8162 \* 5.1. PublicKeyCredential Interface (2) (3) (4) (5) (6) (7) (8)  
8163 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8164 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
8165 \* 5.1.4.1. PublicKeyCredential's  
8166 [[DiscoverFromExternalSource]](origin, options,  
8167 sameOriginWithAncestors) method  
8168 \* 5.1.5. Store an existing credential - PublicKeyCredential's  
8169 [[Store]](credential, sameOriginWithAncestors) method (2)  
8170 \* 5.1.7. Availability of User-Verifying Platform Authenticator -  
8171 PublicKeyCredential's  
8172 isUserVerifyingPlatformAuthenticatorAvailable() method  
8173 \* 5.10.3. Credential Descriptor (dictionary  
8174 PublicKeyCredentialDescriptor)  
8175 \* 7. Relying Party Operations  
8176 \* 7.2. Verifying an authentication assertion  
8177  
8178 #dom-publickeycredential-rawidReferenced in:  
8179 \* 5.1. PublicKeyCredential Interface  
8180 \* 7.2. Verifying an authentication assertion  
8181  
8182 #dom-publickeycredential-getclientextensionresultsReferenced in:  
8183 \* 5.1. PublicKeyCredential Interface  
8184 \* 9.4. Client extension processing  
8185  
8186 #dom-publickeycredential-responseReferenced in:  
8187 \* 5.1. PublicKeyCredential Interface  
8188 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8189 [[Create]](origin, options, sameOriginWithAncestors) method  
8190 \* 5.1.4.1. PublicKeyCredential's  
8191 [[DiscoverFromExternalSource]](origin, options,  
8192 sameOriginWithAncestors) method  
8193 \* 7.2. Verifying an authentication assertion (2)  
8194  
8195 #dom-publickeycredential-identifier-slotReferenced in:  
8196 \* 5.1. PublicKeyCredential Interface (2)  
8197 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8198 [[Create]](origin, options, sameOriginWithAncestors) method  
8199 \* 5.1.4.1. PublicKeyCredential's  
8200 [[DiscoverFromExternalSource]](origin, options,  
8201 sameOriginWithAncestors) method  
8202  
8203 #dom-publickeycredential-clientextensionsresults-slotReferenced in:

```

817f * 5.1. PublicKeyCredential Interface
817e * 5.1.3. Create a new credential - PublicKeyCredential's
818c [[Create]](origin, options, sameOriginWithAncestors) method
8181 * 5.1.4.1. PublicKeyCredential's
8182 [[DiscoverFromExternalSource]](origin, options,
8183 sameOriginWithAncestors) method
8184
8185 #dom-credentialcreationoptions-publickeyReferenced in:
8186 * 5.1.3. Create a new credential - PublicKeyCredential's
8187 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
8188
8189 #dom-credentialrequestoptions-publickeyReferenced in:
8190 * 5.1.4.1. PublicKeyCredential's
8191 [[DiscoverFromExternalSource]](origin, options,
8192 sameOriginWithAncestors) method (2) (3)
8193
8194 #dom-publickeycredential-create-slotReferenced in:
8195 * 4. Terminology
8196 * 5.1. PublicKeyCredential Interface
8197 * 5.4.5. Authenticator Attachment enumeration (enum
8198 AuthenticatorAttachment)
8199 * 5.6. Abort operations with AbortSignal (2) (3) (4) (5)
8200 * 6.2.2. The authenticatorMakeCredential operation
8201 * 14.2. Registration Ceremony Privacy
8202
8203 #dom-publickeycredential-create-origin-options-sameoriginwithancestors-
8204 originReferenced in:
8205 * 5.1.3. Create a new credential - PublicKeyCredential's
8206 [[Create]](origin, options, sameOriginWithAncestors) method
8207
8208 #dom-publickeycredential-create-origin-options-sameoriginwithancestors-
8209 optionsReferenced in:
8210 * 7.1. Registering a new credential
8211
8212 #effective-user-verification-requirement-for-credential-creationReferen
8213 ced in:
8214 * 6.2.2. The authenticatorMakeCredential operation
8215
8216 #credentialcreationdata-attestationobjectresultReferenced in:
8217 * 5.1.3. Create a new credential - PublicKeyCredential's
8218 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
8219 (4) (5)
8220
8221 #credentialcreationdata-clientdatajsonresultReferenced in:
8222 * 5.1.3. Create a new credential - PublicKeyCredential's
8223 [[Create]](origin, options, sameOriginWithAncestors) method
8224
8225 #credentialcreationdata-attestationconveyancepreferenceoptionReferenced
8226 in:
8227 * 5.1.3. Create a new credential - PublicKeyCredential's
8228 [[Create]](origin, options, sameOriginWithAncestors) method
8229
8230 #credentialcreationdata-clientextensionresultsReferenced in:
8231 * 5.1.3. Create a new credential - PublicKeyCredential's
8232 [[Create]](origin, options, sameOriginWithAncestors) method
8233
8234 #dom-publickeycredential-collectfromcredentialstore-slotReferenced in:
8235 * 5.1.4. Use an existing credential to make an assertion -
8236 PublicKeyCredential's [[Get]](options) method
8237
8238 #dom-publickeycredential-discoverfromexternalsource-slotReferenced in:
8239 * 4. Terminology
8240 * 5.1. PublicKeyCredential Interface
8241 * 5.1.4. Use an existing credential to make an assertion -
8242 PublicKeyCredential's [[Get]](options) method
8243 * 5.4.5. Authenticator Attachment enumeration (enum
8244 AuthenticatorAttachment)
8245 * 5.6. Abort operations with AbortSignal (2) (3) (4) (5)
8246 * 6.2.3. The authenticatorGetAssertion operation
8247 * 14.3. Authentication Ceremony Privacy

```

```

8204 * 5.1. PublicKeyCredential Interface
8205 * 5.1.3. Create a new credential - PublicKeyCredential's
8206 [[Create]](origin, options, sameOriginWithAncestors) method
8207 * 5.1.4.1. PublicKeyCredential's
8208 [[DiscoverFromExternalSource]](origin, options,
8209 sameOriginWithAncestors) method
8210
8211 #dom-credentialcreationoptions-publickeyReferenced in:
8212 * 5.1.3. Create a new credential - PublicKeyCredential's
8213 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
8214
8215 #dom-credentialrequestoptions-publickeyReferenced in:
8216 * 5.1.4.1. PublicKeyCredential's
8217 [[DiscoverFromExternalSource]](origin, options,
8218 sameOriginWithAncestors) method (2) (3)
8219
8220 #dom-publickeycredential-create-slotReferenced in:
8221 * 4. Terminology
8222 * 5.1. PublicKeyCredential Interface
8223 * 5.4.5. Authenticator Attachment enumeration (enum
8224 AuthenticatorAttachment)
8225 * 5.6. Abort operations with AbortSignal (2) (3) (4) (5)
8226 * 6.2.2. The authenticatorMakeCredential operation
8227 * 14.2. Registration Ceremony Privacy
8228
8229 #dom-publickeycredential-create-origin-options-sameoriginwithancestors-
8230 originReferenced in:
8231 * 5.1.3. Create a new credential - PublicKeyCredential's
8232 [[Create]](origin, options, sameOriginWithAncestors) method
8233
8234 #dom-publickeycredential-create-origin-options-sameoriginwithancestors-
8235 optionsReferenced in:
8236 * 7.1. Registering a new credential
8237
8238 #effective-user-verification-requirement-for-credential-creationReferen
8239 ced in:
8240 * 6.2.2. The authenticatorMakeCredential operation
8241
8242 #credentialcreationdata-attestationobjectresultReferenced in:
8243 * 5.1.3. Create a new credential - PublicKeyCredential's
8244 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
8245 (4) (5)
8246
8247 #credentialcreationdata-clientdatajsonresultReferenced in:
8248 * 5.1.3. Create a new credential - PublicKeyCredential's
8249 [[Create]](origin, options, sameOriginWithAncestors) method
8250
8251 #credentialcreationdata-attestationconveyancepreferenceoptionReferenced
8252 in:
8253 * 5.1.3. Create a new credential - PublicKeyCredential's
8254 [[Create]](origin, options, sameOriginWithAncestors) method
8255
8256 #credentialcreationdata-clientextensionresultsReferenced in:
8257 * 5.1.3. Create a new credential - PublicKeyCredential's
8258 [[Create]](origin, options, sameOriginWithAncestors) method
8259
8260 #dom-publickeycredential-collectfromcredentialstore-slotReferenced in:
8261 * 5.1.4. Use an existing credential to make an assertion -
8262 PublicKeyCredential's [[Get]](options) method
8263
8264 #dom-publickeycredential-discoverfromexternalsource-slotReferenced in:
8265 * 4. Terminology
8266 * 5.1. PublicKeyCredential Interface
8267 * 5.1.4. Use an existing credential to make an assertion -
8268 PublicKeyCredential's [[Get]](options) method
8269 * 5.4.5. Authenticator Attachment enumeration (enum
8270 AuthenticatorAttachment)
8271 * 5.6. Abort operations with AbortSignal (2) (3) (4) (5)
8272 * 6.2.3. The authenticatorGetAssertion operation
8273 * 14.3. Authentication Ceremony Privacy

```

8248 #dom-publickeycredential-discoverfromexternalsource-origin-options-same  
8249 originwithancestors-originReferenced in:  
8250 \* 5.1.4.1. PublicKeyCredential's  
8251 [[DiscoverFromExternalSource]](origin, options,  
8252 sameOriginWithAncestors) method  
8253  
8254 #effective-user-verification-requirement-for-assertionReferenced in:  
8255 \* 6.2.3. The authenticatorGetAssertion operation  
8256  
8257 #assertioncreationdata-credentialidresultReferenced in:  
8258 \* 5.1.4.1. PublicKeyCredential's  
8259 [[DiscoverFromExternalSource]](origin, options,  
8260 sameOriginWithAncestors) method (2) (3)  
8261  
8262 #assertioncreationdata-clientdatajsonresultReferenced in:  
8263 \* 5.1.4.1. PublicKeyCredential's  
8264 [[DiscoverFromExternalSource]](origin, options,  
8265 sameOriginWithAncestors) method  
8266  
8267 #assertioncreationdata-authenticatordataresultReferenced in:  
8268 \* 5.1.4.1. PublicKeyCredential's  
8269 [[DiscoverFromExternalSource]](origin, options,  
8270 sameOriginWithAncestors) method  
8271  
8272 #assertioncreationdata-signatureresultReferenced in:  
8273 \* 5.1.4.1. PublicKeyCredential's  
8274 [[DiscoverFromExternalSource]](origin, options,  
8275 sameOriginWithAncestors) method  
8276  
8277 #assertioncreationdata-userhandlerresultReferenced in:  
8278 \* 5.1.4.1. PublicKeyCredential's  
8279 [[DiscoverFromExternalSource]](origin, options,  
8280 sameOriginWithAncestors) method (2) (3) (4)  
8281 \* 6.2.3. The authenticatorGetAssertion operation  
8282  
8283 #assertioncreationdata-clientextensionresultsReferenced in:  
8284 \* 5.1.4.1. PublicKeyCredential's  
8285 [[DiscoverFromExternalSource]](origin, options,  
8286 sameOriginWithAncestors) method  
8287  
8288 #authenticatorresponseReferenced in:  
8289 \* 5.1. PublicKeyCredential Interface (2)  
8290 \* 5.2. Authenticator Responses (interface AuthenticatorResponse) (2)  
8291 \* 5.2.1. Information about Public Key Credential (interface  
8292 AuthenticatorAttestationResponse) (2)  
8293 \* 5.2.2. Web Authentication Assertion (interface  
8294 AuthenticatorAssertionResponse) (2)  
8295  
8296 #dom-authenticatorresponse-clientdatajsonReferenced in:  
8297 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8298 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
8299 \* 5.1.4.1. PublicKeyCredential's  
8300 [[DiscoverFromExternalSource]](origin, options,  
8301 sameOriginWithAncestors) method (2)  
8302 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
8303 \* 5.2.1. Information about Public Key Credential (interface  
8304 AuthenticatorAttestationResponse)  
8305 \* 5.2.2. Web Authentication Assertion (interface  
8306 AuthenticatorAssertionResponse)  
8307 \* 7.1. Registering a new credential (2)  
8308 \* 7.2. Verifying an authentication assertion  
8309  
8310 #authenticatorattestationresponseReferenced in:  
8311 \* 5.1. PublicKeyCredential Interface  
8312 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8313 [[Create]](origin, options, sameOriginWithAncestors) method  
8314 \* 5.2.1. Information about Public Key Credential (interface  
8315 AuthenticatorAttestationResponse) (2)  
8316 \* 7. Relying Party Operations

8274 #dom-publickeycredential-discoverfromexternalsource-origin-options-same  
8275 originwithancestors-originReferenced in:  
8276 \* 5.1.4.1. PublicKeyCredential's  
8277 [[DiscoverFromExternalSource]](origin, options,  
8278 sameOriginWithAncestors) method  
8279  
8280 #effective-user-verification-requirement-for-assertionReferenced in:  
8281 \* 6.2.3. The authenticatorGetAssertion operation  
8282  
8283 #assertioncreationdata-credentialidresultReferenced in:  
8284 \* 5.1.4.1. PublicKeyCredential's  
8285 [[DiscoverFromExternalSource]](origin, options,  
8286 sameOriginWithAncestors) method (2) (3)  
8287  
8288 #assertioncreationdata-clientdatajsonresultReferenced in:  
8289 \* 5.1.4.1. PublicKeyCredential's  
8290 [[DiscoverFromExternalSource]](origin, options,  
8291 sameOriginWithAncestors) method  
8292  
8293 #assertioncreationdata-authenticatordataresultReferenced in:  
8294 \* 5.1.4.1. PublicKeyCredential's  
8295 [[DiscoverFromExternalSource]](origin, options,  
8296 sameOriginWithAncestors) method  
8297  
8298 #assertioncreationdata-signatureresultReferenced in:  
8299 \* 5.1.4.1. PublicKeyCredential's  
8300 [[DiscoverFromExternalSource]](origin, options,  
8301 sameOriginWithAncestors) method  
8302  
8303 #assertioncreationdata-userhandlerresultReferenced in:  
8304 \* 5.1.4.1. PublicKeyCredential's  
8305 [[DiscoverFromExternalSource]](origin, options,  
8306 sameOriginWithAncestors) method (2) (3) (4)  
8307 \* 6.2.3. The authenticatorGetAssertion operation  
8308  
8309 #assertioncreationdata-clientextensionresultsReferenced in:  
8310 \* 5.1.4.1. PublicKeyCredential's  
8311 [[DiscoverFromExternalSource]](origin, options,  
8312 sameOriginWithAncestors) method  
8313  
8314 #authenticatorresponseReferenced in:  
8315 \* 5.1. PublicKeyCredential Interface (2)  
8316 \* 5.2. Authenticator Responses (interface AuthenticatorResponse) (2)  
8317 \* 5.2.1. Information about Public Key Credential (interface  
8318 AuthenticatorAttestationResponse) (2)  
8319 \* 5.2.2. Web Authentication Assertion (interface  
8320 AuthenticatorAssertionResponse) (2)  
8321  
8322 #dom-authenticatorresponse-clientdatajsonReferenced in:  
8323 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8324 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
8325 \* 5.1.4.1. PublicKeyCredential's  
8326 [[DiscoverFromExternalSource]](origin, options,  
8327 sameOriginWithAncestors) method (2)  
8328 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
8329 \* 5.2.1. Information about Public Key Credential (interface  
8330 AuthenticatorAttestationResponse)  
8331 \* 5.2.2. Web Authentication Assertion (interface  
8332 AuthenticatorAssertionResponse)  
8333 \* 7.1. Registering a new credential (2)  
8334 \* 7.2. Verifying an authentication assertion  
8335  
8336 #authenticatorattestationresponseReferenced in:  
8337 \* 5.1. PublicKeyCredential Interface  
8338 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8339 [[Create]](origin, options, sameOriginWithAncestors) method  
8340 \* 5.2.1. Information about Public Key Credential (interface  
8341 AuthenticatorAttestationResponse) (2)  
8342 \* 7. Relying Party Operations



8318 \* 7.1. Registering a new credential (2)  
8319  
8320 #dom-authenticatorattestationresponse-attestationobjectReferenced in:  
8321 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8322 [[Create]](origin, options, sameOriginWithAncestors) method  
8323 \* 5.2.1. Information about Public Key Credential (interface  
8324 AuthenticatorAttestationResponse)  
8325 \* 7.1. Registering a new credential  
8326  
8327 #authenticatorassertionresponseReferenced in:  
8328 \* 4. Terminology  
8329 \* 5.1. PublicKeyCredential Interface  
8330 \* 5.1.4.1. PublicKeyCredential's  
8331 [[DiscoverFromExternalSource]](origin, options,  
8332 sameOriginWithAncestors) method  
8333 \* 5.2.2. Web Authentication Assertion (interface  
8334 AuthenticatorAssertionResponse) (2)  
8335 \* 7. Relying Party Operations  
8336  
8337 #dom-authenticatorassertionresponse-authenticatordataReferenced in:  
8338 \* 5.1.4.1. PublicKeyCredential's  
8339 [[DiscoverFromExternalSource]](origin, options,  
8340 sameOriginWithAncestors) method  
8341 \* 5.2.2. Web Authentication Assertion (interface  
8342 AuthenticatorAssertionResponse)  
8343 \* 7.2. Verifying an authentication assertion  
8344  
8345 #dom-authenticatorassertionresponse-signatureReferenced in:  
8346 \* 5.1.4.1. PublicKeyCredential's  
8347 [[DiscoverFromExternalSource]](origin, options,  
8348 sameOriginWithAncestors) method  
8349 \* 5.2.2. Web Authentication Assertion (interface  
8350 AuthenticatorAssertionResponse)  
8351 \* 7.2. Verifying an authentication assertion  
8352  
8353 #dom-authenticatorassertionresponse-userhandleReferenced in:  
8354 \* 2.2.1. Backwards Compatibility with FIDO U2F  
8355 \* 5.1.4.1. PublicKeyCredential's  
8356 [[DiscoverFromExternalSource]](origin, options,  
8357 sameOriginWithAncestors) method  
8358 \* 5.2.2. Web Authentication Assertion (interface  
8359 AuthenticatorAssertionResponse)  
8360 \* 7.2. Verifying an authentication assertion  
8361  
8362 #dictdef-publickeycredentialparametersReferenced in:  
8363 \* 5.3. Parameters for Credential Generation (dictionary  
8364 PublicKeyCredentialParameters)  
8365 \* 5.4. Options for Credential Creation (dictionary  
8366 PublicKeyCredentialCreationOptions) (2)  
8367  
8368 #dom-publickeycredentialparameters-typeReferenced in:  
8369 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8370 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
8371 \* 5.3. Parameters for Credential Generation (dictionary  
8372 PublicKeyCredentialParameters)  
8373  
8374 #dom-publickeycredentialparameters-algReferenced in:  
8375 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8376 [[Create]](origin, options, sameOriginWithAncestors) method  
8377 \* 5.3. Parameters for Credential Generation (dictionary  
8378 PublicKeyCredentialParameters)  
8379  
8380 #dictdef-publickeycredentialcreationoptionsReferenced in:  
8381 \* 5.1.1. CredentialCreationOptions Dictionary Extension  
8382 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8383 [[Create]](origin, options, sameOriginWithAncestors) method  
8384 \* 5.4. Options for Credential Creation (dictionary  
8385 PublicKeyCredentialCreationOptions)  
8386  
8387 #dom-publickeycredentialcreationoptions-rpReferenced in:

8344 \* 7.1. Registering a new credential (2)  
8345  
8346 #dom-authenticatorattestationresponse-attestationobjectReferenced in:  
8347 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8348 [[Create]](origin, options, sameOriginWithAncestors) method  
8349 \* 5.2.1. Information about Public Key Credential (interface  
8350 AuthenticatorAttestationResponse)  
8351 \* 7.1. Registering a new credential  
8352  
8353 #authenticatorassertionresponseReferenced in:  
8354 \* 4. Terminology  
8355 \* 5.1. PublicKeyCredential Interface  
8356 \* 5.1.4.1. PublicKeyCredential's  
8357 [[DiscoverFromExternalSource]](origin, options,  
8358 sameOriginWithAncestors) method  
8359 \* 5.2.2. Web Authentication Assertion (interface  
8360 AuthenticatorAssertionResponse) (2)  
8361 \* 7. Relying Party Operations  
8362  
8363 #dom-authenticatorassertionresponse-authenticatordataReferenced in:  
8364 \* 5.1.4.1. PublicKeyCredential's  
8365 [[DiscoverFromExternalSource]](origin, options,  
8366 sameOriginWithAncestors) method  
8367 \* 5.2.2. Web Authentication Assertion (interface  
8368 AuthenticatorAssertionResponse)  
8369 \* 7.2. Verifying an authentication assertion  
8370  
8371 #dom-authenticatorassertionresponse-signatureReferenced in:  
8372 \* 5.1.4.1. PublicKeyCredential's  
8373 [[DiscoverFromExternalSource]](origin, options,  
8374 sameOriginWithAncestors) method  
8375 \* 5.2.2. Web Authentication Assertion (interface  
8376 AuthenticatorAssertionResponse)  
8377 \* 7.2. Verifying an authentication assertion  
8378  
8379 #dom-authenticatorassertionresponse-userhandleReferenced in:  
8380 \* 2.2.1. Backwards Compatibility with FIDO U2F  
8381 \* 5.1.4.1. PublicKeyCredential's  
8382 [[DiscoverFromExternalSource]](origin, options,  
8383 sameOriginWithAncestors) method  
8384 \* 5.2.2. Web Authentication Assertion (interface  
8385 AuthenticatorAssertionResponse)  
8386 \* 7.2. Verifying an authentication assertion  
8387  
8388 #dictdef-publickeycredentialparametersReferenced in:  
8389 \* 5.3. Parameters for Credential Generation (dictionary  
8390 PublicKeyCredentialParameters)  
8391 \* 5.4. Options for Credential Creation (dictionary  
8392 PublicKeyCredentialCreationOptions) (2)  
8393  
8394 #dom-publickeycredentialparameters-typeReferenced in:  
8395 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8396 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
8397 \* 5.3. Parameters for Credential Generation (dictionary  
8398 PublicKeyCredentialParameters)  
8399  
8400 #dom-publickeycredentialparameters-algReferenced in:  
8401 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8402 [[Create]](origin, options, sameOriginWithAncestors) method  
8403 \* 5.3. Parameters for Credential Generation (dictionary  
8404 PublicKeyCredentialParameters)  
8405  
8406 #dictdef-publickeycredentialcreationoptionsReferenced in:  
8407 \* 5.1.1. CredentialCreationOptions Dictionary Extension  
8408 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8409 [[Create]](origin, options, sameOriginWithAncestors) method  
8410 \* 5.4. Options for Credential Creation (dictionary  
8411 PublicKeyCredentialCreationOptions)  
8412  
8413 #dom-publickeycredentialcreationoptions-rpReferenced in:

838E \* 5.1.3. Create a new credential - PublicKeyCredential's  
838E [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
839C (4) (5) (6)  
8391 \* 5.4. Options for Credential Creation (dictionary  
8392 PublicKeyCredentialCreationOptions)  
8393  
8394 #dom-publickeycredentialcreationoptions-userReferenced in:  
8395 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8396 [[Create]](origin, options, sameOriginWithAncestors) method  
8397 \* 5.4. Options for Credential Creation (dictionary  
8398 PublicKeyCredentialCreationOptions)  
8399 \* 7.1. Registering a new credential  
8400  
8401 #dom-publickeycredentialcreationoptions-challengeReferenced in:  
8402 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8403 [[Create]](origin, options, sameOriginWithAncestors) method  
8404 \* 5.4. Options for Credential Creation (dictionary  
8405 PublicKeyCredentialCreationOptions)  
8406 \* 13.1. Cryptographic Challenges  
8407  
8408 #dom-publickeycredentialcreationoptions-pubkeycredparamsReferenced in:  
8409 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8410 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
8411 \* 5.4. Options for Credential Creation (dictionary  
8412 PublicKeyCredentialCreationOptions)  
8413  
8414 #dom-publickeycredentialcreationoptions-timeoutReferenced in:  
8415 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8416 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
8417 \* 5.4. Options for Credential Creation (dictionary  
8418 PublicKeyCredentialCreationOptions)  
8419  
8420 #dom-publickeycredentialcreationoptions-excludecredentialsReferenced  
8421 in:  
8422 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8423 [[Create]](origin, options, sameOriginWithAncestors) method  
8424 \* 5.4. Options for Credential Creation (dictionary  
8425 PublicKeyCredentialCreationOptions)  
8426 \* 14.2. Registration Ceremony Privacy (2)  
8427  
8428 #dom-publickeycredentialcreationoptions-authenticatorselectionReference  
8429 d in:  
8430 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8431 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
8432 (4) (5) (6)  
8433 \* 5.4. Options for Credential Creation (dictionary  
8434 PublicKeyCredentialCreationOptions)  
8435 \* 6.2.2. The authenticatorMakeCredential operation  
8436  
8437 #dom-publickeycredentialcreationoptions-attestationReferenced in:  
8438 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8439 [[Create]](origin, options, sameOriginWithAncestors) method  
8440 \* 5.4. Options for Credential Creation (dictionary  
8441 PublicKeyCredentialCreationOptions)  
8442  
8443 #dom-publickeycredentialcreationoptions-extensionsReferenced in:  
8444 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8445 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
8446 \* 5.4. Options for Credential Creation (dictionary  
8447 PublicKeyCredentialCreationOptions)  
8448 \* 7.1. Registering a new credential (2)  
8449 \* 7.2. Verifying an authentication assertion  
8450 \* 9.3. Extending request parameters  
8451  
8452 #dictdef-publickeycredentialentityReferenced in:  
8453 \* 5.4.1. Public Key Entity Description (dictionary  
8454 PublicKeyCredentialEntity) (2) (3)  
8455 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
8456 PublicKeyCredentialRpEntity)  
8457 \* 5.4.3. User Account Parameters for Credential Generation

8414 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8415 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
8416 (4) (5) (6)  
8417 \* 5.4. Options for Credential Creation (dictionary  
8418 PublicKeyCredentialCreationOptions)  
8419  
8420 #dom-publickeycredentialcreationoptions-userReferenced in:  
8421 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8422 [[Create]](origin, options, sameOriginWithAncestors) method  
8423 \* 5.4. Options for Credential Creation (dictionary  
8424 PublicKeyCredentialCreationOptions)  
8425 \* 7.1. Registering a new credential  
8426  
8427 #dom-publickeycredentialcreationoptions-challengeReferenced in:  
8428 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8429 [[Create]](origin, options, sameOriginWithAncestors) method  
8430 \* 5.4. Options for Credential Creation (dictionary  
8431 PublicKeyCredentialCreationOptions)  
8432 \* 13.1. Cryptographic Challenges  
8433  
8434 #dom-publickeycredentialcreationoptions-pubkeycredparamsReferenced in:  
8435 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8436 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
8437 \* 5.4. Options for Credential Creation (dictionary  
8438 PublicKeyCredentialCreationOptions)  
8439  
8440 #dom-publickeycredentialcreationoptions-timeoutReferenced in:  
8441 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8442 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
8443 \* 5.4. Options for Credential Creation (dictionary  
8444 PublicKeyCredentialCreationOptions)  
8445  
8446 #dom-publickeycredentialcreationoptions-excludecredentialsReferenced  
8447 in:  
8448 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8449 [[Create]](origin, options, sameOriginWithAncestors) method  
8450 \* 5.4. Options for Credential Creation (dictionary  
8451 PublicKeyCredentialCreationOptions)  
8452 \* 14.2. Registration Ceremony Privacy (2)  
8453  
8454 #dom-publickeycredentialcreationoptions-authenticatorselectionReference  
8455 d in:  
8456 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8457 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
8458 (4) (5) (6)  
8459 \* 5.4. Options for Credential Creation (dictionary  
8460 PublicKeyCredentialCreationOptions)  
8461 \* 6.2.2. The authenticatorMakeCredential operation  
8462  
8463 #dom-publickeycredentialcreationoptions-attestationReferenced in:  
8464 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8465 [[Create]](origin, options, sameOriginWithAncestors) method  
8466 \* 5.4. Options for Credential Creation (dictionary  
8467 PublicKeyCredentialCreationOptions)  
8468  
8469 #dom-publickeycredentialcreationoptions-extensionsReferenced in:  
8470 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8471 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
8472 \* 5.4. Options for Credential Creation (dictionary  
8473 PublicKeyCredentialCreationOptions)  
8474 \* 7.1. Registering a new credential (2)  
8475 \* 7.2. Verifying an authentication assertion  
8476 \* 9.3. Extending request parameters  
8477  
8478 #dictdef-publickeycredentialentityReferenced in:  
8479 \* 5.4.1. Public Key Entity Description (dictionary  
8480 PublicKeyCredentialEntity) (2) (3)  
8481 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
8482 PublicKeyCredentialRpEntity)  
8483 \* 5.4.3. User Account Parameters for Credential Generation

845E (dictionary PublicKeyCredentialUserEntity)  
845F  
8460 #dom-publickeycredentialentity-nameReferenced in:  
8461 \* 5.4. Options for Credential Creation (dictionary  
8462 PublicKeyCredentialCreationOptions) (2)  
8463 \* 5.4.1. Public Key Entity Description (dictionary  
8464 PublicKeyCredentialEntity) (2) (3) (4)  
8465 \* 6.2.2. The authenticatorMakeCredential operation (2)  
8466  
8467 #dom-publickeycredentialentity-iconReferenced in:  
8468 \* 5.4.1. Public Key Entity Description (dictionary  
8469 PublicKeyCredentialEntity)  
8470  
8471 #dictdef-publickeycredentialrpentityReferenced in:  
8472 \* 5.4. Options for Credential Creation (dictionary  
8473 PublicKeyCredentialCreationOptions) (2)  
8474 \* 5.4.1. Public Key Entity Description (dictionary  
8475 PublicKeyCredentialEntity)  
8476 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
8477 PublicKeyCredentialRpEntity) (2)  
8478 \* 6.2.2. The authenticatorMakeCredential operation  
8479  
8480 #dom-publickeycredentialrpentity-idReferenced in:  
8481 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8482 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
8483 (4) (5)  
8484 \* 5.4. Options for Credential Creation (dictionary  
8485 PublicKeyCredentialCreationOptions)  
8486 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
8487 PublicKeyCredentialRpEntity)  
8488 \* 6.2.2. The authenticatorMakeCredential operation (2) (3) (4)  
8489  
8490 #dictdef-publickeycredentialuserentityReferenced in:  
8491 \* 5.4. Options for Credential Creation (dictionary  
8492 PublicKeyCredentialCreationOptions) (2)  
8493 \* 5.4.1. Public Key Entity Description (dictionary  
8494 PublicKeyCredentialEntity) (2)  
8495 \* 5.4.3. User Account Parameters for Credential Generation  
8496 (dictionary PublicKeyCredentialUserEntity) (2)  
8497 \* 6.2.2. The authenticatorMakeCredential operation  
8498  
8499 #dom-publickeycredentialuserentity-idReferenced in:  
8500 \* 5.4. Options for Credential Creation (dictionary  
8501 PublicKeyCredentialCreationOptions)  
8502 \* 5.4.3. User Account Parameters for Credential Generation  
8503 (dictionary PublicKeyCredentialUserEntity)  
8504 \* 6.2.2. The authenticatorMakeCredential operation  
8505  
8506 #dom-publickeycredentialuserentity-displaynameReferenced in:  
8507 \* 4. Terminology  
8508 \* 5.4. Options for Credential Creation (dictionary  
8509 PublicKeyCredentialCreationOptions)  
8510 \* 5.4.1. Public Key Entity Description (dictionary  
8511 PublicKeyCredentialEntity)  
8512 \* 5.4.3. User Account Parameters for Credential Generation  
8513 (dictionary PublicKeyCredentialUserEntity) (2) (3)  
8514 \* 6.2.2. The authenticatorMakeCredential operation  
8515  
8516 #dictdef-authenticatorselectioncriteriaReferenced in:  
8517 \* 5.4. Options for Credential Creation (dictionary  
8518 PublicKeyCredentialCreationOptions) (2)  
8519 \* 5.4.4. Authenticator Selection Criteria (dictionary  
8520 AuthenticatorSelectionCriteria) (2)  
8521  
8522 #dom-authenticatorselectioncriteria-authenticatorattachmentReferenced  
8523 in:  
8524 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8525 [[Create]](origin, options, sameOriginWithAncestors) method  
8526 \* 5.4.4. Authenticator Selection Criteria (dictionary  
8527 AuthenticatorSelectionCriteria)

8484 (dictionary PublicKeyCredentialUserEntity)  
8485  
8486 #dom-publickeycredentialentity-nameReferenced in:  
8487 \* 5.4. Options for Credential Creation (dictionary  
8488 PublicKeyCredentialCreationOptions) (2)  
8489 \* 5.4.1. Public Key Entity Description (dictionary  
8490 PublicKeyCredentialEntity) (2) (3) (4)  
8491 \* 6.2.2. The authenticatorMakeCredential operation (2)  
8492  
8493 #dom-publickeycredentialentity-iconReferenced in:  
8494 \* 5.4.1. Public Key Entity Description (dictionary  
8495 PublicKeyCredentialEntity)  
8496  
8497 #dictdef-publickeycredentialrpentityReferenced in:  
8498 \* 5.4. Options for Credential Creation (dictionary  
8499 PublicKeyCredentialCreationOptions) (2)  
8500 \* 5.4.1. Public Key Entity Description (dictionary  
8501 PublicKeyCredentialEntity)  
8502 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
8503 PublicKeyCredentialRpEntity) (2)  
8504 \* 6.2.2. The authenticatorMakeCredential operation  
8505  
8506 #dom-publickeycredentialrpentity-idReferenced in:  
8507 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8508 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
8509 (4) (5)  
8510 \* 5.4. Options for Credential Creation (dictionary  
8511 PublicKeyCredentialCreationOptions)  
8512 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
8513 PublicKeyCredentialRpEntity)  
8514 \* 6.2.2. The authenticatorMakeCredential operation (2) (3) (4)  
8515  
8516 #dictdef-publickeycredentialuserentityReferenced in:  
8517 \* 5.4. Options for Credential Creation (dictionary  
8518 PublicKeyCredentialCreationOptions) (2)  
8519 \* 5.4.1. Public Key Entity Description (dictionary  
8520 PublicKeyCredentialEntity) (2)  
8521 \* 5.4.3. User Account Parameters for Credential Generation  
8522 (dictionary PublicKeyCredentialUserEntity) (2)  
8523 \* 6.2.2. The authenticatorMakeCredential operation  
8524  
8525 #dom-publickeycredentialuserentity-idReferenced in:  
8526 \* 5.4. Options for Credential Creation (dictionary  
8527 PublicKeyCredentialCreationOptions)  
8528 \* 5.4.3. User Account Parameters for Credential Generation  
8529 (dictionary PublicKeyCredentialUserEntity)  
8530 \* 6.2.2. The authenticatorMakeCredential operation  
8531  
8532 #dom-publickeycredentialuserentity-displaynameReferenced in:  
8533 \* 4. Terminology  
8534 \* 5.4. Options for Credential Creation (dictionary  
8535 PublicKeyCredentialCreationOptions)  
8536 \* 5.4.1. Public Key Entity Description (dictionary  
8537 PublicKeyCredentialEntity)  
8538 \* 5.4.3. User Account Parameters for Credential Generation  
8539 (dictionary PublicKeyCredentialUserEntity) (2) (3)  
8540 \* 6.2.2. The authenticatorMakeCredential operation  
8541  
8542 #dictdef-authenticatorselectioncriteriaReferenced in:  
8543 \* 5.4. Options for Credential Creation (dictionary  
8544 PublicKeyCredentialCreationOptions) (2)  
8545 \* 5.4.4. Authenticator Selection Criteria (dictionary  
8546 AuthenticatorSelectionCriteria) (2)  
8547  
8548 #dom-authenticatorselectioncriteria-authenticatorattachmentReferenced  
8549 in:  
8550 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8551 [[Create]](origin, options, sameOriginWithAncestors) method  
8552 \* 5.4.4. Authenticator Selection Criteria (dictionary  
8553 AuthenticatorSelectionCriteria)

```

8528 #dom-authenticatorselectioncriteria-requireresidentkeyReferenced in:
8529 * 5.1.3. Create a new credential - PublicKeyCredential's
8530 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8531 * 5.4.4. Authenticator Selection Criteria (dictionary
8532 AuthenticatorSelectionCriteria)
8533 * 6.2.2. The authenticatorMakeCredential operation
8534
8535 #dom-authenticatorselectioncriteria-userverificationReferenced in:
8536 * 5.1.3. Create a new credential - PublicKeyCredential's
8537 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8538 * 5.4.4. Authenticator Selection Criteria (dictionary
8539 AuthenticatorSelectionCriteria)
8540
8541 #enumdef-authenticatorattachmentReferenced in:
8542 * 5.4.4. Authenticator Selection Criteria (dictionary
8543 AuthenticatorSelectionCriteria) (2)
8544 * 5.4.5. Authenticator Attachment enumeration (enum
8545 AuthenticatorAttachment) (2)
8546
8547 #attachment-modalityReferenced in:
8548 * 5.4.5. Authenticator Attachment enumeration (enum
8549 AuthenticatorAttachment) (2)
8550
8551 #platform-authenticatorsReferenced in:
8552 * 5.1.7. Availability of User-Verifying Platform Authenticator -
8553 PublicKeyCredential's
8554 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)
8555 (5)
8556 * 5.4.5. Authenticator Attachment enumeration (enum
8557 AuthenticatorAttachment) (2)
8558 * 12.1. Registration
8559 * 12.2. Registration Specifically with User Verifying Platform
8560 Authenticator (2)
8561 * 14.2. Registration Ceremony Privacy
8562
8563 #roaming-authenticatorsReferenced in:
8564 * 1.1.3. Other use cases and configurations
8565 * 4. Terminology
8566 * 5.4.5. Authenticator Attachment enumeration (enum
8567 AuthenticatorAttachment) (2)
8568 * 12.1. Registration
8569
8570 #platform-attachmentReferenced in:
8571 * 5.4.5. Authenticator Attachment enumeration (enum
8572 AuthenticatorAttachment)
8573
8574 #platform-credentialReferenced in:
8575 * 5.4.5. Authenticator Attachment enumeration (enum
8576 AuthenticatorAttachment) (2)
8577
8578 #cross-platform-attachedReferenced in:
8579 * 5.4.5. Authenticator Attachment enumeration (enum
8580 AuthenticatorAttachment) (2)
8581
8582 #roaming-credentialReferenced in:
8583 * 5.4.5. Authenticator Attachment enumeration (enum
8584 AuthenticatorAttachment) (2)
8585
8586 #attestation-conveyanceReferenced in:
8587 * 4. Terminology
8588 * 5.4. Options for Credential Creation (dictionary
8589 PublicKeyCredentialCreationOptions)
8590 * 5.4.6. Attestation Conveyance Preference enumeration (enum
8591 AttestationConveyancePreference)
8592
8593 #enumdef-attestationconveyancepreferenceReferenced in:
8594 * 5.4. Options for Credential Creation (dictionary
8595 PublicKeyCredentialCreationOptions) (2)
8596 * 5.4.6. Attestation Conveyance Preference enumeration (enum
8597

```

```

8554 #dom-authenticatorselectioncriteria-requireresidentkeyReferenced in:
8555 * 5.1.3. Create a new credential - PublicKeyCredential's
8556 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8557 * 5.4.4. Authenticator Selection Criteria (dictionary
8558 AuthenticatorSelectionCriteria)
8559 * 6.2.2. The authenticatorMakeCredential operation
8560
8561 #dom-authenticatorselectioncriteria-userverificationReferenced in:
8562 * 5.1.3. Create a new credential - PublicKeyCredential's
8563 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8564 * 5.4.4. Authenticator Selection Criteria (dictionary
8565 AuthenticatorSelectionCriteria)
8566
8567 #enumdef-authenticatorattachmentReferenced in:
8568 * 5.4.4. Authenticator Selection Criteria (dictionary
8569 AuthenticatorSelectionCriteria) (2)
8570 * 5.4.5. Authenticator Attachment enumeration (enum
8571 AuthenticatorAttachment) (2)
8572
8573 #attachment-modalityReferenced in:
8574 * 5.4.5. Authenticator Attachment enumeration (enum
8575 AuthenticatorAttachment) (2)
8576
8577 #platform-authenticatorsReferenced in:
8578 * 5.1.7. Availability of User-Verifying Platform Authenticator -
8579 PublicKeyCredential's
8580 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)
8581 (5)
8582 * 5.4.5. Authenticator Attachment enumeration (enum
8583 AuthenticatorAttachment) (2)
8584 * 12.1. Registration
8585 * 12.2. Registration Specifically with User Verifying Platform
8586 Authenticator (2)
8587 * 14.2. Registration Ceremony Privacy
8588
8589 #roaming-authenticatorsReferenced in:
8590 * 1.1.3. Other use cases and configurations
8591 * 4. Terminology
8592 * 5.4.5. Authenticator Attachment enumeration (enum
8593 AuthenticatorAttachment) (2)
8594 * 12.1. Registration
8595
8596 #platform-attachmentReferenced in:
8597 * 5.4.5. Authenticator Attachment enumeration (enum
8598 AuthenticatorAttachment)
8599
8600 #platform-credentialReferenced in:
8601 * 5.4.5. Authenticator Attachment enumeration (enum
8602 AuthenticatorAttachment) (2)
8603
8604 #cross-platform-attachedReferenced in:
8605 * 5.4.5. Authenticator Attachment enumeration (enum
8606 AuthenticatorAttachment) (2)
8607
8608 #roaming-credentialReferenced in:
8609 * 5.4.5. Authenticator Attachment enumeration (enum
8610 AuthenticatorAttachment) (2)
8611
8612 #attestation-conveyanceReferenced in:
8613 * 4. Terminology
8614 * 5.4. Options for Credential Creation (dictionary
8615 PublicKeyCredentialCreationOptions)
8616 * 5.4.6. Attestation Conveyance Preference enumeration (enum
8617 AttestationConveyancePreference)
8618
8619 #enumdef-attestationconveyancepreferenceReferenced in:
8620 * 5.4. Options for Credential Creation (dictionary
8621 PublicKeyCredentialCreationOptions) (2)
8622 * 5.4.6. Attestation Conveyance Preference enumeration (enum
8623

```

859f AttestationConveyancePreference) (2)  
859f  
860c #dom-attestationconveyancepreference-noneReferenced in:  
8601 \* 5.4. Options for Credential Creation (dictionary  
8602 PublicKeyCredentialCreationOptions)  
8603 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
8604 AttestationConveyancePreference)  
8605  
8606 #dom-attestationconveyancepreference-indirectReferenced in:  
8607 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
8608 AttestationConveyancePreference)  
8609  
861c #dom-attestationconveyancepreference-directReferenced in:  
8611 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
8612 AttestationConveyancePreference)  
8613  
8614 #dictdef-publickeycredentialrequestoptionsReferenced in:  
8615 \* 5.1.2. CredentialRequestOptions Dictionary Extension  
8616 \* 5.1.4.1. PublicKeyCredential's  
8617 [[DiscoverFromExternalSource]](origin, options,  
8618 sameOriginWithAncestors) method  
8619 \* 5.5. Options for Assertion Generation (dictionary  
8620 PublicKeyCredentialRequestOptions) (2)  
8621 \* 7.2. Verifying an authentication assertion  
8622  
8623 #dom-publickeycredentialrequestoptions-challengeReferenced in:  
8624 \* 5.1.4.1. PublicKeyCredential's  
8625 [[DiscoverFromExternalSource]](origin, options,  
8626 sameOriginWithAncestors) method  
8627 \* 5.5. Options for Assertion Generation (dictionary  
8628 PublicKeyCredentialRequestOptions) (2)  
8629 \* 13.1. Cryptographic Challenges  
8630  
8631 #dom-publickeycredentialrequestoptions-timeoutReferenced in:  
8632 \* 5.1.4.1. PublicKeyCredential's  
8633 [[DiscoverFromExternalSource]](origin, options,  
8634 sameOriginWithAncestors) method (2)  
8635 \* 5.5. Options for Assertion Generation (dictionary  
8636 PublicKeyCredentialRequestOptions)  
8637  
8638 #dom-publickeycredentialrequestoptions-rpidReferenced in:  
8639 \* 5.1.4.1. PublicKeyCredential's  
8640 [[DiscoverFromExternalSource]](origin, options,  
8641 sameOriginWithAncestors) method (2) (3) (4)  
8642 \* 5.5. Options for Assertion Generation (dictionary  
8643 PublicKeyCredentialRequestOptions)  
8644  
8645 #dom-publickeycredentialrequestoptions-allowcredentialsReferenced in:  
8646 \* 5.1.4.1. PublicKeyCredential's  
8647 [[DiscoverFromExternalSource]](origin, options,  
8648 sameOriginWithAncestors) method (2) (3) (4)  
8649 \* 5.5. Options for Assertion Generation (dictionary  
8650 PublicKeyCredentialRequestOptions)  
8651 \* 7.2. Verifying an authentication assertion (2)  
8652 \* 14.3. Authentication Ceremony Privacy (2)  
8653  
8654 #dom-publickeycredentialrequestoptions-userverificationReferenced in:  
8655 \* 5.1.4.1. PublicKeyCredential's  
8656 [[DiscoverFromExternalSource]](origin, options,  
8657 sameOriginWithAncestors) method (2)  
8658 \* 5.5. Options for Assertion Generation (dictionary  
8659 PublicKeyCredentialRequestOptions)  
8660  
8661 #dom-publickeycredentialrequestoptions-extensionsReferenced in:  
8662 \* 5.1.4.1. PublicKeyCredential's  
8663 [[DiscoverFromExternalSource]](origin, options,  
8664 sameOriginWithAncestors) method (2)  
8665 \* 5.5. Options for Assertion Generation (dictionary  
8666 PublicKeyCredentialRequestOptions)  
8667 \* 7.2. Verifying an authentication assertion

8624 AttestationConveyancePreference) (2)  
8625  
8626 #dom-attestationconveyancepreference-noneReferenced in:  
8627 \* 5.4. Options for Credential Creation (dictionary  
8628 PublicKeyCredentialCreationOptions)  
8629 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
8630 AttestationConveyancePreference)  
8631  
8632 #dom-attestationconveyancepreference-indirectReferenced in:  
8633 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
8634 AttestationConveyancePreference)  
8635  
8636 #dom-attestationconveyancepreference-directReferenced in:  
8637 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
8638 AttestationConveyancePreference)  
8639  
8640 #dictdef-publickeycredentialrequestoptionsReferenced in:  
8641 \* 5.1.2. CredentialRequestOptions Dictionary Extension  
8642 \* 5.1.4.1. PublicKeyCredential's  
8643 [[DiscoverFromExternalSource]](origin, options,  
8644 sameOriginWithAncestors) method  
8645 \* 5.5. Options for Assertion Generation (dictionary  
8646 PublicKeyCredentialRequestOptions) (2)  
8647 \* 7.2. Verifying an authentication assertion  
8648  
8649 #dom-publickeycredentialrequestoptions-challengeReferenced in:  
8650 \* 5.1.4.1. PublicKeyCredential's  
8651 [[DiscoverFromExternalSource]](origin, options,  
8652 sameOriginWithAncestors) method  
8653 \* 5.5. Options for Assertion Generation (dictionary  
8654 PublicKeyCredentialRequestOptions) (2)  
8655 \* 13.1. Cryptographic Challenges  
8656  
8657 #dom-publickeycredentialrequestoptions-timeoutReferenced in:  
8658 \* 5.1.4.1. PublicKeyCredential's  
8659 [[DiscoverFromExternalSource]](origin, options,  
8660 sameOriginWithAncestors) method (2)  
8661 \* 5.5. Options for Assertion Generation (dictionary  
8662 PublicKeyCredentialRequestOptions)  
8663  
8664 #dom-publickeycredentialrequestoptions-rpidReferenced in:  
8665 \* 5.1.4.1. PublicKeyCredential's  
8666 [[DiscoverFromExternalSource]](origin, options,  
8667 sameOriginWithAncestors) method (2) (3) (4)  
8668 \* 5.5. Options for Assertion Generation (dictionary  
8669 PublicKeyCredentialRequestOptions)  
8670  
8671 #dom-publickeycredentialrequestoptions-allowcredentialsReferenced in:  
8672 \* 5.1.4.1. PublicKeyCredential's  
8673 [[DiscoverFromExternalSource]](origin, options,  
8674 sameOriginWithAncestors) method (2) (3) (4) (5) (6)  
8675 \* 5.5. Options for Assertion Generation (dictionary  
8676 PublicKeyCredentialRequestOptions)  
8677 \* 7.2. Verifying an authentication assertion (2)  
8678 \* 14.3. Authentication Ceremony Privacy (2)  
8679  
8680 #dom-publickeycredentialrequestoptions-userverificationReferenced in:  
8681 \* 5.1.4.1. PublicKeyCredential's  
8682 [[DiscoverFromExternalSource]](origin, options,  
8683 sameOriginWithAncestors) method (2)  
8684 \* 5.5. Options for Assertion Generation (dictionary  
8685 PublicKeyCredentialRequestOptions)  
8686  
8687 #dom-publickeycredentialrequestoptions-extensionsReferenced in:  
8688 \* 5.1.4.1. PublicKeyCredential's  
8689 [[DiscoverFromExternalSource]](origin, options,  
8690 sameOriginWithAncestors) method (2)  
8691 \* 5.5. Options for Assertion Generation (dictionary  
8692 PublicKeyCredentialRequestOptions)  
8693 \* 7.2. Verifying an authentication assertion

```

8668 #dictdef-authenticationextensionsclientinputsReferenced in:
8669 * 5.4. Options for Credential Creation (dictionary
8670 PublicKeyCredentialCreationOptions) (2)
8671 * 5.5. Options for Assertion Generation (dictionary
8672 PublicKeyCredentialRequestOptions) (2)
8673 * 10.1. FIDO AppID Extension (appid)
8674 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
8675 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
8676 * 10.4. Authenticator Selection Extension (authnSel)
8677 * 10.5. Supported Extensions Extension (exts)
8678 * 10.6. User Verification Index Extension (uvi)
8679 * 10.7. Location Extension (loc)
8680 * 10.8. User Verification Method Extension (uvm)
8681
8682 #dictdef-authenticationextensionsclientoutputsReferenced in:
8683 * 5.1. PublicKeyCredential Interface
8684 * 5.1.3. Create a new credential - PublicKeyCredential's
8685 [[Create]](origin, options, sameOriginWithAncestors) method
8686 * 5.1.4.1. PublicKeyCredential's
8687 [[DiscoverFromExternalSource]](origin, options,
8688 sameOriginWithAncestors) method
8689 * 7.1. Registering a new credential
8690 * 7.2. Verifying an authentication assertion
8691 * 10.1. FIDO AppID Extension (appid)
8692 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
8693 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
8694 * 10.4. Authenticator Selection Extension (authnSel)
8695 * 10.5. Supported Extensions Extension (exts)
8696 * 10.6. User Verification Index Extension (uvi)
8697 * 10.7. Location Extension (loc)
8698 * 10.8. User Verification Method Extension (uvm)
8699
8700 #dictdef-collectedclientdataReferenced in:
8701 * 5.1.3. Create a new credential - PublicKeyCredential's
8702 [[Create]](origin, options, sameOriginWithAncestors) method
8703 * 5.1.4.1. PublicKeyCredential's
8704 [[DiscoverFromExternalSource]](origin, options,
8705 sameOriginWithAncestors) method
8706 * 5.10.1. Client data used in WebAuthn signatures (dictionary
8707 CollectedClientData) (2) (3)
8708
8709 #client-dataReferenced in:
8710 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
8711 * 6. WebAuthn Authenticator Model (2) (3) (4)
8712 * 6.1. Authenticator data (2)
8713 * 7.1. Registering a new credential
8714 * 7.2. Verifying an authentication assertion
8715 * 9. WebAuthn Extensions
8716
8717 #dictdef-tokenbindingReferenced in:
8718 * 5.10.1. Client data used in WebAuthn signatures (dictionary
8719 CollectedClientData)
8720
8721 #dom-tokenbinding-statusReferenced in:
8722 * 7.1. Registering a new credential
8723 * 7.2. Verifying an authentication assertion
8724
8725 #dom-tokenbinding-idReferenced in:
8726 * 7.1. Registering a new credential
8727 * 7.2. Verifying an authentication assertion
8728
8729 #enumdef-tokenbindingstatusReferenced in:
8730 * 5.10.1. Client data used in WebAuthn signatures (dictionary
8731 CollectedClientData)
8732
8733 #dom-collectedclientdata-typeReferenced in:
8734 * 5.1.3. Create a new credential - PublicKeyCredential's
8735 [[Create]](origin, options, sameOriginWithAncestors) method
8736 * 5.1.4.1. PublicKeyCredential's
8737

```

```

8694 #dictdef-authenticationextensionsclientinputsReferenced in:
8695 * 5.4. Options for Credential Creation (dictionary
8696 PublicKeyCredentialCreationOptions) (2)
8697 * 5.5. Options for Assertion Generation (dictionary
8698 PublicKeyCredentialRequestOptions) (2)
8699 * 10.1. FIDO AppID Extension (appid)
8700 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
8701 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
8702 * 10.4. Authenticator Selection Extension (authnSel)
8703 * 10.5. Supported Extensions Extension (exts)
8704 * 10.6. User Verification Index Extension (uvi)
8705 * 10.7. Location Extension (loc)
8706 * 10.8. User Verification Method Extension (uvm)
8707
8708 #dictdef-authenticationextensionsclientoutputsReferenced in:
8709 * 5.1. PublicKeyCredential Interface
8710 * 5.1.3. Create a new credential - PublicKeyCredential's
8711 [[Create]](origin, options, sameOriginWithAncestors) method
8712 * 5.1.4.1. PublicKeyCredential's
8713 [[DiscoverFromExternalSource]](origin, options,
8714 sameOriginWithAncestors) method
8715 * 7.1. Registering a new credential
8716 * 7.2. Verifying an authentication assertion
8717 * 10.1. FIDO AppID Extension (appid)
8718 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
8719 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
8720 * 10.4. Authenticator Selection Extension (authnSel)
8721 * 10.5. Supported Extensions Extension (exts)
8722 * 10.6. User Verification Index Extension (uvi)
8723 * 10.7. Location Extension (loc)
8724 * 10.8. User Verification Method Extension (uvm)
8725
8726 #dictdef-collectedclientdataReferenced in:
8727 * 5.1.3. Create a new credential - PublicKeyCredential's
8728 [[Create]](origin, options, sameOriginWithAncestors) method
8729 * 5.1.4.1. PublicKeyCredential's
8730 [[DiscoverFromExternalSource]](origin, options,
8731 sameOriginWithAncestors) method
8732 * 5.10.1. Client data used in WebAuthn signatures (dictionary
8733 CollectedClientData) (2) (3)
8734
8735 #client-dataReferenced in:
8736 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
8737 * 6. WebAuthn Authenticator Model (2) (3) (4)
8738 * 6.1. Authenticator data (2)
8739 * 7.1. Registering a new credential
8740 * 7.2. Verifying an authentication assertion
8741 * 9. WebAuthn Extensions
8742
8743 #dictdef-tokenbindingReferenced in:
8744 * 5.10.1. Client data used in WebAuthn signatures (dictionary
8745 CollectedClientData)
8746
8747 #dom-tokenbinding-statusReferenced in:
8748 * 7.1. Registering a new credential
8749 * 7.2. Verifying an authentication assertion
8750
8751 #dom-tokenbinding-idReferenced in:
8752 * 7.1. Registering a new credential
8753 * 7.2. Verifying an authentication assertion
8754
8755 #enumdef-tokenbindingstatusReferenced in:
8756 * 5.10.1. Client data used in WebAuthn signatures (dictionary
8757 CollectedClientData)
8758
8759 #dom-collectedclientdata-typeReferenced in:
8760 * 5.1.3. Create a new credential - PublicKeyCredential's
8761 [[Create]](origin, options, sameOriginWithAncestors) method
8762 * 5.1.4.1. PublicKeyCredential's
8763

```

873E [[DiscoverFromExternalSource]](origin, options,  
873F sameOriginWithAncestors) method  
874C \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
8741 CollectedClientData)  
8742 \* 7.1. Registering a new credential  
8743 \* 7.2. Verifying an authentication assertion  
8744  
8745 #dom-collectedclientdata-challengeReferenced in:  
8746 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8747 [[Create]](origin, options, sameOriginWithAncestors) method  
8748 \* 5.1.4.1. PublicKeyCredential's  
8749 [[DiscoverFromExternalSource]](origin, options,  
8750 sameOriginWithAncestors) method  
8751 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
8752 CollectedClientData)  
8753 \* 7.1. Registering a new credential  
8754 \* 7.2. Verifying an authentication assertion  
8755  
8756 #dom-collectedclientdata-originReferenced in:  
8757 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8758 [[Create]](origin, options, sameOriginWithAncestors) method  
8759 \* 5.1.4.1. PublicKeyCredential's  
8760 [[DiscoverFromExternalSource]](origin, options,  
8761 sameOriginWithAncestors) method  
8762 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
8763 CollectedClientData)  
8764 \* 7.1. Registering a new credential  
8765 \* 7.2. Verifying an authentication assertion  
8766  
8767 #dom-collectedclientdata-tokenbindingReferenced in:  
8768 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8769 [[Create]](origin, options, sameOriginWithAncestors) method  
8770 \* 5.1.4.1. PublicKeyCredential's  
8771 [[DiscoverFromExternalSource]](origin, options,  
8772 sameOriginWithAncestors) method  
8773 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
8774 CollectedClientData)  
8775 \* 7.1. Registering a new credential (2)  
8776 \* 7.2. Verifying an authentication assertion (2)  
8777  
8778 #collectedclientdata-json-serialized-client-dataReferenced in:  
8779 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8780 [[Create]](origin, options, sameOriginWithAncestors) method  
8781 \* 5.1.4.1. PublicKeyCredential's  
8782 [[DiscoverFromExternalSource]](origin, options,  
8783 sameOriginWithAncestors) method  
8784 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
8785 \* 5.2.1. Information about Public Key Credential (interface  
8786 AuthenticatorAttestationResponse) (2)  
8787 \* 5.2.2. Web Authentication Assertion (interface  
8788 AuthenticatorAssertionResponse)  
8789 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
8790 CollectedClientData)  
8791  
8792 #collectedclientdata-hash-of-the-serialized-client-dataReferenced in:  
8793 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8794 [[Create]](origin, options, sameOriginWithAncestors) method  
8795 \* 5.1.4.1. PublicKeyCredential's  
8796 [[DiscoverFromExternalSource]](origin, options,  
8797 sameOriginWithAncestors) method  
8798 \* 5.2.1. Information about Public Key Credential (interface  
8799 AuthenticatorAttestationResponse)  
8800 \* 5.2.2. Web Authentication Assertion (interface  
8801 AuthenticatorAssertionResponse)  
8802 \* 6. WebAuthn Authenticator Model  
8803 \* 6.1.2. FIDO U2F signature format compatibility (2)  
8804 \* 6.2.2. The authenticatorMakeCredential operation  
8805 \* 6.2.3. The authenticatorGetAssertion operation (2)  
8806 \* 6.3.2. Attestation Statement Formats (2)  
8807 \* 6.3.4. Generating an Attestation Object

8764 [[DiscoverFromExternalSource]](origin, options,  
8765 sameOriginWithAncestors) method  
8766 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
8767 CollectedClientData)  
8768 \* 7.1. Registering a new credential  
8769 \* 7.2. Verifying an authentication assertion  
8770  
8771 #dom-collectedclientdata-challengeReferenced in:  
8772 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8773 [[Create]](origin, options, sameOriginWithAncestors) method  
8774 \* 5.1.4.1. PublicKeyCredential's  
8775 [[DiscoverFromExternalSource]](origin, options,  
8776 sameOriginWithAncestors) method  
8777 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
8778 CollectedClientData)  
8779 \* 7.1. Registering a new credential  
8780 \* 7.2. Verifying an authentication assertion  
8781  
8782 #dom-collectedclientdata-originReferenced in:  
8783 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8784 [[Create]](origin, options, sameOriginWithAncestors) method  
8785 \* 5.1.4.1. PublicKeyCredential's  
8786 [[DiscoverFromExternalSource]](origin, options,  
8787 sameOriginWithAncestors) method  
8788 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
8789 CollectedClientData)  
8790 \* 7.1. Registering a new credential  
8791 \* 7.2. Verifying an authentication assertion  
8792  
8793 #dom-collectedclientdata-tokenbindingReferenced in:  
8794 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8795 [[Create]](origin, options, sameOriginWithAncestors) method  
8796 \* 5.1.4.1. PublicKeyCredential's  
8797 [[DiscoverFromExternalSource]](origin, options,  
8798 sameOriginWithAncestors) method  
8799 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
8800 CollectedClientData)  
8801 \* 7.1. Registering a new credential (2)  
8802 \* 7.2. Verifying an authentication assertion (2)  
8803  
8804 #collectedclientdata-json-serialized-client-dataReferenced in:  
8805 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8806 [[Create]](origin, options, sameOriginWithAncestors) method  
8807 \* 5.1.4.1. PublicKeyCredential's  
8808 [[DiscoverFromExternalSource]](origin, options,  
8809 sameOriginWithAncestors) method  
8810 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
8811 \* 5.2.1. Information about Public Key Credential (interface  
8812 AuthenticatorAttestationResponse) (2)  
8813 \* 5.2.2. Web Authentication Assertion (interface  
8814 AuthenticatorAssertionResponse)  
8815 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
8816 CollectedClientData)  
8817  
8818 #collectedclientdata-hash-of-the-serialized-client-dataReferenced in:  
8819 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8820 [[Create]](origin, options, sameOriginWithAncestors) method  
8821 \* 5.1.4.1. PublicKeyCredential's  
8822 [[DiscoverFromExternalSource]](origin, options,  
8823 sameOriginWithAncestors) method  
8824 \* 5.2.1. Information about Public Key Credential (interface  
8825 AuthenticatorAttestationResponse)  
8826 \* 5.2.2. Web Authentication Assertion (interface  
8827 AuthenticatorAssertionResponse)  
8828 \* 6. WebAuthn Authenticator Model  
8829 \* 6.1.2. FIDO U2F signature format compatibility (2)  
8830 \* 6.2.2. The authenticatorMakeCredential operation  
8831 \* 6.2.3. The authenticatorGetAssertion operation (2)  
8832 \* 6.3.2. Attestation Statement Formats (2)  
8833 \* 6.3.4. Generating an Attestation Object

```

880E * 7.1. Registering a new credential
880F * 8.2. Packed Attestation Statement Format
8810 * 8.3. TPM Attestation Statement Format
8811 * 8.4. Android Key Attestation Statement Format
8812 * 8.5. Android SafetyNet Attestation Statement Format
8813 * 8.6. FIDO U2F Attestation Statement Format
8814
881E #enumdef-publickeycredentialtypeReferenced in:
881F * 4. Terminology
8820 * 5.1.3. Create a new credential - PublicKeyCredential's
8821 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8822 * 5.3. Parameters for Credential Generation (dictionary
8823 PublicKeyCredentialParameters)
8824 * 5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)
8825 * 5.10.3. Credential Descriptor (dictionary
8826 PublicKeyCredentialDescriptor)
8827 * 6.2.2. The authenticatorMakeCredential operation (2) (3)
882E
882F #dom-publickeycredentialtype-public-keyReferenced in:
8830 * 4. Terminology
8831 * 5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)
8832 * 6.2.2. The authenticatorMakeCredential operation
8833
8834 #dictdef-publickeycredentialdescriptorReferenced in:
8835 * 5.1.4.1. PublicKeyCredential's
8836 [[DiscoverFromExternalSource]](origin, options,
8837 sameOriginWithAncestors) method
8838 * 5.4. Options for Credential Creation (dictionary
8839 PublicKeyCredentialCreationOptions) (2)
8840 * 5.5. Options for Assertion Generation (dictionary
8841 PublicKeyCredentialRequestOptions) (2) (3)
8842 * 5.10.3. Credential Descriptor (dictionary
8843 PublicKeyCredentialDescriptor)
8844 * 6.2.2. The authenticatorMakeCredential operation
8845 * 6.2.3. The authenticatorGetAssertion operation
8846
8847 #dom-publickeycredentialdescriptor-typeReferenced in:
8848 * 5.1.4.1. PublicKeyCredential's
8849 [[DiscoverFromExternalSource]](origin, options,
8850 sameOriginWithAncestors) method
8851 * 5.10.3. Credential Descriptor (dictionary
8852 PublicKeyCredentialDescriptor)
8853 * 6.2.2. The authenticatorMakeCredential operation
8854
8855 #dom-publickeycredentialdescriptor-idReferenced in:
8856 * 5.1.4.1. PublicKeyCredential's
8857 [[DiscoverFromExternalSource]](origin, options,
8858 sameOriginWithAncestors) method (2)
8859 * 5.10.3. Credential Descriptor (dictionary
8860 PublicKeyCredentialDescriptor)
8861 * 6.2.2. The authenticatorMakeCredential operation
8862 * 6.2.3. The authenticatorGetAssertion operation
8863
8864 #dom-publickeycredentialdescriptor-transportReferenced in:
8865 * 5.1.3. Create a new credential - PublicKeyCredential's
8866 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8867 * 5.1.4.1. PublicKeyCredential's
8868 [[DiscoverFromExternalSource]](origin, options,
8869 sameOriginWithAncestors) method (2)
8870 * 5.10.3. Credential Descriptor (dictionary
8871 PublicKeyCredentialDescriptor)
8872
8873 #enumdef-authenticatortransportReferenced in:
8874 * 5.10.3. Credential Descriptor (dictionary
8875 PublicKeyCredentialDescriptor)
8876 * 5.10.4. Authenticator Transport enumeration (enum
8877 AuthenticatorTransport)
8878
8879 #dom-authenticatortransport-usbReferenced in:
8880 * 5.10.4. Authenticator Transport enumeration (enum

```

```

8834 * 7.1. Registering a new credential
8835 * 8.2. Packed Attestation Statement Format
8836 * 8.3. TPM Attestation Statement Format
8837 * 8.4. Android Key Attestation Statement Format
8838 * 8.5. Android SafetyNet Attestation Statement Format
8839 * 8.6. FIDO U2F Attestation Statement Format
8840
8841 #enumdef-publickeycredentialtypeReferenced in:
8842 * 4. Terminology
8843 * 5.1.3. Create a new credential - PublicKeyCredential's
8844 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8845 * 5.3. Parameters for Credential Generation (dictionary
8846 PublicKeyCredentialParameters)
8847 * 5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)
8848 * 5.10.3. Credential Descriptor (dictionary
8849 PublicKeyCredentialDescriptor)
8850 * 6.2.2. The authenticatorMakeCredential operation (2) (3)
8851
8852 #dom-publickeycredentialtype-public-keyReferenced in:
8853 * 4. Terminology
8854 * 5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)
8855 * 6.2.2. The authenticatorMakeCredential operation
8856
8857 #dictdef-publickeycredentialdescriptorReferenced in:
8858
8859 * 5.4. Options for Credential Creation (dictionary
8860 PublicKeyCredentialCreationOptions) (2)
8861 * 5.5. Options for Assertion Generation (dictionary
8862 PublicKeyCredentialRequestOptions) (2) (3)
8863 * 5.10.3. Credential Descriptor (dictionary
8864 PublicKeyCredentialDescriptor)
8865 * 6.2.2. The authenticatorMakeCredential operation
8866 * 6.2.3. The authenticatorGetAssertion operation
8867
8868 #dom-publickeycredentialdescriptor-typeReferenced in:
8869 * 5.1.4.1. PublicKeyCredential's
8870 [[DiscoverFromExternalSource]](origin, options,
8871 sameOriginWithAncestors) method
8872 * 5.10.3. Credential Descriptor (dictionary
8873 PublicKeyCredentialDescriptor)
8874 * 6.2.2. The authenticatorMakeCredential operation
8875
8876 #dom-publickeycredentialdescriptor-idReferenced in:
8877 * 5.1.4.1. PublicKeyCredential's
8878 [[DiscoverFromExternalSource]](origin, options,
8879 sameOriginWithAncestors) method
8880 * 5.10.3. Credential Descriptor (dictionary
8881 PublicKeyCredentialDescriptor)
8882 * 6.2.2. The authenticatorMakeCredential operation
8883 * 6.2.3. The authenticatorGetAssertion operation
8884
8885 #dom-publickeycredentialdescriptor-transportReferenced in:
8886 * 5.1.3. Create a new credential - PublicKeyCredential's
8887 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8888 * 5.1.4.1. PublicKeyCredential's
8889 [[DiscoverFromExternalSource]](origin, options,
8890 sameOriginWithAncestors) method (2) (3)
8891 * 5.10.3. Credential Descriptor (dictionary
8892 PublicKeyCredentialDescriptor)
8893
8894 #enumdef-authenticatortransportReferenced in:
8895 * 5.10.3. Credential Descriptor (dictionary
8896 PublicKeyCredentialDescriptor)
8897 * 5.10.4. Authenticator Transport enumeration (enum
8898 AuthenticatorTransport)
8899
8900 #dom-authenticatortransport-usbReferenced in:
8901 * 5.10.4. Authenticator Transport enumeration (enum

```



```

887E AuthenticatorTransport)
887F
888C #dom-authenticatortransport-nfcReferenced in:
888D * 5.10.4. Authenticator Transport enumeration (enum
888E AuthenticatorTransport)
888F
8884 #dom-authenticatortransport-bleReferenced in:
8885 * 5.10.4. Authenticator Transport enumeration (enum
8886 AuthenticatorTransport)
8887
888E #dom-authenticatortransport-internalReferenced in:
888F * 5.10.4. Authenticator Transport enumeration (enum
8890 AuthenticatorTransport)
8891
8892 #typedefdef-cosealgorithmidentifierReferenced in:
8893 * 5.1.3. Create a new credential - PublicKeyCredential's
8894 [[Create]](origin, options, sameOriginWithAncestors) method
8895 * 5.3. Parameters for Credential Generation (dictionary
8896 PublicKeyCredentialParameters)
8897 * 5.10.5. Cryptographic Algorithm Identifier (typedef
8898 COSEAlgorithmIdentifier)
8899 * 6.2.2. The authenticatorMakeCredential operation
8900 * 6.3.1. Attested credential data
8901 * 8.2. Packed Attestation Statement Format
8902 * 8.3. TPM Attestation Statement Format
8903
8904 #enumdef-userverificationrequirementReferenced in:
8905 * 5.4.4. Authenticator Selection Criteria (dictionary
8906 AuthenticatorSelectionCriteria) (2)
8907 * 5.5. Options for Assertion Generation (dictionary
8908 PublicKeyCredentialRequestOptions) (2)
8909 * 5.10.6. User Verification Requirement enumeration (enum
8910 UserVerificationRequirement)
8911
8912 #dom-userverificationrequirement-requiredReferenced in:
8913 * 5.1.3. Create a new credential - PublicKeyCredential's
8914 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8915 * 5.1.4.1. PublicKeyCredential's
8916 [[DiscoverFromExternalSource]](origin, options,
8917 sameOriginWithAncestors) method (2)
8918 * 5.10.6. User Verification Requirement enumeration (enum
8919 UserVerificationRequirement)
8920
8921 #dom-userverificationrequirement-preferredReferenced in:
8922 * 5.1.3. Create a new credential - PublicKeyCredential's
8923 [[Create]](origin, options, sameOriginWithAncestors) method
8924 * 5.1.4.1. PublicKeyCredential's
8925 [[DiscoverFromExternalSource]](origin, options,
8926 sameOriginWithAncestors) method
8927 * 5.10.6. User Verification Requirement enumeration (enum
8928 UserVerificationRequirement)
8929
8930 #dom-userverificationrequirement-discouragedReferenced in:
8931 * 5.1.3. Create a new credential - PublicKeyCredential's
8932 [[Create]](origin, options, sameOriginWithAncestors) method
8933 * 5.1.4.1. PublicKeyCredential's
8934 [[DiscoverFromExternalSource]](origin, options,
8935 sameOriginWithAncestors) method
8936 * 5.10.6. User Verification Requirement enumeration (enum
8937 UserVerificationRequirement)
8938
8939 #authenticator-modelReferenced in:
8940 * 6. WebAuthn Authenticator Model
8941
8942 #authenticator-credentials-mapReferenced in:
8943 * 6.2.1. Lookup Credential Source by Credential ID algorithm
8944 * 6.2.2. The authenticatorMakeCredential operation
8945 * 6.2.3. The authenticatorGetAssertion operation
8946
8947 #attestation-signatureReferenced in:

```

```

8901 AuthenticatorTransport)
8902
890C #dom-authenticatortransport-nfcReferenced in:
890D * 5.10.4. Authenticator Transport enumeration (enum
890E AuthenticatorTransport)
890F
8907 #dom-authenticatortransport-bleReferenced in:
8908 * 5.10.4. Authenticator Transport enumeration (enum
8909 AuthenticatorTransport)
8910
8911 #typedefdef-cosealgorithmidentifierReferenced in:
8912 * 5.1.3. Create a new credential - PublicKeyCredential's
8913 [[Create]](origin, options, sameOriginWithAncestors) method
8914 * 5.3. Parameters for Credential Generation (dictionary
8915 PublicKeyCredentialParameters)
8916 * 5.10.5. Cryptographic Algorithm Identifier (typedef
8917 COSEAlgorithmIdentifier)
8918 * 6.2.2. The authenticatorMakeCredential operation
8919 * 6.3.1. Attested credential data
8920 * 8.2. Packed Attestation Statement Format
8921 * 8.3. TPM Attestation Statement Format
8922
892C #enumdef-userverificationrequirementReferenced in:
892D * 5.4.4. Authenticator Selection Criteria (dictionary
892E AuthenticatorSelectionCriteria) (2)
892F * 5.5. Options for Assertion Generation (dictionary
8930 PublicKeyCredentialRequestOptions) (2)
8931 * 5.10.6. User Verification Requirement enumeration (enum
8932 UserVerificationRequirement)
8933
8934 #dom-userverificationrequirement-requiredReferenced in:
8935 * 5.1.3. Create a new credential - PublicKeyCredential's
8936 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8937 * 5.1.4.1. PublicKeyCredential's
8938 [[DiscoverFromExternalSource]](origin, options,
8939 sameOriginWithAncestors) method (2)
8940 * 5.10.6. User Verification Requirement enumeration (enum
8941 UserVerificationRequirement)
8942
8943 #dom-userverificationrequirement-preferredReferenced in:
8944 * 5.1.3. Create a new credential - PublicKeyCredential's
8945 [[Create]](origin, options, sameOriginWithAncestors) method
8946 * 5.1.4.1. PublicKeyCredential's
8947 [[DiscoverFromExternalSource]](origin, options,
8948 sameOriginWithAncestors) method
8949 * 5.10.6. User Verification Requirement enumeration (enum
8950 UserVerificationRequirement)
8951
8952 #dom-userverificationrequirement-discouragedReferenced in:
8953 * 5.1.3. Create a new credential - PublicKeyCredential's
8954 [[Create]](origin, options, sameOriginWithAncestors) method
8955 * 5.1.4.1. PublicKeyCredential's
8956 [[DiscoverFromExternalSource]](origin, options,
8957 sameOriginWithAncestors) method
8958 * 5.10.6. User Verification Requirement enumeration (enum
8959 UserVerificationRequirement)
8960
8961 #authenticator-modelReferenced in:
8962 * 6. WebAuthn Authenticator Model
8963
8964 #authenticator-credentials-mapReferenced in:
8965 * 6.2.1. Lookup Credential Source by Credential ID algorithm
8966 * 6.2.2. The authenticatorMakeCredential operation
8967 * 6.2.3. The authenticatorGetAssertion operation
8968
8969 #attestation-signatureReferenced in:

```

894f \* 4. Terminology  
894f \* 6. WebAuthn Authenticator Model (2) (3)  
895c \* 6.3. Attestation  
8951 \* 7.1. Registering a new credential  
8952 \* 8.6. FIDO U2F Attestation Statement Format  
8953  
8954 #assertion-signatureReferenced in:  
8955 \* 6. WebAuthn Authenticator Model (2)  
8956 \* 6.1.2. FIDO U2F signature format compatibility (2)  
8957 \* 6.2.3. The authenticatorGetAssertion operation (2) (3)  
8958  
8959 #authenticator-dataReferenced in:  
896c \* 5.1.4.1. PublicKeyCredential's  
8961 [[DiscoverFromExternalSource]](origin, options,  
8962 sameOriginWithAncestors) method  
8963 \* 5.2.1. Information about Public Key Credential (interface  
8964 AuthenticatorAttestationResponse) (2)  
8965 \* 5.2.2. Web Authentication Assertion (interface  
8966 AuthenticatorAssertionResponse)  
8967 \* 6. WebAuthn Authenticator Model (2)  
8968 \* 6.1. Authenticator data (2) (3) (4) (5) (6) (7) (8) (9)  
8969 \* 6.1.1. Signature Counter Considerations (2)  
897c \* 6.1.2. FIDO U2F signature format compatibility (2) (3)  
8971 \* 6.2.2. The authenticatorMakeCredential operation  
8972 \* 6.2.3. The authenticatorGetAssertion operation  
8973 \* 6.3. Attestation (2)  
8974 \* 6.3.1. Attested credential data  
8975 \* 6.3.2. Attestation Statement Formats (2)  
8976 \* 6.3.4. Generating an Attestation Object  
8977 \* 7.1. Registering a new credential  
8978 \* 8.5. Android SafetyNet Attestation Statement Format  
8979 \* 9.5. Authenticator extension processing (2)  
898c \* 10.6. User Verification Index Extension (uvi)  
8981 \* 10.8. User Verification Method Extension (uvm)  
8982 \* 13.2.1. Attestation Certificate Hierarchy  
8983 \* 13.4. credentialId Unsigned  
8984  
8985 #rpIdhashReferenced in:  
8986 \* 6.1.2. FIDO U2F signature format compatibility  
8987 \* 7.2. Verifying an authentication assertion  
8988  
8989 #flagsReferenced in:  
899c \* 5.10.6. User Verification Requirement enumeration (enum  
8991 UserVerificationRequirement) (2)  
8992 \* 6.1. Authenticator data  
8993 \* 6.1.2. FIDO U2F signature format compatibility  
8994 \* 7.1. Registering a new credential (2)  
8995 \* 7.2. Verifying an authentication assertion (2)  
8996  
8997 #signcountReferenced in:  
8998 \* 6.1.1. Signature Counter Considerations (2)  
8999 \* 7.2. Verifying an authentication assertion (2) (3)  
900c  
9001 #attestedcredentialdataReferenced in:  
9002 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9003 [[Create]](origin, options, sameOriginWithAncestors) method  
9004 \* 6.1. Authenticator data (2)  
9005 \* 6.1.2. FIDO U2F signature format compatibility  
9006 \* 6.2.2. The authenticatorMakeCredential operation  
9007 \* 6.2.3. The authenticatorGetAssertion operation  
9008 \* 7.1. Registering a new credential (2)  
9009 \* 8.3. TPM Attestation Statement Format  
901c \* 8.4. Android Key Attestation Statement Format  
9011 \* 8.6. FIDO U2F Attestation Statement Format  
9012  
9013 #authdataextensionsReferenced in:  
9014 \* 6.1. Authenticator data  
9015 \* 6.1.2. FIDO U2F signature format compatibility  
9016 \* 6.2.2. The authenticatorMakeCredential operation  
9017 \* 6.2.3. The authenticatorGetAssertion operation

8967 \* 4. Terminology  
8968 \* 6. WebAuthn Authenticator Model (2) (3)  
8969 \* 6.3. Attestation  
897c \* 7.1. Registering a new credential  
8971 \* 8.6. FIDO U2F Attestation Statement Format  
8972  
8973 #assertion-signatureReferenced in:  
8974 \* 6. WebAuthn Authenticator Model (2)  
8975 \* 6.1.2. FIDO U2F signature format compatibility (2)  
8976 \* 6.2.3. The authenticatorGetAssertion operation (2) (3)  
8977  
8978 #authenticator-dataReferenced in:  
8979 \* 5.1.4.1. PublicKeyCredential's  
897c [[DiscoverFromExternalSource]](origin, options,  
8981 sameOriginWithAncestors) method  
8982 \* 5.2.1. Information about Public Key Credential (interface  
8983 AuthenticatorAttestationResponse) (2)  
8984 \* 5.2.2. Web Authentication Assertion (interface  
8985 AuthenticatorAssertionResponse)  
8986 \* 6. WebAuthn Authenticator Model (2)  
8987 \* 6.1. Authenticator data (2) (3) (4) (5) (6) (7) (8) (9)  
8988 \* 6.1.1. Signature Counter Considerations (2)  
8989 \* 6.1.2. FIDO U2F signature format compatibility (2) (3)  
899c \* 6.2.2. The authenticatorMakeCredential operation  
8991 \* 6.2.3. The authenticatorGetAssertion operation  
8992 \* 6.3. Attestation (2)  
8993 \* 6.3.1. Attested credential data  
8994 \* 6.3.2. Attestation Statement Formats (2)  
8995 \* 6.3.4. Generating an Attestation Object  
8996 \* 7.1. Registering a new credential  
8997 \* 8.5. Android SafetyNet Attestation Statement Format  
8998 \* 9.5. Authenticator extension processing (2)  
8999 \* 10.6. User Verification Index Extension (uvi)  
900c \* 10.8. User Verification Method Extension (uvm)  
9001 \* 13.2.1. Attestation Certificate Hierarchy  
9002 \* 13.4. credentialId Unsigned  
9003  
9004 #rpIdhashReferenced in:  
9005 \* 6.1.2. FIDO U2F signature format compatibility  
9006 \* 7.2. Verifying an authentication assertion  
9007  
9008 #flagsReferenced in:  
9009 \* 5.10.6. User Verification Requirement enumeration (enum  
901c UserVerificationRequirement) (2)  
9011 \* 6.1. Authenticator data  
9012 \* 6.1.2. FIDO U2F signature format compatibility  
9013 \* 7.1. Registering a new credential (2)  
9014 \* 7.2. Verifying an authentication assertion (2)  
9015  
9016 #signcountReferenced in:  
9017 \* 6.1.1. Signature Counter Considerations (2)  
9018 \* 7.2. Verifying an authentication assertion (2) (3)  
9019  
902c #attestedcredentialdataReferenced in:  
9021 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9022 [[Create]](origin, options, sameOriginWithAncestors) method  
9023 \* 6.1. Authenticator data (2)  
9024 \* 6.1.2. FIDO U2F signature format compatibility  
9025 \* 6.2.2. The authenticatorMakeCredential operation  
9026 \* 6.2.3. The authenticatorGetAssertion operation  
9027 \* 7.1. Registering a new credential (2)  
9028 \* 8.3. TPM Attestation Statement Format  
9029 \* 8.4. Android Key Attestation Statement Format  
903c \* 8.6. FIDO U2F Attestation Statement Format  
9031  
9032 #authdataextensionsReferenced in:  
9033 \* 6.1. Authenticator data  
9034 \* 6.1.2. FIDO U2F signature format compatibility  
9035 \* 6.2.2. The authenticatorMakeCredential operation  
9036 \* 6.2.3. The authenticatorGetAssertion operation

9018 \* 7.1. Registering a new credential (2)  
9019 \* 7.2. Verifying an authentication assertion (2)  
9020 \* 9.5. Authenticator extension processing (2)  
9021  
9022 #signature-counterReferenced in:  
9023 \* 6.1. Authenticator data  
9024 \* 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6) (7) (8)  
9025 (9) (10)  
9026 \* 6.2.2. The authenticatorMakeCredential operation (2) (3) (4)  
9027 \* 6.2.3. The authenticatorGetAssertion operation (2)  
9028 \* 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6)  
9029  
9030 #authenticator-operationsReferenced in:  
9031 \* 4. Terminology  
9032  
9033 #authenticator-sessionReferenced in:  
9034 \* 5.6. Abort operations with AbortSignal (2)  
9035 \* 6.2.2. The authenticatorMakeCredential operation  
9036 \* 6.2.3. The authenticatorGetAssertion operation  
9037 \* 6.2.4. The authenticatorCancel operation (2)  
9038  
9039 #credential-id-looking-upReferenced in:  
9040 \* 6.2.2. The authenticatorMakeCredential operation  
9041 \* 6.2.3. The authenticatorGetAssertion operation  
9042  
9043 #authenticatormakecredentialReferenced in:  
9044 \* 4. Terminology (2) (3) (4)  
9045 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9046 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
9047 \* 6. WebAuthn Authenticator Model  
9048 \* 6.1.2. FIDO U2F signature format compatibility  
9049 \* 6.2.4. The authenticatorCancel operation (2)  
9050 \* 9. WebAuthn Extensions  
9051 \* 9.2. Defining extensions  
9052 \* 9.5. Authenticator extension processing  
9053  
9054 #authenticatorgetassertionReferenced in:  
9055 \* 4. Terminology (2) (3)  
9056 \* 5.1.4.1. PublicKeyCredential's  
9057 [[DiscoverFromExternalSource]](origin, options,  
9058 sameOriginWithAncestors) method (2) (3) (4)  
9059 \* 6. WebAuthn Authenticator Model  
9060 \* 6.1. Authenticator data  
9061 \* 6.1.1. Signature Counter Considerations (2) (3)  
9062 \* 6.2.4. The authenticatorCancel operation (2)  
9063 \* 9. WebAuthn Extensions  
9064 \* 9.2. Defining extensions  
9065 \* 9.5. Authenticator extension processing  
9066 \* 10.1. FIDO AppID Extension (appid)  
9067  
9068 #authenticatorcancelReferenced in:  
9069 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9070 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
9071 (4) (5)  
9072 \* 5.1.4.1. PublicKeyCredential's  
9073 [[DiscoverFromExternalSource]](origin, options,  
9074 sameOriginWithAncestors) method (2) (3) (4)  
9075 \* 6.2.2. The authenticatorMakeCredential operation  
9076 \* 6.2.3. The authenticatorGetAssertion operation  
9077  
9078 #attestation-objectReferenced in:  
9079 \* 4. Terminology (2) (3)  
9080 \* 5. Web Authentication API  
9081 \* 5.2.1. Information about Public Key Credential (interface  
9082 AuthenticatorAttestationResponse) (2)  
9083 \* 5.4. Options for Credential Creation (dictionary  
9084 PublicKeyCredentialCreationOptions) (2)  
9085 \* 6.2.2. The authenticatorMakeCredential operation (2)  
9086 \* 6.3. Attestation (2) (3)  
9087 \* 6.3.1. Attested credential data

9037 \* 7.1. Registering a new credential (2)  
9038 \* 7.2. Verifying an authentication assertion (2)  
9039 \* 9.5. Authenticator extension processing (2)  
9040  
9041 #signature-counterReferenced in:  
9042 \* 6.1. Authenticator data  
9043 \* 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6) (7) (8)  
9044 (9) (10)  
9045 \* 6.2.2. The authenticatorMakeCredential operation (2) (3) (4)  
9046 \* 6.2.3. The authenticatorGetAssertion operation (2)  
9047 \* 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6)  
9048  
9049 #authenticator-operationsReferenced in:  
9050 \* 4. Terminology  
9051  
9052 #authenticator-sessionReferenced in:  
9053 \* 5.6. Abort operations with AbortSignal (2)  
9054 \* 6.2.2. The authenticatorMakeCredential operation  
9055 \* 6.2.3. The authenticatorGetAssertion operation  
9056 \* 6.2.4. The authenticatorCancel operation (2)  
9057  
9058 #credential-id-looking-upReferenced in:  
9059 \* 6.2.2. The authenticatorMakeCredential operation  
9060 \* 6.2.3. The authenticatorGetAssertion operation  
9061  
9062 #authenticatormakecredentialReferenced in:  
9063 \* 4. Terminology (2) (3) (4)  
9064 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9065 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
9066 \* 6. WebAuthn Authenticator Model  
9067 \* 6.1.2. FIDO U2F signature format compatibility  
9068 \* 6.2.4. The authenticatorCancel operation (2)  
9069 \* 9. WebAuthn Extensions  
9070 \* 9.2. Defining extensions  
9071 \* 9.5. Authenticator extension processing  
9072  
9073 #authenticatorgetassertionReferenced in:  
9074 \* 4. Terminology (2) (3)  
9075 \* 5.1.4.1. PublicKeyCredential's  
9076 [[DiscoverFromExternalSource]](origin, options,  
9077 sameOriginWithAncestors) method (2) (3) (4)  
9078 \* 6. WebAuthn Authenticator Model  
9079 \* 6.1. Authenticator data  
9080 \* 6.1.1. Signature Counter Considerations (2) (3)  
9081 \* 6.2.4. The authenticatorCancel operation (2)  
9082 \* 9. WebAuthn Extensions  
9083 \* 9.2. Defining extensions  
9084 \* 9.5. Authenticator extension processing  
9085 \* 10.1. FIDO AppID Extension (appid)  
9086  
9087 #authenticatorcancelReferenced in:  
9088 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9089 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
9090 (4) (5)  
9091 \* 5.1.4.1. PublicKeyCredential's  
9092 [[DiscoverFromExternalSource]](origin, options,  
9093 sameOriginWithAncestors) method (2) (3) (4) (5)  
9094 \* 6.2.2. The authenticatorMakeCredential operation  
9095 \* 6.2.3. The authenticatorGetAssertion operation  
9096  
9097 #attestation-objectReferenced in:  
9098 \* 4. Terminology (2) (3)  
9099 \* 5. Web Authentication API  
9100 \* 5.2.1. Information about Public Key Credential (interface  
9101 AuthenticatorAttestationResponse) (2)  
9102 \* 5.4. Options for Credential Creation (dictionary  
9103 PublicKeyCredentialCreationOptions) (2)  
9104 \* 6.2.2. The authenticatorMakeCredential operation (2)  
9105 \* 6.3. Attestation (2) (3)  
9106 \* 6.3.1. Attested credential data

908E \* 6.3.4. Generating an Attestation Object (2)  
908E \* 7.1. Registering a new credential  
909C  
9091 #attestation-statementReferenced in:  
9092 \* 4. Terminology (2)  
9093 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9094 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
9095 \* 5.2.1. Information about Public Key Credential (interface  
9096 AuthenticatorAttestationResponse) (2) (3)  
9097 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
9098 AttestationConveyancePreference) (2) (3) (4) (5) (6)  
9099 \* 6.3. Attestation (2) (3) (4) (5) (6) (7) (8)  
9100 \* 6.3.2. Attestation Statement Formats (2) (3) (4)  
9101 \* 7.1. Registering a new credential  
9102 \* 8.7. None Attestation Statement Format  
9103 \* 13.3. Security Benefits for Relying Parties  
9104 \* 13.3.1. Considerations for Self and None Attestation Types and  
9105 Ignoring Attestation (2) (3)  
9106  
9107 #attestation-statement-formatReferenced in:  
9108 \* 5.2.1. Information about Public Key Credential (interface  
9109 AuthenticatorAttestationResponse)  
9110 \* 5.10.4. Authenticator Transport enumeration (enum  
9111 AuthenticatorTransport)  
9112 \* 6.2.2. The authenticatorMakeCredential operation  
9113 \* 6.3. Attestation (2) (3) (4) (5) (6) (7)  
9114 \* 6.3.2. Attestation Statement Formats (2) (3) (4)  
9115 \* 6.3.4. Generating an Attestation Object  
9116 \* 7.1. Registering a new credential (2)  
9117  
9118 #attestation-typeReferenced in:  
9119 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9120 [[Create]](origin, options, sameOriginWithAncestors) method  
9121 \* 6.3. Attestation (2) (3) (4) (5) (6)  
9122 \* 6.3.2. Attestation Statement Formats (2)  
9123  
9124 #attested-credential-dataReferenced in:  
9125 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9126 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
9127 \* 6.1. Authenticator data (2) (3) (4) (5)  
9128 \* 6.2.2. The authenticatorMakeCredential operation  
9129 \* 6.3. Attestation (2)  
9130 \* 6.3.1. Attested credential data  
9131 \* 6.3.3. Attestation Types  
9132  
9133 #aaguidReferenced in:  
9134 \* 4. Terminology  
9135 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9136 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
9137 (4)  
9138 \* 7.1. Registering a new credential  
9139 \* 8.2. Packed Attestation Statement Format  
9140 \* 8.3. TPM Attestation Statement Format  
9141  
9142 #credentialidlengthReferenced in:  
9143 \* 6.1. Authenticator data  
9144  
9145 #credentialidReferenced in:  
9146 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9147 [[Create]](origin, options, sameOriginWithAncestors) method  
9148 \* 6.1. Authenticator data  
9149 \* 7.1. Registering a new credential (2)  
9150  
9151 #credentialpublickeyReferenced in:  
9152 \* 6.1. Authenticator data  
9153 \* 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE\_Key  
9154 format  
9155 \* 7.1. Registering a new credential  
9156 \* 8.2. Packed Attestation Statement Format  
9157 \* 8.3. TPM Attestation Statement Format

9107 \* 6.3.4. Generating an Attestation Object (2)  
9108 \* 7.1. Registering a new credential  
9109  
9110 #attestation-statementReferenced in:  
9111 \* 4. Terminology (2)  
9112 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9113 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
9114 \* 5.2.1. Information about Public Key Credential (interface  
9115 AuthenticatorAttestationResponse) (2) (3)  
9116 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
9117 AttestationConveyancePreference) (2) (3) (4) (5) (6)  
9118 \* 6.3. Attestation (2) (3) (4) (5) (6) (7) (8)  
9119 \* 6.3.2. Attestation Statement Formats (2) (3) (4)  
9120 \* 7.1. Registering a new credential  
9121 \* 8.7. None Attestation Statement Format  
9122 \* 13.3. Security Benefits for Relying Parties  
9123 \* 13.3.1. Considerations for Self and None Attestation Types and  
9124 Ignoring Attestation (2) (3)  
9125  
9126 #attestation-statement-formatReferenced in:  
9127 \* 5.2.1. Information about Public Key Credential (interface  
9128 AuthenticatorAttestationResponse)  
9129 \* 5.10.4. Authenticator Transport enumeration (enum  
9130 AuthenticatorTransport)  
9131 \* 6.2.2. The authenticatorMakeCredential operation  
9132 \* 6.3. Attestation (2) (3) (4) (5) (6) (7)  
9133 \* 6.3.2. Attestation Statement Formats (2) (3) (4)  
9134 \* 6.3.4. Generating an Attestation Object  
9135 \* 7.1. Registering a new credential (2)  
9136  
9137 #attestation-typeReferenced in:  
9138 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9139 [[Create]](origin, options, sameOriginWithAncestors) method  
9140 \* 6.3. Attestation (2) (3) (4) (5) (6)  
9141 \* 6.3.2. Attestation Statement Formats (2)  
9142  
9143 #attested-credential-dataReferenced in:  
9144 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9145 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
9146 \* 6.1. Authenticator data (2) (3) (4) (5)  
9147 \* 6.2.2. The authenticatorMakeCredential operation  
9148 \* 6.3. Attestation (2)  
9149 \* 6.3.1. Attested credential data  
9150 \* 6.3.3. Attestation Types  
9151  
9152 #aaguidReferenced in:  
9153 \* 4. Terminology  
9154 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9155 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
9156 (4)  
9157 \* 7.1. Registering a new credential  
9158 \* 8.2. Packed Attestation Statement Format  
9159 \* 8.3. TPM Attestation Statement Format  
9160  
9161 #credentialidlengthReferenced in:  
9162 \* 6.1. Authenticator data  
9163  
9164 #credentialidReferenced in:  
9165 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9166 [[Create]](origin, options, sameOriginWithAncestors) method  
9167 \* 6.1. Authenticator data  
9168 \* 7.1. Registering a new credential (2)  
9169  
9170 #credentialpublickeyReferenced in:  
9171 \* 6.1. Authenticator data  
9172 \* 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE\_Key  
9173 format  
9174 \* 7.1. Registering a new credential  
9175 \* 8.2. Packed Attestation Statement Format  
9176 \* 8.3. TPM Attestation Statement Format

915E \* 8.4. Android Key Attestation Statement Format  
915F  
9160 #signing-procedureReferenced in:  
9161 \* 6.3.2. Attestation Statement Formats  
9162 \* 6.3.4. Generating an Attestation Object  
9163  
9164 #authenticator-data-for-the-attestationReferenced in:  
9165 \* 8.2. Packed Attestation Statement Format  
9166 \* 8.3. TPM Attestation Statement Format  
9167 \* 8.4. Android Key Attestation Statement Format (2)  
9168 \* 8.5. Android SafetyNet Attestation Statement Format  
9169 \* 8.6. FIDO U2F Attestation Statement Format  
9170  
9171 #verification-procedure-inputsReferenced in:  
9172 \* 8.2. Packed Attestation Statement Format  
9173 \* 8.3. TPM Attestation Statement Format  
9174 \* 8.4. Android Key Attestation Statement Format  
9175 \* 8.5. Android SafetyNet Attestation Statement Format  
9176 \* 8.6. FIDO U2F Attestation Statement Format  
9177  
9178 #authenticator-data-claimed-to-have-been-used-for-the-attestationReferenced in:  
9179 nced in:  
9180 \* 8.4. Android Key Attestation Statement Format  
9181  
9182 #attestation-trust-pathReferenced in:  
9183 \* 6.3.2. Attestation Statement Formats  
9184 \* 8.2. Packed Attestation Statement Format (2) (3)  
9185 \* 8.3. TPM Attestation Statement Format  
9186 \* 8.4. Android Key Attestation Statement Format  
9187 \* 8.5. Android SafetyNet Attestation Statement Format  
9188 \* 8.6. FIDO U2F Attestation Statement Format  
9189  
9190 #basic-attestationReferenced in:  
9191 \* 14.1. Attestation Privacy  
9192  
9193 #basicReferenced in:  
9194 \* 8.2. Packed Attestation Statement Format (2)  
9195 \* 8.4. Android Key Attestation Statement Format (2)  
9196 \* 8.5. Android SafetyNet Attestation Statement Format (2)  
9197 \* 8.6. FIDO U2F Attestation Statement Format (2)  
9198  
9199 #self-attestationReferenced in:  
9200 \* 4. Terminology (2) (3) (4)  
9201 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9202 [[Create]](origin, options, sameOriginWithAncestors) method  
9203 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
9204 AttestationConveyancePreference)  
9205 \* 6.3. Attestation (2)  
9206 \* 6.3.2. Attestation Statement Formats  
9207 \* 6.3.3. Attestation Types  
9208 \* 7.1. Registering a new credential (2) (3)  
9209 \* 8.2. Packed Attestation Statement Format (2)  
9210 \* 13.2.2. Attestation Certificate and Attestation Certificate CA  
9211 Compromise  
9212 \* 13.3.1. Considerations for Self and None Attestation Types and  
9213 Ignoring Attestation  
9214  
9215 #selfReferenced in:  
9216 \* 8.2. Packed Attestation Statement Format  
9217  
9218 #attestation-caReferenced in:  
9219 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
9220 AttestationConveyancePreference)  
9221 \* 6.3.3. Attestation Types (2)  
9222 \* 14.1. Attestation Privacy (2)  
9223  
9224 #attcaReferenced in:  
9225 \* 8.2. Packed Attestation Statement Format  
9226 \* 8.3. TPM Attestation Statement Format (2)  
9227 \* 8.6. FIDO U2F Attestation Statement Format

9177 \* 8.4. Android Key Attestation Statement Format  
9178  
9179 #signing-procedureReferenced in:  
9180 \* 6.3.2. Attestation Statement Formats  
9181 \* 6.3.4. Generating an Attestation Object  
9182  
9183 #authenticator-data-for-the-attestationReferenced in:  
9184 \* 8.2. Packed Attestation Statement Format  
9185 \* 8.3. TPM Attestation Statement Format  
9186 \* 8.4. Android Key Attestation Statement Format (2)  
9187 \* 8.5. Android SafetyNet Attestation Statement Format  
9188 \* 8.6. FIDO U2F Attestation Statement Format  
9189  
9190 #verification-procedure-inputsReferenced in:  
9191 \* 8.2. Packed Attestation Statement Format  
9192 \* 8.3. TPM Attestation Statement Format  
9193 \* 8.4. Android Key Attestation Statement Format  
9194 \* 8.5. Android SafetyNet Attestation Statement Format  
9195 \* 8.6. FIDO U2F Attestation Statement Format  
9196  
9197 #authenticator-data-claimed-to-have-been-used-for-the-attestationReferenced in:  
9198 nced in:  
9199 \* 8.4. Android Key Attestation Statement Format  
9200  
9201 #attestation-trust-pathReferenced in:  
9202 \* 6.3.2. Attestation Statement Formats  
9203 \* 8.2. Packed Attestation Statement Format (2) (3)  
9204 \* 8.3. TPM Attestation Statement Format  
9205 \* 8.4. Android Key Attestation Statement Format  
9206 \* 8.5. Android SafetyNet Attestation Statement Format  
9207 \* 8.6. FIDO U2F Attestation Statement Format  
9208  
9209 #basic-attestationReferenced in:  
9210 \* 14.1. Attestation Privacy  
9211  
9212 #basicReferenced in:  
9213 \* 8.2. Packed Attestation Statement Format (2)  
9214 \* 8.4. Android Key Attestation Statement Format (2)  
9215 \* 8.5. Android SafetyNet Attestation Statement Format (2)  
9216 \* 8.6. FIDO U2F Attestation Statement Format (2)  
9217  
9218 #self-attestationReferenced in:  
9219 \* 4. Terminology (2) (3) (4)  
9220 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9221 [[Create]](origin, options, sameOriginWithAncestors) method  
9222 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
9223 AttestationConveyancePreference)  
9224 \* 6.3. Attestation (2)  
9225 \* 6.3.2. Attestation Statement Formats  
9226 \* 6.3.3. Attestation Types  
9227 \* 7.1. Registering a new credential (2) (3)  
9228 \* 8.2. Packed Attestation Statement Format (2)  
9229 \* 13.2.2. Attestation Certificate and Attestation Certificate CA  
9230 Compromise  
9231 \* 13.3.1. Considerations for Self and None Attestation Types and  
9232 Ignoring Attestation  
9233  
9234 #selfReferenced in:  
9235 \* 8.2. Packed Attestation Statement Format  
9236  
9237 #attestation-caReferenced in:  
9238 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
9239 AttestationConveyancePreference)  
9240 \* 6.3.3. Attestation Types (2)  
9241 \* 14.1. Attestation Privacy (2)  
9242  
9243 #attcaReferenced in:  
9244 \* 8.2. Packed Attestation Statement Format  
9245 \* 8.3. TPM Attestation Statement Format (2)  
9246 \* 8.6. FIDO U2F Attestation Statement Format

9228 #elliptic-curve-based-direct-anonymous-attestationReferenced in:  
9229 \* 14.1. Attestation Privacy  
9230  
9231 #ecdaaReferenced in:  
9232 \* 6.3.2. Attestation Statement Formats  
9233 \* 6.3.3. Attestation Types  
9234 \* 7.1. Registering a new credential  
9235 \* 8.2. Packed Attestation Statement Format (2) (3) (4)  
9236 \* 8.3. TPM Attestation Statement Format (2) (3) (4)  
9237 \* 13.2.2. Attestation Certificate and Attestation Certificate CA  
9238 Compromise  
9239  
9240 #no-attestation-statementReferenced in:  
9241 \* 13.3.1. Considerations for Self and None Attestation Types and  
9242 Ignoring Attestation  
9243  
9244 #noneReferenced in:  
9245 \* 8.7. None Attestation Statement Format (2)  
9246 \* 13.3.1. Considerations for Self and None Attestation Types and  
9247 Ignoring Attestation  
9248  
9249 #attestation-statement-format-identifierReferenced in:  
9250 \* 6.3.2. Attestation Statement Formats  
9251 \* 6.3.4. Generating an Attestation Object  
9252  
9253 #identifier-of-the-ecdaa-issuer-public-keyReferenced in:  
9254 \* 7.1. Registering a new credential  
9255 \* 8.2. Packed Attestation Statement Format  
9256 \* 8.3. TPM Attestation Statement Format (2)  
9257  
9258 #ecdaa-issuer-public-keyReferenced in:  
9259 \* 6.3.2. Attestation Statement Formats  
9260 \* 7.1. Registering a new credential  
9261 \* 8.2. Packed Attestation Statement Format (2) (3)  
9262 \* 14.1. Attestation Privacy  
9263  
9264 #registration-extensionReferenced in:  
9265 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9266 [[Create]](origin, options, sameOriginWithAncestors) method  
9267 \* 9. WebAuthn Extensions (2) (3) (4) (5) (6)  
9268 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
9269 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
9270 \* 10.4. Authenticator Selection Extension (authnSel)  
9271 \* 10.5. Supported Extensions Extension (exts)  
9272 \* 10.6. User Verification Index Extension (uvi)  
9273 \* 10.7. Location Extension (loc)  
9274 \* 10.8. User Verification Method Extension (uvm)  
9275 \* 10.9. Biometric Authenticator Performance Bounds Extension  
9276 (biometricPerfBounds)  
9277 \* 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)  
9278 (6) (7)  
9279  
9280 #authentication-extensionReferenced in:  
9281 \* 5.1.4.1. PublicKeyCredential's  
9282 [[DiscoverFromExternalSource]](origin, options,  
9283 sameOriginWithAncestors) method  
9284 \* 9. WebAuthn Extensions (2) (3) (4) (5) (6)  
9285 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
9286 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
9287 \* 10.6. User Verification Index Extension (uvi)  
9288 \* 10.7. Location Extension (loc)  
9289 \* 10.8. User Verification Method Extension (uvm)  
9290 \* 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)  
9291 (6)  
9292  
9293 #client-extensionReferenced in:  
9294 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9295 [[Create]](origin, options, sameOriginWithAncestors) method  
9296 \* 5.1.4.1. PublicKeyCredential's  
9297

9247 #elliptic-curve-based-direct-anonymous-attestationReferenced in:  
9248 \* 14.1. Attestation Privacy  
9249  
9250 #ecdaaReferenced in:  
9251 \* 6.3.2. Attestation Statement Formats  
9252 \* 6.3.3. Attestation Types  
9253 \* 7.1. Registering a new credential  
9254 \* 8.2. Packed Attestation Statement Format (2) (3) (4)  
9255 \* 8.3. TPM Attestation Statement Format (2) (3) (4)  
9256 \* 13.2.2. Attestation Certificate and Attestation Certificate CA  
9257 Compromise  
9258  
9259 #no-attestation-statementReferenced in:  
9260 \* 13.3.1. Considerations for Self and None Attestation Types and  
9261 Ignoring Attestation  
9262  
9263 #noneReferenced in:  
9264 \* 8.7. None Attestation Statement Format (2)  
9265 \* 13.3.1. Considerations for Self and None Attestation Types and  
9266 Ignoring Attestation  
9267  
9268 #attestation-statement-format-identifierReferenced in:  
9269 \* 6.3.2. Attestation Statement Formats  
9270 \* 6.3.4. Generating an Attestation Object  
9271  
9272 #identifier-of-the-ecdaa-issuer-public-keyReferenced in:  
9273 \* 7.1. Registering a new credential  
9274 \* 8.2. Packed Attestation Statement Format  
9275 \* 8.3. TPM Attestation Statement Format (2)  
9276  
9277 #ecdaa-issuer-public-keyReferenced in:  
9278 \* 6.3.2. Attestation Statement Formats  
9279 \* 7.1. Registering a new credential  
9280 \* 8.2. Packed Attestation Statement Format (2) (3)  
9281 \* 14.1. Attestation Privacy  
9282  
9283 #registration-extensionReferenced in:  
9284 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9285 [[Create]](origin, options, sameOriginWithAncestors) method  
9286 \* 9. WebAuthn Extensions (2) (3) (4) (5) (6)  
9287 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
9288 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
9289 \* 10.4. Authenticator Selection Extension (authnSel)  
9290 \* 10.5. Supported Extensions Extension (exts)  
9291 \* 10.6. User Verification Index Extension (uvi)  
9292 \* 10.7. Location Extension (loc)  
9293 \* 10.8. User Verification Method Extension (uvm)  
9294 \* 10.9. Biometric Authenticator Performance Bounds Extension  
9295 (biometricPerfBounds)  
9296 \* 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)  
9297 (6) (7)  
9298  
9299 #authentication-extensionReferenced in:  
9300 \* 5.1.4.1. PublicKeyCredential's  
9301 [[DiscoverFromExternalSource]](origin, options,  
9302 sameOriginWithAncestors) method  
9303 \* 9. WebAuthn Extensions (2) (3) (4) (5) (6)  
9304 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
9305 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
9306 \* 10.6. User Verification Index Extension (uvi)  
9307 \* 10.7. Location Extension (loc)  
9308 \* 10.8. User Verification Method Extension (uvm)  
9309 \* 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)  
9310 (6)  
9311  
9312 #client-extensionReferenced in:  
9313 \* 5.1.3. Create a new credential - PublicKeyCredential's  
9314 [[Create]](origin, options, sameOriginWithAncestors) method  
9315 \* 5.1.4.1. PublicKeyCredential's  
9316

```

929E [[DiscoverFromExternalSource]](origin, options,
929F sameOriginWithAncestors) method
9300 * 9. WebAuthn Extensions
9301 * 9.2. Defining extensions
9302 * 9.4. Client extension processing
9303 * 10.1. FIDO AppID Extension (appid)
9304
9305 #authenticator-extensionReferenced in:
9306 * 5.1.3. Create a new credential - PublicKeyCredential's
9307 [[Create]](origin, options, sameOriginWithAncestors) method
9308 * 5.1.4.1. PublicKeyCredential's
9309 [[DiscoverFromExternalSource]](origin, options,
9310 sameOriginWithAncestors) method
9311 * 9. WebAuthn Extensions (2) (3)
9312 * 9.2. Defining extensions (2)
9313 * 9.3. Extending request parameters
9314 * 9.5. Authenticator extension processing
9315
9316 #extension-identifierReferenced in:
9317 * 5.1. PublicKeyCredential Interface
9318 * 5.1.3. Create a new credential - PublicKeyCredential's
9319 [[Create]](origin, options, sameOriginWithAncestors) method
9320 * 5.1.4.1. PublicKeyCredential's
9321 [[DiscoverFromExternalSource]](origin, options,
9322 sameOriginWithAncestors) method
9323 * 6.1. Authenticator data
9324 * 6.2.2. The authenticatorMakeCredential operation (2)
9325 * 6.2.3. The authenticatorGetAssertion operation (2)
9326 * 7.1. Registering a new credential (2)
9327 * 7.2. Verifying an authentication assertion (2)
9328 * 9. WebAuthn Extensions (2)
9329 * 9.2. Defining extensions
9330 * 9.3. Extending request parameters
9331 * 9.4. Client extension processing (2)
9332 * 9.5. Authenticator extension processing (2)
9333 * 10.5. Supported Extensions Extension (exts) (2)
9334 * 11.2. WebAuthn Extension Identifier Registrations
9335
9336 #client-extension-inputReferenced in:
9337 * 5.7. Authentication Extensions Client Inputs (typedef
9338 AuthenticationExtensionsClientInputs)
9339 * 7.1. Registering a new credential
9340 * 7.2. Verifying an authentication assertion
9341 * 9. WebAuthn Extensions (2) (3) (4)
9342 * 9.2. Defining extensions
9343 * 9.3. Extending request parameters (2) (3) (4) (5) (6)
9344 * 9.4. Client extension processing (2) (3) (4)
9345
9346 #authenticator-extension-inputReferenced in:
9347 * 5.9. Authentication Extensions Authenticator Inputs (typedef
9348 AuthenticationExtensionsAuthenticatorInputs)
9349 * 6.2.2. The authenticatorMakeCredential operation (2)
9350 * 6.2.3. The authenticatorGetAssertion operation (2)
9351 * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
9352 * 9.2. Defining extensions
9353 * 9.3. Extending request parameters (2) (3)
9354 * 9.4. Client extension processing
9355 * 9.5. Authenticator extension processing (2) (3)
9356
9357 #client-extension-processingReferenced in:
9358 * 5.1. PublicKeyCredential Interface
9359 * 5.1.3. Create a new credential - PublicKeyCredential's
9360 [[Create]](origin, options, sameOriginWithAncestors) method (2)
9361 * 5.1.4.1. PublicKeyCredential's
9362 [[DiscoverFromExternalSource]](origin, options,
9363 sameOriginWithAncestors) method (2)
9364 * 9. WebAuthn Extensions (2) (3) (4)
9365 * 9.2. Defining extensions
9366
9367 #client-extension-outputReferenced in:

```

```

9317 [[DiscoverFromExternalSource]](origin, options,
9318 sameOriginWithAncestors) method
9319 * 9. WebAuthn Extensions
9320 * 9.2. Defining extensions
9321 * 9.4. Client extension processing
9322 * 10.1. FIDO AppID Extension (appid)
9323
9324 #authenticator-extensionReferenced in:
9325 * 5.1.3. Create a new credential - PublicKeyCredential's
9326 [[Create]](origin, options, sameOriginWithAncestors) method
9327 * 5.1.4.1. PublicKeyCredential's
9328 [[DiscoverFromExternalSource]](origin, options,
9329 sameOriginWithAncestors) method
9330 * 9. WebAuthn Extensions (2) (3)
9331 * 9.2. Defining extensions (2)
9332 * 9.3. Extending request parameters
9333 * 9.5. Authenticator extension processing
9334
9335 #extension-identifierReferenced in:
9336 * 5.1. PublicKeyCredential Interface
9337 * 5.1.3. Create a new credential - PublicKeyCredential's
9338 [[Create]](origin, options, sameOriginWithAncestors) method
9339 * 5.1.4.1. PublicKeyCredential's
9340 [[DiscoverFromExternalSource]](origin, options,
9341 sameOriginWithAncestors) method
9342 * 6.1. Authenticator data
9343 * 6.2.2. The authenticatorMakeCredential operation (2)
9344 * 6.2.3. The authenticatorGetAssertion operation (2)
9345 * 7.1. Registering a new credential (2)
9346 * 7.2. Verifying an authentication assertion (2)
9347 * 9. WebAuthn Extensions (2)
9348 * 9.2. Defining extensions
9349 * 9.3. Extending request parameters
9350 * 9.4. Client extension processing (2)
9351 * 9.5. Authenticator extension processing (2)
9352 * 10.5. Supported Extensions Extension (exts) (2)
9353 * 11.2. WebAuthn Extension Identifier Registrations
9354
9355 #client-extension-inputReferenced in:
9356 * 5.7. Authentication Extensions Client Inputs (typedef
9357 AuthenticationExtensionsClientInputs)
9358 * 7.1. Registering a new credential
9359 * 7.2. Verifying an authentication assertion
9360 * 9. WebAuthn Extensions (2) (3) (4)
9361 * 9.2. Defining extensions
9362 * 9.3. Extending request parameters (2) (3) (4) (5) (6)
9363 * 9.4. Client extension processing (2) (3) (4)
9364
9365 #authenticator-extension-inputReferenced in:
9366 * 5.9. Authentication Extensions Authenticator Inputs (typedef
9367 AuthenticationExtensionsAuthenticatorInputs)
9368 * 6.2.2. The authenticatorMakeCredential operation (2)
9369 * 6.2.3. The authenticatorGetAssertion operation (2)
9370 * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
9371 * 9.2. Defining extensions
9372 * 9.3. Extending request parameters (2) (3)
9373 * 9.4. Client extension processing
9374 * 9.5. Authenticator extension processing (2) (3)
9375
9376 #client-extension-processingReferenced in:
9377 * 5.1. PublicKeyCredential Interface
9378 * 5.1.3. Create a new credential - PublicKeyCredential's
9379 [[Create]](origin, options, sameOriginWithAncestors) method (2)
9380 * 5.1.4.1. PublicKeyCredential's
9381 [[DiscoverFromExternalSource]](origin, options,
9382 sameOriginWithAncestors) method (2)
9383 * 9. WebAuthn Extensions (2) (3) (4)
9384 * 9.2. Defining extensions
9385
9386 #client-extension-outputReferenced in:

```

```

936E * 5.1. PublicKeyCredential Interface
936F * 5.1.3. Create a new credential - PublicKeyCredential's
9370 [[Create]](origin, options, sameOriginWithAncestors) method (2)
9371 * 5.1.4.1. PublicKeyCredential's
9372 [[DiscoverFromExternalSource]](origin, options,
9373 sameOriginWithAncestors) method (2)
9374 * 5.8. Authentication Extensions Client Outputs (typedef
9375 AuthenticationExtensionsClientOutputs)
9376 * 7.1. Registering a new credential
9377 * 7.2. Verifying an authentication assertion
937E * 9. WebAuthn Extensions (2) (3) (4)
937F * 9.2. Defining extensions (2) (3)
9380 * 9.4. Client extension processing (2) (3)
9381
9382 #authenticator-extension-processingReferenced in:
9383 * 6.2.2. The authenticatorMakeCredential operation
9384 * 6.2.3. The authenticatorGetAssertion operation
9385 * 9. WebAuthn Extensions
9386 * 9.2. Defining extensions
9387 * 9.5. Authenticator extension processing
938E
938F #authenticator-extension-outputReferenced in:
9390 * 6.1. Authenticator data
9391 * 7.1. Registering a new credential
9392 * 7.2. Verifying an authentication assertion
9393 * 9. WebAuthn Extensions (2) (3) (4)
9394 * 9.2. Defining extensions (2) (3)
9395 * 9.4. Client extension processing
9396 * 9.5. Authenticator extension processing
9397 * 10.5. Supported Extensions Extension (exts)
939E * 10.6. User Verification Index Extension (uvi)
939F * 10.8. User Verification Method Extension (uvm)
9400
9401 #appidReferenced in:
9402 * 3. Dependencies
9403
9404 #dictdef-txauthgenericargReferenced in:
9405 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
9406
9407 #typedefdef-authenticatorselectionlistReferenced in:
9408 * 10.4. Authenticator Selection Extension (authnSel) (2)
9409
9410 #typedefdef-aaguidReferenced in:
9411 * 10.4. Authenticator Selection Extension (authnSel)
9412
9413 #typedefdef-authenticationextensionssupportedReferenced in:
9414 * 10.5. Supported Extensions Extension (exts)
9415
9416 #typedefdef-uvmentryReferenced in:
9417 * 10.8. User Verification Method Extension (uvm)
941E
941F #typedefdef-uvmentriesReferenced in:
9420 * 10.8. User Verification Method Extension (uvm)
9421
9422 #anonymization-caReferenced in:
9423 * 5.1.3. Create a new credential - PublicKeyCredential's
9424 [[Create]](origin, options, sameOriginWithAncestors) method
9425 * 5.4.6. Attestation Conveyance Preference enumeration (enum
9426 AttestationConveyancePreference)
9427 * 14.1. Attestation Privacy (2) (3)
942E

```

```

9387 * 5.1. PublicKeyCredential Interface
938E * 5.1.3. Create a new credential - PublicKeyCredential's
938F [[Create]](origin, options, sameOriginWithAncestors) method (2)
9390 * 5.1.4.1. PublicKeyCredential's
9391 [[DiscoverFromExternalSource]](origin, options,
9392 sameOriginWithAncestors) method (2)
9393 * 5.8. Authentication Extensions Client Outputs (typedef
9394 AuthenticationExtensionsClientOutputs)
9395 * 7.1. Registering a new credential
9396 * 7.2. Verifying an authentication assertion
9397 * 9. WebAuthn Extensions (2) (3) (4)
939E * 9.2. Defining extensions (2) (3)
939F * 9.4. Client extension processing (2) (3)
9400
9401 #authenticator-extension-processingReferenced in:
9402 * 6.2.2. The authenticatorMakeCredential operation
9403 * 6.2.3. The authenticatorGetAssertion operation
9404 * 9. WebAuthn Extensions
9405 * 9.2. Defining extensions
9406 * 9.5. Authenticator extension processing
9407
940E #authenticator-extension-outputReferenced in:
940F * 6.1. Authenticator data
9410 * 7.1. Registering a new credential
9411 * 7.2. Verifying an authentication assertion
9412 * 9. WebAuthn Extensions (2) (3) (4)
9413 * 9.2. Defining extensions (2) (3)
9414 * 9.4. Client extension processing
9415 * 9.5. Authenticator extension processing
941E * 10.5. Supported Extensions Extension (exts)
941F * 10.6. User Verification Index Extension (uvi)
9420 * 10.8. User Verification Method Extension (uvm)
9421
9422 #appidReferenced in:
9423 * 3. Dependencies
9424
9425 #dictdef-txauthgenericargReferenced in:
9426 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
9427
9428 #typedefdef-authenticatorselectionlistReferenced in:
9429 * 10.4. Authenticator Selection Extension (authnSel) (2)
942E
942F #typedefdef-aaguidReferenced in:
9430 * 10.4. Authenticator Selection Extension (authnSel)
9431
9432 #typedefdef-authenticationextensionssupportedReferenced in:
9433 * 10.5. Supported Extensions Extension (exts)
9434
9435 #typedefdef-uvmentryReferenced in:
9436 * 10.8. User Verification Method Extension (uvm)
9437
9438 #typedefdef-uvmentriesReferenced in:
9439 * 10.8. User Verification Method Extension (uvm)
9440
9441 #anonymization-caReferenced in:
9442 * 5.1.3. Create a new credential - PublicKeyCredential's
9443 [[Create]](origin, options, sameOriginWithAncestors) method
9444 * 5.4.6. Attestation Conveyance Preference enumeration (enum
9445 AttestationConveyancePreference)
9446 * 14.1. Attestation Privacy (2) (3)
9447

```