

0001 THE\_URL:file://localhost/Users/jehodges/documents/work/standards/W3C/webauthn/index-master-tr-5e63e57-WD-07.html

0002 THE\_TITLE:Web Authentication: An API for accessing Public Key Credentials - Level 1

0003 ^I Jump to Table of Contents-> Pop Out Sidebar

0004 W3C

0005 Web Authentication: An API for accessing Public Key Credentials - Level 1

0006 W3C Working Draft, 5 December 2017

0007 This version:

0008 <https://www.w3.org/TR/2017/WD-webauthn-20171205/>

0009 Latest published Version:

0010 <https://www.w3.org/TR/webauthn/>

0011 Editor's Draft:

0012 <https://w3c.github.io/webauthn/>

0013 Previous versions:

0014 <https://www.w3.org/TR/2017/WD-webauthn-20170811/>

0015 <https://www.w3.org/TR/2017/WD-webauthn-20170505/>

0016 <https://www.w3.org/TR/2017/WD-webauthn-20170216/>

0017 <https://www.w3.org/TR/2016/WD-webauthn-20161207/>

0018 <https://www.w3.org/TR/2016/WD-webauthn-20160928/>

0019 <https://www.w3.org/TR/2016/WD-webauthn-20160902/>

0020 <https://www.w3.org/TR/2016/WD-webauthn-20160531/>

0021 Issue Tracking:

0022 [Github](#)

0023 Editors:

0024 [Vijay Bharadwaj \(Microsoft\)](#)

0025 [Hubert Le Van Gong \(PayPal\)](#)

0026 [Dirk Balfanz \(Google\)](#)

0027 [Alexei Czeskis \(Google\)](#)

0028 [Arnar Birgisson \(Google\)](#)

0029 [Jeff Hodges \(PayPal\)](#)

0030 Michael B. Jones (Microsoft)

0031 Rolf Lindemann (Nok Nok Labs)

0032 [J.C. Jones \(Mozilla\)](#)

0042 Tests:

0043 [web-platform-tests webauthn/](#) (ongoing work)

0044 Copyright 2017 W3C^ (MIT, ERCIM, Keio, Beihang). W3C liability,  
0045 trademark and document use rules apply.

0051 Abstract

0052

0001 THE\_URL:file://localhost/Users/jehodges/documents/work/standards/W3C/webauthn/index-master-tr-e155bae-CR-00.html

0002 THE\_TITLE:Web Authentication: An API for accessing Public Key Credentials Level 1

0003 ^I Jump to Table of Contents-> Pop Out Sidebar

0004 W3C

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

0006 W3C Candidate Recommendation, 20 March 2018

0007 This version:

0008 <https://www.w3.org/TR/2018/CR-webauthn-20180320/>

0009 Latest published Version:

0010 <https://www.w3.org/TR/webauthn/>

0011 Editor's Draft:

0012 <https://w3c.github.io/webauthn/>

0013 Previous Versions:

0014 <https://www.w3.org/TR/2018/WD-webauthn-20180315/>

0015 <https://www.w3.org/TR/2018/WD-webauthn-20180306/>

0016 <https://www.w3.org/TR/2017/WD-webauthn-20171205/>

0017 <https://www.w3.org/TR/2017/WD-webauthn-20170811/>

0018 <https://www.w3.org/TR/2017/WD-webauthn-20170505/>

0019 <https://www.w3.org/TR/2017/WD-webauthn-20170216/>

0020 <https://www.w3.org/TR/2016/WD-webauthn-20161207/>

0021 <https://www.w3.org/TR/2016/WD-webauthn-20160928/>

0022 <https://www.w3.org/TR/2016/WD-webauthn-20160902/>

0023 <https://www.w3.org/TR/2016/WD-webauthn-20160531/>

0024 Issue Tracking:

0025 [GitHub](#)

0026 Editors:

0027 [Dirk Balfanz \(Google\)](#)

0028 [Alexei Czeskis \(Google\)](#)

0029 [Jeff Hodges \(PayPal\)](#)

0030 [J.C. Jones \(Mozilla\)](#)

0031 [Michael B. Jones \(Microsoft\)](#)

0032 [Akshay Kumar \(Microsoft\)](#)

0033 [Angelo Liao \(Microsoft\)](#)

0034 [Rolf Lindemann \(Nok Nok Labs\)](#)

0035 [Emil Lundberg \(Yubico\)](#)

0036 Former Editors:

0037 [Vijay Bharadwaj \(Microsoft\)](#)

0038 [Arnar Birgisson \(Google\)](#)

0039 [Hubert Le Van Gong \(PayPal\)](#)

0040 Contributors:

0041 [Christiaan Brand \(Google\)](#)

0042 [Adam Langley \(Google\)](#)

0043 [Giridhar Mandyam \(Qualcomm\)](#)

0044 [Mike West \(Google\)](#)

0045 [Jeffrey Yasskin \(Google\)](#)

0046 Tests:

0047 [web-platform-tests webauthn/](#) (ongoing work)

0048 Copyright 2018 W3C^ (MIT, ERCIM, Keio, Beihang). W3C liability,  
0049 trademark and document use rules apply.

0051 Abstract

0052

0052 This specification defines an API enabling the creation and use of  
0053 strong, attested, scoped, public key-based credentials by web  
0054 applications, for the purpose of strongly authenticating users.  
0055 Conceptually, one or more public key credentials, each scoped to a  
0056 given Relying Party, are created and stored on an authenticator by the  
0057 user agent in conjunction with the web application. The user agent  
0058 mediates access to public key credentials in order to preserve user  
0059 privacy. Authenticators are responsible for ensuring that no operation  
0060 is performed without user consent. Authenticators provide cryptographic  
0061 proof of their properties to relying parties via attestation. This  
0062 specification also describes the functional model for WebAuthn  
0063 conformant authenticators, including their signature and attestation  
0064 functionality.

0065 Status of this document

0066 This section describes the status of this document at the time of its  
0067 publication. Other documents may supersede this document. A list of  
0068 current W3C publications and the latest revision of this technical  
0069 report can be found in the W3C technical reports index at  
0070 <https://www.w3.org/TR/>.

0074 This document was published by the Web Authentication Working Group as  
0075 a **Working Draft**. This document is intended to become a W3C  
0076 Recommendation. Feedback and comments on this specification are  
0077 welcome. Please use Github issues. Discussions may also be found in the  
0078 public-webauthn@w3.org archives.

0079 **Publication as a Working Draft does not imply endorsement by the W3C  
0080 Membership. This is a draft document and may be updated, replaced or  
0081 obsoleted by other documents at any time. It is inappropriate to cite  
0082 this document as other than work in progress.**

0084 This document was produced by a group operating under the W3C Patent  
0085 Policy. W3C maintains a public list of any patent disclosures made in  
0086 connection with the deliverables of the group; that page also includes  
0087 instructions for disclosing a patent. An individual who has actual  
0088 knowledge of a patent which the individual believes contains Essential  
0089 Claim(s) must disclose the information in accordance with section 6 of  
0090 the W3C Patent Policy.

0091 This document is governed by the 1 **March** 2017 W3C Process Document.

0092 Table of Contents

- 0093 1. 1 Introduction
- 0094 1. 1.1 Use Cases
- 0095 1. 1.1.1 Registration
- 0100 2. 1.1.2 Authentication
- 0101 3. 1.1.3 Other use cases and configurations
- 0102 2. 2 Conformance
- 0103 1. 2.1 User Agents

0067 This specification defines an API enabling the creation and use of  
0068 strong, attested, scoped, public key-based credentials by web  
0069 applications, for the purpose of strongly authenticating users.  
0070 Conceptually, one or more public key credentials, each scoped to a  
0071 given Relying Party, are created and stored on an authenticator by the  
0072 user agent in conjunction with the web application. The user agent  
0073 mediates access to public key credentials in order to preserve user  
0074 privacy. Authenticators are responsible for ensuring that no operation  
0075 is performed without user consent. Authenticators provide cryptographic  
0076 proof of their properties to relying parties via attestation. This  
0077 specification also describes the functional model for WebAuthn  
0078 conformant authenticators, including their signature and attestation  
0079 functionality.

0080 Status of this document

0081 This section describes the status of this document at the time of its  
0082 publication. Other documents may supersede this document. A list of  
0083 current W3C publications and the latest revision of this technical  
0084 report can be found in the W3C technical reports index at  
0085 <https://www.w3.org/TR/>.

0086 **For the Web Authentication specification to move to Proposed  
0087 Recommendation we must show two independent, interoperable  
0088 implementations of the Web Authentication API in browsers. We will also  
0089 have multiple interoperable implementations of the AppID extension,  
0090 validating the extensions framework. All other extensions are "at  
0091 risk". If there are not multiple interoperable implementations, each  
0092 may independently be removed or made informative at Proposed  
0093 Recommendation.**

0094 **We have had two informal interoperability tests with implementations in  
0095 three browsers. There is no preliminary implementation report at this  
0096 time.**

0097 This document was published by the Web Authentication Working Group as  
0098 a **Candidate Recommendation**. This document is intended to become a W3C  
0099 Recommendation. Feedback and comments on this specification are  
0100 welcome. Please use Github issues. Discussions may also be found in the  
0101 public-webauthn@w3.org archives. **W3C publishes a Candidate  
0102 Recommendation to indicate that the document is believed to be stable  
0103 and to encourage implementation by the developer community.**

0104 **The deadline for comments for this Candidate Recommendation is 1 May  
0105 2018.**

0106 **Publication as a Candidate Recommendation does not imply endorsement by  
0107 the W3C Membership. This is a draft document and may be updated,  
0108 replaced or obsoleted by other documents at any time. It is  
0109 inappropriate to cite this document as other than work in progress.**

0110 This document was produced by a group operating under the W3C Patent  
0111 Policy. W3C maintains a public list of any patent disclosures made in  
0112 connection with the deliverables of the group; that page also includes  
0113 instructions for disclosing a patent. An individual who has actual  
0114 knowledge of a patent which the individual believes contains Essential  
0115 Claim(s) must disclose the information in accordance with section 6 of  
0116 the W3C Patent Policy.

0117 This document is governed by the 1 **February** 2018 W3C Process Document.

0118 Table of Contents

- 0119 1. 1 Introduction
- 0120 1. 1.1 Use Cases
- 0121 1. 1.1.1 Registration
- 0122 2. 1.1.2 Authentication
- 0123 3. 1.1.3 Other use cases and configurations
- 0124 2. 2 Conformance
- 0125 1. 2.1 User Agents

- 0104 2. 2.2 Authenticators
- 0105 3. 2.3 Relying Parties
- 0106 3. 3 Dependencies
- 0107 4. 4 Terminology
- 0108 5. 5 Web Authentication API
- 0109 1. 5.1 PublicKeyCredential Interface
- 0110 1. 5.1.1 CredentialCreationOptions Extension
- 0111 2. 5.1.2 CredentialRequestOptions Extension
- 0112 3. 5.1.3 Create a new credential - PublicKeyCredential's
- 0113 [[Create]](origin, options, sameOriginWithAncestors)
- 0114 method
- 0115 4. 5.1.4 Use an existing credential to make an assertion -
- 0116 PublicKeyCredential's [[Get]](options) method
- 0117 1. 5.1.4.1 PublicKeyCredential's
- 0118 [[DiscoverFromExternalSource]](origin, options,
- 0119 sameOriginWithAncestors) method
- 0120 5. 5.1.5 Store an existing credential -
- 0121 PublicKeyCredential's [[Store]](credential,
- 0122 sameOriginWithAncestors) method
- 0123 6. 5.1.6 Availability of User-Verifying Platform
- 0124 Authenticator - PublicKeyCredential's
- 0125 isUserVerifyingPlatformAuthenticatorAvailable() method
- 0126 2. 5.2 Authenticator Responses (interface AuthenticatorResponse)
- 0127 1. 5.2.1 Information about Public Key Credential (interface
- 0128 AuthenticatorAttestationResponse)
- 0129 2. 5.2.2 Web Authentication Assertion (interface
- 0130 AuthenticatorAssertionResponse)
- 0131 3. 5.3 Parameters for Credential Generation (dictionary
- 0132 PublicKeyCredentialParameters)
- 0133 4. 5.4 Options for Credential Creation (dictionary
- 0134 MakePublicKeyCredentialOptions)
- 0135 1. 5.4.1 Public Key Entity Description (dictionary
- 0136 PublicKeyCredentialEntity)
- 0137 2. 5.4.2 RP Parameters for Credential Generation (dictionary
- 0138 PublicKeyCredentialRpEntity)
- 0139 3. 5.4.3 User Account Parameters for Credential Generation
- 0140 (dictionary PublicKeyCredentialUserEntity)
- 0141 4. 5.4.4 Authenticator Selection Criteria (dictionary
- 0142 AuthenticatorSelectionCriteria)
- 0143 5. 5.4.5 Authenticator Attachment enumeration (enum
- 0144 AuthenticatorAttachment)
- 0145 6. 5.4.6 Attestation Conveyance Preference enumeration (enum
- 0146 AttestationConveyancePreference)
- 0147 5. 5.5 Options for Assertion Generation (dictionary
- 0148 PublicKeyCredentialRequestOptions)
- 0149 6. 5.6 Abort operations with AbortSignal
- 0150 7. 5.7 Authentication Extensions (typedef
- 0151 AuthenticationExtensions)
- 0152 8. 5.8 Supporting Data Structures
- 0153 1. 5.8.1 Client data used in WebAuthn signatures (dictionary
- 0154 CollectedClientData)
- 0155 2. 5.8.2 Credential Type enumeration (enum
- 0156 PublicKeyCredentialType)
- 0157 3. 5.8.3 Credential Descriptor (dictionary
- 0158 PublicKeyCredentialDescriptor)
- 0159 4. 5.8.4 Authenticator Transport enumeration (enum
- 0160 AuthenticatorTransport)
- 0161 5. 5.8.5 Cryptographic Algorithm Identifier (typedef
- 0162 COSEAlgorithmIdentifier)
- 0163 6. 5.8.6 User Verification Requirement enumeration (enum

- 0137 2. 2.2 Authenticators
- 0138 1. 2.2.1 Backwards Compatibility with FIDO U2F
- 0139 3. 2.3 Relying Parties
- 0140 4. 2.4 All Conformance Classes
- 0141 3. 3 Dependencies
- 0142 4. 4 Terminology
- 0143 5. 5 Web Authentication API
- 0144 1. 5.1 PublicKeyCredential Interface
- 0145 1. 5.1.1 CredentialCreationOptions Dictionary Extension
- 0146 2. 5.1.2 CredentialRequestOptions Dictionary Extension
- 0147 3. 5.1.3 Create a new credential - PublicKeyCredential's
- 0148 [[Create]](origin, options, sameOriginWithAncestors)
- 0149 method
- 0150 4. 5.1.4 Use an existing credential to make an assertion -
- 0151 PublicKeyCredential's [[Get]](options) method
- 0152 1. 5.1.4.1 PublicKeyCredential's
- 0153 [[DiscoverFromExternalSource]](origin, options,
- 0154 sameOriginWithAncestors) method
- 0155 5. 5.1.5 Store an existing credential -
- 0156 PublicKeyCredential's [[Store]](credential,
- 0157 sameOriginWithAncestors) method
- 0158 6. 5.1.6 Preventing silent access to an existing credential
- 0159 - PublicKeyCredential's
- 0160 [[preventSilentAccess]](credential,
- 0161 sameOriginWithAncestors) method
- 0162 7. 5.1.7 Availability of User-Verifying Platform
- 0163 Authenticator - PublicKeyCredential's
- 0164 isUserVerifyingPlatformAuthenticatorAvailable() method
- 0165 2. 5.2 Authenticator Responses (interface AuthenticatorResponse)
- 0166 1. 5.2.1 Information about Public Key Credential (interface
- 0167 AuthenticatorAttestationResponse)
- 0168 2. 5.2.2 Web Authentication Assertion (interface
- 0169 AuthenticatorAssertionResponse)
- 0170 3. 5.3 Parameters for Credential Generation (dictionary
- 0171 PublicKeyCredentialParameters)
- 0172 4. 5.4 Options for Credential Creation (dictionary
- 0173 PublicKeyCredentialCreationOptions)
- 0174 1. 5.4.1 Public Key Entity Description (dictionary
- 0175 PublicKeyCredentialEntity)
- 0176 2. 5.4.2 RP Parameters for Credential Generation (dictionary
- 0177 PublicKeyCredentialRpEntity)
- 0178 3. 5.4.3 User Account Parameters for Credential Generation
- 0179 (dictionary PublicKeyCredentialUserEntity)
- 0180 4. 5.4.4 Authenticator Selection Criteria (dictionary
- 0181 AuthenticatorSelectionCriteria)
- 0182 5. 5.4.5 Authenticator Attachment enumeration (enum
- 0183 AuthenticatorAttachment)
- 0184 6. 5.4.6 Attestation Conveyance Preference enumeration (enum
- 0185 AttestationConveyancePreference)
- 0186 5. 5.5 Options for Assertion Generation (dictionary
- 0187 PublicKeyCredentialRequestOptions)
- 0188 6. 5.6 Abort operations with AbortSignal
- 0189 7. 5.7 Authentication Extensions Client Inputs (typedef
- 0190 AuthenticationExtensionsClientInputs)
- 0191 8. 5.8 Authentication Extensions Client Outputs (typedef
- 0192 AuthenticationExtensionsClientOutputs)
- 0193 9. 5.9 Authentication Extensions Authenticator Inputs (typedef
- 0194 AuthenticationExtensionsAuthenticatorInputs)
- 0195 10. 5.10 Supporting Data Structures
- 0196 1. 5.10.1 Client data used in WebAuthn signatures
- 0197 (dictionary CollectedClientData)
- 0198 2. 5.10.2 Credential Type enumeration (enum
- 0199 PublicKeyCredentialType)
- 0200 3. 5.10.3 Credential Descriptor (dictionary
- 0201 PublicKeyCredentialDescriptor)
- 0202 4. 5.10.4 Authenticator Transport enumeration (enum
- 0203 AuthenticatorTransport)
- 0204 5. 5.10.5 Cryptographic Algorithm Identifier (typedef
- 0205 COSEAlgorithmIdentifier)
- 0206 6. 5.10.6 User Verification Requirement enumeration (enum

- 0164 UserVerificationRequirement)
- 0165 6. 6 WebAuthn Authenticator model
- 0166 1. 6.1 Authenticator data
- 0167 1. 6.1.1 Signature Counter Considerations
- 0168 2. 6.2 Authenticator operations
- 0169 1. 6.2.1 The authenticatorMakeCredential operation
- 0170 2. 6.2.2 The authenticatorGetAssertion operation
- 0171 3. 6.2.3 The authenticatorCancel operation
- 0172 3. 6.3 Attestation
- 0173 1. 6.3.1 Attested credential data
- 2. 6.3.2 Attestation Statement Formats
- 0174 3. 6.3.3 Attestation Types
- 0175 4. 6.3.4 Generating an Attestation Object
- 0176 5. 6.3.5 Security Considerations
- 0177 1. 6.3.5.1 Privacy
- 0178 2. 6.3.5.2 Attestation Certificate and Attestation
- 0179 Certificate CA Compromise
- 0180 3. 6.3.5.3 Attestation Certificate Hierarchy
- 0181 7. 7 Relying Party Operations
- 0182 1. 7.1 Registering a new credential
- 0183 2. 7.2 Verifying an authentication assertion
- 0184 8. 8 Defined Attestation Statement Formats
- 0185 1. 8.1 Attestation Statement Format Identifiers
- 0186 2. 8.2 Packed Attestation Statement Format
- 0187 1. 8.2.1 Packed attestation statement certificate
- 0188 requirements
- 0189 3. 8.3 TPM Attestation Statement Format
- 0190 1. 8.3.1 TPM attestation statement certificate requirements
- 0191 4. 8.4 Android Key Attestation Statement Format
- 0192 5. 8.5 Android SafetyNet Attestation Statement Format
- 0193 6. 8.6 FIDO U2F Attestation Statement Format
- 0194 9. 9 WebAuthn Extensions
- 0195 1. 9.1 Extension Identifiers
- 0196 2. 9.2 Defining extensions
- 0197 3. 9.3 Extending request parameters
- 0198 4. 9.4 Client extension processing
- 0199 5. 9.5 Authenticator extension processing
- 0200 6. 9.6 Example Extension
- 0201 10. 10 Defined Extensions
- 0202 1. 10.1 FIDO AppID Extension (appid)
- 0203 2. 10.2 Simple Transaction Authorization Extension (txAuthSimple)
- 0204 3. 10.3 Generic Transaction Authorization Extension
- 0205 (txAuthGeneric)
- 0206 4. 10.4 Authenticator Selection Extension (authnSel)
- 0207 5. 10.5 Supported Extensions Extension (exts)
- 0208 6. 10.6 User Verification Index Extension (uvi)
- 0209 7. 10.7 Location Extension (loc)
- 0210 8. 10.8 User Verification Method Extension (uvm)
- 0211 11. 11 IANA Considerations
- 0212 1. 11.1 WebAuthn Attestation Statement Format Identifier
- 0213 Registrations
- 0214 2. 11.2 WebAuthn Extension Identifier Registrations
- 0215 3. 11.3 COSE Algorithm Registrations
- 0216 12. 12 Sample scenarios
- 0217 1. 12.1 Registration
- 0218 2. 12.2 Registration Specifically with User Verifying Platform
- 0219 Authenticator
- 0220 3. 12.3 Authentication
- 0221 4. 12.4 Aborting Authentication Operations
- 0222 5. 12.5 Decommissioning
- 0223 13. 13 Security Considerations
- 0224 1. 13.1 Cryptographic Challenges
- 0225 14. 14 Acknowledgements
- 0226 15. Index
- 0227

- 0207 UserVerificationRequirement)
- 0208 6. 6 WebAuthn Authenticator Model
- 0209 1. 6.1 Authenticator data
- 0210 1. 6.1.1 Signature Counter Considerations
- 0211 2. 6.2 Authenticator operations
- 0212 1. 6.2.1 Lookup Credential Source by Credential ID algorithm
- 0213 2. 6.2.2 The authenticatorMakeCredential operation
- 0214 3. 6.2.3 The authenticatorGetAssertion operation
- 0215 4. 6.2.4 The authenticatorCancel operation
- 0216 3. 6.3 Attestation
- 0217 1. 6.3.1 Attested credential data
- 0218 1. 6.3.1.1 Examples of credentialPublicKey Values
- 0219 encoded in COSE\_Key format
- 0220 2. 6.3.2 Attestation Statement Formats
- 0221 3. 6.3.3 Attestation Types
- 0222 4. 6.3.4 Generating an Attestation Object
- 0223 5. 6.3.5 Signature Formats for Packed Attestation, FIDO U2F
- 0224 Attestation, and Assertion Signatures
- 0225 7. 7 Relying Party Operations
- 0226 1. 7.1 Registering a new credential
- 0227 2. 7.2 Verifying an authentication assertion
- 0228 8. 8 Defined Attestation Statement Formats
- 0229 1. 8.1 Attestation Statement Format Identifiers
- 0230 2. 8.2 Packed Attestation Statement Format
- 0231 1. 8.2.1 Packed attestation statement certificate
- 0232 requirements
- 0233 3. 8.3 TPM Attestation Statement Format
- 0234 1. 8.3.1 TPM attestation statement certificate requirements
- 0235 4. 8.4 Android Key Attestation Statement Format
- 0236 5. 8.5 Android SafetyNet Attestation Statement Format
- 0237 6. 8.6 FIDO U2F Attestation Statement Format
- 0238 7. 8.7 None Attestation Statement Format
- 0239 9. 9 WebAuthn Extensions
- 0240 1. 9.1 Extension Identifiers
- 0241 2. 9.2 Defining extensions
- 0242 3. 9.3 Extending request parameters
- 0243 4. 9.4 Client extension processing
- 0244 5. 9.5 Authenticator extension processing
- 0245 10. 10 Defined Extensions
- 0246 1. 10.1 FIDO AppID Extension (appid)
- 0247 2. 10.2 Simple Transaction Authorization Extension (txAuthSimple)
- 0248 3. 10.3 Generic Transaction Authorization Extension
- 0249 (txAuthGeneric)
- 0250 4. 10.4 Authenticator Selection Extension (authnSel)
- 0251 5. 10.5 Supported Extensions Extension (exts)
- 0252 6. 10.6 User Verification Index Extension (uvi)
- 0253 7. 10.7 Location Extension (loc)
- 0254 8. 10.8 User Verification Method Extension (uvm)
- 0255 9. 10.9 Biometric Authenticator Performance Bounds Extension
- 0256 (biometricPerfBounds)
- 0257 11. 11 IANA Considerations
- 0258 1. 11.1 WebAuthn Attestation Statement Format Identifier
- 0259 Registrations
- 0260 2. 11.2 WebAuthn Extension Identifier Registrations
- 0261 3. 11.3 COSE Algorithm Registrations
- 0262 12. 12 Sample scenarios
- 0263 1. 12.1 Registration
- 0264 2. 12.2 Registration Specifically with User Verifying Platform
- 0265 Authenticator
- 0266 3. 12.3 Authentication
- 0267 4. 12.4 Aborting Authentication Operations
- 0268 5. 12.5 Decommissioning
- 0269 13. 13 Security Considerations
- 0270 1. 13.1 Cryptographic Challenges
- 0271 2. 13.2 Attestation Security Considerations
- 0272 1. 13.2.1 Attestation Certificate Hierarchy

- 0228 1. Terms defined by this specification
- 0229 2. Terms defined by reference
- 0230 16. References
  - 0231 1. Normative References
  - 0232 2. Informative References
- 0233 17. IDL Index
- 0234 18. Issues Index

0235

0236 1. Introduction

0237

0238 This section is not normative.

0239

0240 This specification defines an API enabling the creation and use of

0241 strong, attested, scoped, public key-based credentials by web

0242 applications, for the purpose of strongly authenticating users. A

0243 public key credential is created and stored by an authenticator at the

0244 behest of a Relying Party, subject to user consent. Subsequently, the

0245 public key credential can only be accessed by origins belonging to that

0246 Relying Party. This scoping is enforced jointly by conforming User

0247 Agents and authenticators. Additionally, privacy across Relying Parties

0248 is maintained; Relying Parties are not able to detect any properties,

0249 or even the existence, of credentials scoped to other Relying Parties.

0250

0251 Relying Parties employ the Web Authentication API during two distinct,

0252 but related, ceremonies involving a user. The first is Registration,

0253 where a public key credential is created on an authenticator, and

0254 associated by a Relying Party with the present user's account (the

0255 account **may** already exist or **may** be created at this time). The second

0256 is Authentication, where the Relying Party is presented with an

0257 Authentication Assertion proving the presence and consent of the user

0258 who registered the public key credential. Functionally, the Web

0259 Authentication API comprises a PublicKeyCredential which extends the

0260 Credential Management API [CREDENTIAL-MANAGEMENT-1], and infrastructure

0261 which allows those credentials to be used with

0262 navigator.credentials.create() and navigator.credentials.get(). The

0263 former is used during Registration, and the latter during

0264 Authentication.

0265

0266 Broadly, compliant authenticators protect public key credentials, and

0267 interact with user agents to implement the Web Authentication API. Some

0268 authenticators **may** run on the same computing device (e.g., smart phone,

0269 tablet, desktop PC) as the user agent is running on. For instance, such

0270 an authenticator might consist of a Trusted Execution Environment (TEE)

0271 applet, a Trusted Platform Module (TPM), or a Secure Element (SE)

0272 integrated into the computing device in conjunction with some means for

0273 user verification, along with appropriate platform software to mediate

0274 access to these components' functionality. Other authenticators **may**

0275 operate autonomously from the computing device running the user agent,

0276 and be accessed over a transport such as Universal Serial Bus (USB),

0277 Bluetooth Low Energy (BLE) or Near Field Communications (NFC).

0278

0279 1.1. Use Cases

0280

0281 The below use case scenarios illustrate use of two very different types

0282 of authenticators, as well as outline further scenarios. Additional

0283 scenarios, including sample code, are given later in 12 Sample

0284 scenarios.

0285

0286 1.1.1. Registration

0287

- 0273 2. 13.2.2 Attestation Certificate and Attestation
- 0274 Certificate CA Compromise
- 0275 3. 13.3 credentialId Unsigned
- 0276 4. 13.4 Browser Permissions Framework and Extensions
- 0277 14. 14 Privacy Considerations
  - 0278 1. 14.1 Attestation Privacy
  - 0279 2. 14.2 Registration Ceremony Privacy
  - 0280 3. 14.3 Authentication Ceremony Privacy
- 0281 15. 15 Acknowledgements
- 0282 16. Index
  - 0283 1. Terms defined by this specification
  - 0284 2. Terms defined by reference
- 0285 17. References
  - 0286 1. Normative References
  - 0287 2. Informative References
- 0288 18. IDL Index
- 0289 19. Issues Index

0290

0291 1. Introduction

0292

0293 This section is not normative.

0294

0295 This specification defines an API enabling the creation and use of

0296 strong, attested, scoped, public key-based credentials by web

0297 applications, for the purpose of strongly authenticating users. A

0298 public key credential is created and stored by an authenticator at the

0299 behest of a Relying Party, subject to user consent. Subsequently, the

0300 public key credential can only be accessed by origins belonging to that

0301 Relying Party. This scoping is enforced jointly by conforming User

0302 Agents and authenticators. Additionally, privacy across Relying Parties

0303 is maintained; Relying Parties are not able to detect any properties,

0304 or even the existence, of credentials scoped to other Relying Parties.

0305

0306 Relying Parties employ the Web Authentication API during two distinct,

0307 but related, ceremonies involving a user. The first is Registration,

0308 where a public key credential is created on an authenticator, and

0309 associated by a Relying Party with the present user's account (the

0310 account **MAY** already exist or **MAY** be created at this time). The second

0311 is Authentication, where the Relying Party is presented with an

0312 Authentication Assertion proving the presence and consent of the user

0313 who registered the public key credential. Functionally, the Web

0314 Authentication API comprises a PublicKeyCredential which extends the

0315 Credential Management API [CREDENTIAL-MANAGEMENT-1], and infrastructure

0316 which allows those credentials to be used with

0317 navigator.credentials.create() and navigator.credentials.get(). The

0318 former is used during Registration, and the latter during

0319 Authentication.

0320

0321 Broadly, compliant authenticators protect public key credentials, and

0322 interact with user agents to implement the Web Authentication API. Some

0323 authenticators **MAY** run on the same computing device (e.g., smart phone,

0324 tablet, desktop PC) as the user agent is running on. For instance, such

0325 an authenticator might consist of a Trusted Execution Environment (TEE)

0326 applet, a Trusted Platform Module (TPM), or a Secure Element (SE)

0327 integrated into the computing device in conjunction with some means for

0328 user verification, along with appropriate platform software to mediate

0329 access to these components' functionality. Other authenticators **MAY**

0330 operate autonomously from the computing device running the user agent,

0331 and be accessed over a transport such as Universal Serial Bus (USB),

0332 Bluetooth Low Energy (BLE) or Near Field Communications (NFC).

0333

0334 1.1. Use Cases

0335

0336 The below use case scenarios illustrate use of two very different types

0337 of authenticators, as well as outline further scenarios. Additional

0338 scenarios, including sample code, are given later in 12 Sample

0339 scenarios.

0340

0341 1.1.1. Registration

0342

028E \* On a phone:  
 028E + User navigates to example.com in a browser and signs in to an  
 029C existing account using whatever method they have been using  
 0291 (possibly a legacy method such as a password), or creates a  
 0292 new account.  
 0293 + The phone prompts, "Do you want to register this device with  
 0294 example.com?"  
 0295 + User agrees.  
 0296 + The phone prompts the user for a previously configured  
 0297 authorization gesture (PIN, biometric, etc.); the user  
 029E provides this.  
 029E + Website shows message, "Registration complete."

### 1.1.2. Authentication

0303 \* On a laptop or desktop:  
 0304 + User navigates to example.com in a browser, sees an option to  
 0305 "Sign in with your phone."  
 0306 + User chooses this option and gets a message from the browser,  
 0307 "Please complete this action on your phone."  
 0308 \* Next, on their phone:  
 0309 + User sees a discrete prompt or notification, "Sign in to  
 0310 example.com."  
 0311 + User selects this prompt / notification.  
 0312 + User is shown a list of their example.com identities, e.g.,  
 0313 "Sign in as Alice / Sign in as Bob."  
 0314 + User picks an identity, is prompted for an authorization  
 0315 gesture (PIN, biometric, etc.) and provides this.  
 0316 \* Now, back on the laptop:  
 0317 + Web page shows that the selected user is signed in, and  
 0318 navigates to the signed-in page.

### 1.1.3. Other use cases and configurations

0320 A variety of additional use cases and configurations are also possible,  
0321 including (but not limited to):

- 0322 \* A user navigates to example.com on their laptop, is guided through  
0323 a flow to create and register a credential on their phone.
- 0324 \* A user obtains an discrete, roaming authenticator, such as a "fob"  
0325 with USB or USB+NFC/BLE connectivity options, loads example.com in  
0326 their browser on a laptop or phone, and is guided through a flow to  
0327 create and register a credential on the fob.
- 0328 \* A Relying Party prompts the user for their authorization gesture in  
0329 order to authorize a single transaction, such as a payment or other  
0330 financial transaction.

## 2. Conformance

0334 This specification defines three conformance classes. Each of these  
0335 classes is specified so that conforming members of the class are secure  
0336 against non-conforming or hostile members of the other classes.

### 2.1. User Agents

0340 A User Agent MUST behave as described by 5 Web Authentication API in  
0341 order to be considered conformant. Conforming User Agents MAY implement  
0342 algorithms given in this specification in any way desired, so long as  
0343 the end result is indistinguishable from the result that would be  
0344 obtained by the specification's algorithms.

0345 A conforming User Agent MUST also be a conforming implementation of the  
0346 IDL fragments of this specification, as described in the "Web IDL"  
0347 specification. [WebIDL-1]

### 2.2. Authenticators

0352 An authenticator MUST provide the operations defined by 6 WebAuthn  
0353 Authenticator model, and those operations MUST behave as described  
0354 there. This is a set of functional and security requirements for an  
0355 authenticator to be usable by a Conforming User Agent.

0343 \* On a phone:  
 0344 + User navigates to example.com in a browser and signs in to an  
 034E existing account using whatever method they have been using  
 034F (possibly a legacy method such as a password), or creates a  
 0347 new account.  
 034E + The phone prompts, "Do you want to register this device with  
 034F example.com?"  
 0350 + User agrees.  
 0351 + The phone prompts the user for a previously configured  
 0352 authorization gesture (PIN, biometric, etc.); the user  
 0353 provides this.  
 0354 + Website shows message, "Registration complete."

### 1.1.2. Authentication

035E \* On a laptop or desktop:  
 035F + User navigates to example.com in a browser, sees an option to  
 0360 "Sign in with your phone."  
 0361 + User chooses this option and gets a message from the browser,  
 0362 "Please complete this action on your phone."  
 0363 \* Next, on their phone:  
 0364 + User sees a discrete prompt or notification, "Sign in to  
 0365 example.com."  
 0366 + User selects this prompt / notification.  
 0367 + User is shown a list of their example.com identities, e.g.,  
 0368 "Sign in as Alice / Sign in as Bob."  
 0369 + User picks an identity, is prompted for an authorization  
 0370 gesture (PIN, biometric, etc.) and provides this.  
 0371 \* Now, back on the laptop:  
 0372 + Web page shows that the selected user is signed in, and  
 0373 navigates to the signed-in page.

### 1.1.3. Other use cases and configurations

0374 A variety of additional use cases and configurations are also possible,  
0375 including (but not limited to):

- 0376 \* A user navigates to example.com on their laptop, is guided through  
0377 a flow to create and register a credential on their phone.
- 0378 \* A user obtains a discrete, roaming authenticator, such as a "fob"  
0379 with USB or USB+NFC/BLE connectivity options, loads example.com in  
0380 their browser on a laptop or phone, and is guided through a flow to  
0381 create and register a credential on the fob.
- 0382 \* A Relying Party prompts the user for their authorization gesture in  
0383 order to authorize a single transaction, such as a payment or other  
0384 financial transaction.

## 2. Conformance

0385 This specification defines three conformance classes. Each of these  
0386 classes is specified so that conforming members of the class are secure  
0387 against non-conforming or hostile members of the other classes.

### 2.1. User Agents

0388 A User Agent MUST behave as described by 5 Web Authentication API in  
0389 order to be considered conformant. Conforming User Agents MAY implement  
0390 algorithms given in this specification in any way desired, so long as  
0391 the end result is indistinguishable from the result that would be  
0392 obtained by the specification's algorithms.

0393 A conforming User Agent MUST also be a conforming implementation of the  
0394 IDL fragments of this specification, as described in the "Web IDL"  
0395 specification. [WebIDL-1]

### 2.2. Authenticators

0396 An authenticator MUST provide the operations defined by 6 WebAuthn  
0397 Authenticator Model, and those operations MUST behave as described  
0398 there. This is a set of functional and security requirements for an  
0399 authenticator to be usable by a Conforming User Agent.

035E As described in 1.1 Use Cases, an authenticator may be implemented in  
035F the operating system underlying the User Agent, or in external  
036C hardware, or a combination of both.  
0361  
0362

### 2.3. Relying Parties

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

### 3. Dependencies

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

#### Base64url encoding

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

#### CBOR

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

#### CDDL

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

#### COSE

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

#### Credential Management

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

#### DOM

DOMException and the DOMException values used in this specification are defined in [DOM4].

#### ECMAScript

%ArrayBuffer% is defined in [ECMAScript].

#### HTML

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

#### Web IDL

Many of the interface definitions and all of the IDL in this specification depend on [WebIDL-1]. This updated version of the

0363  
0364  
0365  
0366

0367  
0368  
0369  
0370  
0371  
0372  
0373  
0374  
0375  
0376  
0377  
0378  
0379  
0380  
0381  
0382  
0383

0384  
0385  
0386  
0387  
0388  
0389  
0390  
0391  
0392  
0393  
0394  
0395  
0396  
0397  
0398  
0399  
0400  
0401  
0402  
0403  
0404  
0405  
0406  
0407  
0408  
0409  
0410  
0411  
0412

0413 As described in 1.1 Use Cases, an authenticator may be implemented in  
0414 the operating system underlying the User Agent, or in external  
041E hardware, or a combination of both.  
041F  
0417

### 2.2.1. Backwards Compatibility with FIDO U2F

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

### 2.3. Relying Parties

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

### 2.4. All Conformance Classes

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

0418  
0419  
0420  
0421  
0422  
0423  
0424  
042E  
042E  
0427  
0428  
0429  
0430  
0431  
0432  
0433  
0434  
0435  
0436  
0437  
043E  
043E  
043E  
044C  
0441  
0442  
0443  
0444  
044E  
044E  
0447  
044E  
044E  
045C  
0451  
0452  
0453  
0454  
045E  
045E  
0457  
045E  
045E  
046C  
0461  
0462  
0463  
0464  
046E  
046E  
0467  
046E  
046E  
047C  
0471  
0472  
0473  
0474  
047E  
047E  
0477  
047E  
047E  
048C  
0481  
0482

### 3. Dependencies

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

#### Base64url encoding

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

#### CBOR

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

#### CDDL

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

#### COSE

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

#### Credential Management

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

#### DOM

DOMException and the DOMException values used in this specification are defined in [DOM4].

#### ECMAScript

%ArrayBuffer% is defined in [ECMAScript].

#### HTML

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

#### Web IDL

Many of the interface definitions and all of the IDL in this specification depend on [WebIDL-1]. This updated version of the

0413 Web IDL standard adds support for Promises, which are now the  
0414 preferred mechanism for asynchronous interaction in all new web  
0415 APIs.  
0416

0417 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",  
0418 "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this  
0419 document are to be interpreted as described in [RFC2119].  
0420

#### 0421 4. Terminology

##### 0422 Assertion

0423 See Authentication Assertion.  
0424

##### 0425 Attestation

0426 Generally, attestation is a statement serving to bear witness,  
0427 confirm, or authenticate. In the WebAuthn context, attestation  
0428 is employed to attest to the provenance of an authenticator and  
0429 the data it emits; including, for example: credential IDs,  
0430 credential key pairs, signature counters, etc. An attestation  
0431 statement is conveyed in an attestation object during  
0432 registration. See also 6.3 Attestation and Figure 3. Whether or  
0433 how the client platform conveys the attestation statement and  
0434 AAGUID portions of the attestation object to the Relying Party  
0435 is described by attestation conveyance.  
0436

##### 0437 Attestation Certificate

0438 A X.509 Certificate for the attestation key pair used by an  
0439 authenticator to attest to its manufacture and capabilities. At  
0440 registration time, the authenticator uses the attestation  
0441 private key to sign the Relying Party-specific credential public  
0442 key (and additional data) that it generates and returns via the  
0443 authenticatorMakeCredential operation. Relying Parties use the  
0444 attestation public key conveyed in the attestation certificate  
0445 to verify the attestation signature. Note that in the case of  
0446 self attestation, the authenticator has no distinct attestation  
0447 key pair nor attestation certificate, see self attestation for  
0448 details.  
0449

##### 0450 Authentication

0451 The ceremony where a user, and the user's computing device(s)  
0452 (containing at least one authenticator) work in concert to  
0453 cryptographically prove to an Relying Party that the user  
0454 controls the credential private key associated with a  
0455 previously-registered public key credential (see Registration).  
0456 Note that this includes a test of user presence or user  
0457 verification.  
0458

##### 0459 Authentication Assertion

0460 The cryptographically signed AuthenticatorAssertionResponse  
0461 object returned by an authenticator as the result of a  
0462 authenticatorGetAssertion operation.  
0463

0464 This corresponds to the [CREDENTIAL-MANAGEMENT-1]  
0465 specification's single-use credentials.  
0466

##### 0467 Authenticator

0468 A cryptographic entity used by a WebAuthn Client to (i) generate  
0469 a public key credential and register it with a Relying Party,  
0470 and (ii) authenticate by potentially verifying the user, and  
0471 then cryptographically signing and returning, in the form of an  
0472 Authentication Assertion, a challenge and other data presented  
0473 by a Relying Party (in concert with the WebAuthn Client).  
0474

##### 0475 Authorization Gesture

0476

0483 Web IDL standard adds support for Promises, which are now the  
0484 preferred mechanism for asynchronous interaction in all new web  
0485 APIs.  
0486

#### 0487 FIDO AppID

0488 The algorithms for determining the FacetID of a calling  
0489 application and determining if a caller's FacetID is authorized  
0490 for an AppID (used only in the appid extension) are defined by  
0491 [FIDO-APPID].  
0492

0493 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",  
0494 "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this  
0495 document are to be interpreted as described in [RFC2119].  
0496

#### 0497 4. Terminology

##### 0498 Assertion

0499 See Authentication Assertion.  
0500

##### 0501 Attestation

0502 Generally, attestation is a statement serving to bear witness,  
0503 confirm, or authenticate. In the WebAuthn context, attestation  
0504 is employed to attest to the provenance of an authenticator and  
0505 the data it emits; including, for example: credential IDs,  
0506 credential key pairs, signature counters, etc. An attestation  
0507 statement is conveyed in an attestation object during  
0508 registration. See also 6.3 Attestation and Figure 3. Whether or  
0509 how the client platform conveys the attestation statement and  
0510 AAGUID portions of the attestation object to the Relying Party  
0511 is described by attestation conveyance.  
0512

##### 0513 Attestation Certificate

0514 A X.509 Certificate for the attestation key pair used by an  
0515 authenticator to attest to its manufacture and capabilities. At  
0516 registration time, the authenticator uses the attestation  
0517 private key to sign the Relying Party-specific credential public  
0518 key (and additional data) that it generates and returns via the  
0519 authenticatorMakeCredential operation. Relying Parties use the  
0520 attestation public key conveyed in the attestation certificate  
0521 to verify the attestation signature. Note that in the case of  
0522 self attestation, the authenticator has no distinct attestation  
0523 key pair nor attestation certificate, see self attestation for  
0524 details.  
0525

##### 0526 Authentication

0527 The ceremony where a user, and the user's computing device(s)  
0528 (containing at least one authenticator) work in concert to  
0529 cryptographically prove to a Relying Party that the user  
0530 controls the credential private key associated with a  
0531 previously-registered public key credential (see Registration).  
0532 Note that this includes a test of user presence or user  
0533 verification.  
0534

##### 0535 Authentication Assertion

0536 The cryptographically signed AuthenticatorAssertionResponse  
0537 object returned by an authenticator as the result of an  
0538 authenticatorGetAssertion operation.  
0539

0540 This corresponds to the [CREDENTIAL-MANAGEMENT-1]  
0541 specification's single-use credentials.  
0542

##### 0543 Authenticator

0544 A cryptographic entity used by a WebAuthn Client to (i) generate  
0545 a public key credential and register it with a Relying Party,  
0546 and (ii) authenticate by potentially verifying the user, and  
0547 then cryptographically signing and returning, in the form of an  
0548 Authentication Assertion, a challenge and other data presented  
0549 by a Relying Party (in concert with the WebAuthn Client).  
0550

##### 0551 Authorization Gesture

0552

0477 An authorization gesture is a physical interaction performed by  
 0478 a user with an authenticator as part of a ceremony, such as  
 0479 registration or authentication. By making such an authorization  
 0480 gesture, a user provides consent for (i.e., authorizes) a  
 0481 ceremony to proceed. This **may** involve user verification if the  
 0482 employed authenticator is capable, or it **may** involve a simple  
 0483 test of user presence.  
 0484  
**Biometric Recognition**  
 0485 The automated recognition of individuals based on their  
 0486 biological and behavioral characteristics  
 0487 [ISOBiometricVocabulary].  
 0488  
**Ceremony**  
 0490 The concept of a ceremony [Ceremony] is an extension of the  
 0491 concept of a network protocol, with human nodes alongside  
 0492 computer nodes and with communication links that include user  
 0493 interface(s), human-to-human communication, and transfers of  
 0494 physical objects that carry data. What is out-of-band to a  
 0495 protocol is in-band to a ceremony. In this specification,  
 0496 Registration and Authentication are ceremonies, and an  
 0497 authorization gesture is often a component of those ceremonies.  
**Client**  
 0498 See Conforming User Agent.  
**Client-Side**  
 0500 This refers in general to the combination of the user's platform  
 0501 device, user agent, authenticators, and everything gluing it all  
 0502 together.  
**Client-side-resident Credential Private Key**  
 0503 A Client-side-resident Credential Private Key is stored either  
 0504 on the client platform, or in some cases on the authenticator  
 0505 itself, e.g., in the case of a discrete first-factor roaming  
 0506 authenticator. Such client-side credential private key storage  
 0507 has the property that the authenticator is able to select the  
 0508 credential private key given only an RP ID, possibly with user  
 0509 assistance (e.g., by providing the user a pick list of  
 0510 credentials associated with the RP ID). By definition, the  
 0511 private key is always exclusively controlled by the  
 0512 Authenticator. In the case of a Client-side-resident Credential  
 0513 Private Key, the Authenticator might offload storage of wrapped  
 0514 key material to the client platform, but the client platform is  
 0515 not expected to offload the key storage to remote entities (e.g.  
 0516 RP Server).  
**Conforming User Agent**  
 0517 A user agent implementing, in conjunction with the underlying  
 0518 platform, the Web Authentication API and algorithms given in  
 0519 this specification, and handling communication between  
 0520 authenticators and Relying Parties.  
**Credential ID**  
 0521 A probabilistically-unique byte sequence identifying a public  
 0522 key credential source and its authentication assertions.  
 0523  
 0524 Credential IDs are generated by authenticators in two forms:  
 0525  
 0526 1. At least 16 bytes that include at least 100 bits of entropy,  
 0527 or  
 0528 2. The public key credential source, without its Credential ID,  
 0529 encrypted so only its managing authenticator can decrypt it.  
 0530 This form allows the authenticator to be nearly stateless, by  
 0531 having the Relying Party store any necessary state.  
 0532 Note: [FIDO-UAF-AUTHNR-CMDS] includes guidance on encryption  
 0533 techniques under "Security Guidelines".  
 0534  
 0535  
 0536  
 0537  
 0538  
 0539  
 0540  
 0541  
 0542  
 0543

0553 An authorization gesture is a physical interaction performed by  
 0554 a user with an authenticator as part of a ceremony, such as  
 0555 registration or authentication. By making such an authorization  
 0556 gesture, a user provides consent for (i.e., authorizes) a  
 0557 ceremony to proceed. This **MAY** involve user verification if the  
 0558 employed authenticator is capable, or it **MAY** involve a simple  
 0559 test of user presence.  
 0560  
**Biometric Recognition**  
 0561 The automated recognition of individuals based on their  
 0562 biological and behavioral characteristics  
 0563 [ISOBiometricVocabulary].  
**Biometric Authenticator**  
 0564 Any authenticator that implements biometric recognition.  
**Ceremony**  
 0565 The concept of a ceremony [Ceremony] is an extension of the  
 0566 concept of a network protocol, with human nodes alongside  
 0567 computer nodes and with communication links that include user  
 0568 interface(s), human-to-human communication, and transfers of  
 0569 physical objects that carry data. What is out-of-band to a  
 0570 protocol is in-band to a ceremony. In this specification,  
 0571 Registration and Authentication are ceremonies, and an  
 0572 authorization gesture is often a component of those ceremonies.  
**Client**  
 0573 See [WebAuthn Client](#), Conforming User Agent.  
**Client-Side**  
 0574 This refers in general to the combination of the user's platform  
 0575 device, user agent, authenticators, and everything gluing it all  
 0576 together.  
**Client-side-resident Credential Private Key**  
 0577 A Client-side-resident Credential Private Key is stored either  
 0578 on the client platform, or in some cases on the authenticator  
 0579 itself, e.g., in the case of a discrete first-factor roaming  
 0580 authenticator. Such client-side credential private key storage  
 0581 has the property that the authenticator is able to select the  
 0582 credential private key given only an RP ID, possibly with user  
 0583 assistance (e.g., by providing the user a pick list of  
 0584 credentials associated with the RP ID). By definition, the  
 0585 private key is always exclusively controlled by the  
 0586 Authenticator. In the case of a Client-side-resident Credential  
 0587 Private Key, the Authenticator might offload storage of wrapped  
 0588 key material to the client platform, but the client platform is  
 0589 not expected to offload the key storage to remote entities (e.g.  
 0590 RP Server).  
**Conforming User Agent**  
 0591 A user agent implementing, in conjunction with the underlying  
 0592 platform, the Web Authentication API and algorithms given in  
 0593 this specification, and handling communication between  
 0594 authenticators and Relying Parties.  
**Credential ID**  
 0595 A probabilistically-unique byte sequence identifying a public  
 0596 key credential source and its authentication assertions.  
 0597  
 0598 Credential IDs are generated by authenticators in two forms:  
 0599  
 0600 1. At least 16 bytes that include at least 100 bits of entropy,  
 0601 or  
 0602 2. The public key credential source, without its Credential ID,  
 0603 encrypted so only its managing authenticator can decrypt it.  
 0604 This form allows the authenticator to be nearly stateless, by  
 0605 having the Relying Party store any necessary state.  
 0606 Note: [FIDO-UAF-AUTHNR-CMDS] includes guidance on encryption  
 0607 techniques under "Security Guidelines".  
 0608  
 0609  
 0610  
 0611  
 0612  
 0613  
 0614  
 0615  
 0616  
 0617  
 0618  
 0619  
 0620  
 0621  
 0622

0544 Relying Parties do not need to distinguish these two Credential  
 0545 ID forms.  
 0546  
 0547  
 0548 **Credential Public Key**  
 0549 The public key portion of an Relying Party-specific credential  
 0550 key pair, generated by an authenticator and returned to an  
 0551 Relying Party at registration time (see also public key  
 0552 credential). The private key portion of the credential key pair  
 0553 is known as the credential private key. Note that in the case of  
 0554 self attestation, the credential key pair is also used as the  
 0555 attestation key pair, see self attestation for details.  
 0556

0557 **Public Key Credential Source**  
 0558 A credential source ([CREDENTIAL-MANAGEMENT-1]) used by an  
 0559 authenticator to generate authentication assertions. A public  
 0560 key credential source has:

- + A Credential ID.
- + A credential private key.
- + The Relying Party Identifier for the Relying Party that created this credential source.
- + An optional user handle for the person who created this credential source.
- + Optional other information used by the authenticator to inform its UI. For example, this might include the user's

0570 displayName.

0571  
 0572 The authenticatorMakeCredential operation creates a public key  
 0573 credential source bound to a managing authenticator and returns  
 0574 the credential public key associated with its credential private  
 0575 key. The Relying Party can use this credential public key to  
 0576 verify the authentication assertions created by this public key  
 0577 credential source.  
 0578

0579 **Public Key Credential**  
 0580 Generically, a credential is data one entity presents to another  
 0581 in order to authenticate the former to the latter [RFC4949]. The  
 0582 term public key credential refers to one of: a public key  
 0583 credential source, the possibly-attested credential public key  
 0584 corresponding to a public key credential source, or an  
 0585 authentication assertion. Which one is generally determined by  
 0586 context.  
 0587

0588 **Note:** This is a willful violation of [RFC4949]. In English, a  
 0589 "credential" is both a) the thing presented to prove a statement  
 0590 and b) intended to be used multiple times. It's impossible to  
 0591 achieve both criteria securely with a single piece of data in a  
 0592 public key system. [RFC4949] chooses to define a credential as  
 0593 the thing that can be used multiple times (the public key),

0623 Relying Parties do not need to distinguish these two Credential  
 0624 ID forms.  
 0625  
 0626  
 0627  
 0628 **Credential Public Key**  
 0629 The public key portion of a Relying Party-specific credential  
 0630 key pair, generated by an authenticator and returned to a  
 0631 Relying Party at registration time (see also public key  
 0632 credential). The private key portion of the credential key pair  
 0633 is known as the credential private key. Note that in the case of  
 0634 self attestation, the credential key pair is also used as the  
 0635 attestation key pair, see self attestation for details.  
 0636

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

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

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

**id**  
 A Credential ID.

**privateKey**  
 The credential private key.

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

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

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

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

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

**Note:** This is a willful violation of [RFC4949]. In English, a "credential" is both a) the thing presented to prove a statement and b) intended to be used multiple times. It's impossible to achieve both criteria securely with a single piece of data in a public key system. [RFC4949] chooses to define a credential as the thing that can be used multiple times (the public key),

0594 while this specification gives "credential" the English term's  
 0595 flexibility. This specification uses more specific terms to  
 0596 identify the data related to an [RFC4949] credential:  
 0597  
 0598 "Authentication information" (possibly including a private key)  
 0599 Public key credential source  
 0600  
 0601 "Signed value"  
 0602 Authentication assertion  
 0603  
 0604 [RFC4949] "credential"  
 0605 Credential public key or attestation object  
 0606  
 0607 At registration time, the authenticator creates an asymmetric  
 0608 key pair, and stores its private key portion and information  
 0609 from the Relying Party into a public key credential source. The  
 0610 public key portion is returned to the Relying Party, who then  
 0611 stores it in conjunction with the present user's account.  
 0612 Subsequently, only that Relying Party, as identified by its RP  
 0613 ID, is able to employ the public key credential in  
 0614 authentication ceremonies, via the get() method. The Relying  
 0615 Party uses its stored copy of the credential public key to  
 0616 verify the resultant authentication assertion.  
 0617  
 0618 **Rate Limiting**  
 0619 The process (also known as throttling) by which an authenticator  
 0620 implements controls against brute force attacks by limiting the  
 0621 number of consecutive failed authentication attempts within a  
 0622 given period of time. If the limit is reached, the authenticator  
 0623 should impose a delay that increases exponentially with each  
 0624 successive attempt, or disable the current authentication  
 0625 modality and offer a different authentication factor if  
 0626 available. Rate limiting is often implemented as an aspect of  
 0627 user verification.  
 0628  
 0629 **Registration**  
 0630 The ceremony where a user, a Relying Party, and the user's  
 0631 computing device(s) (containing at least one authenticator) work  
 0632 in concert to create a public key credential and associate it  
 0633 with the user's Relying Party account. Note that this includes  
 0634 employing a test of user presence or user verification.  
 0635  
 0636 **Relying Party**  
 0637 The entity whose web application utilizes the Web Authentication  
 0638 API to register and authenticate users. See Registration and  
 0639 Authentication, respectively.  
 0640  
 0641 Note: While the term Relying Party is used in other contexts  
 0642 (e.g., X.509 and OAuth), an entity acting as a Relying Party in  
 0643 one context is not necessarily a Relying Party in other  
 0644 contexts.  
 0645  
 0646 **Relying Party Identifier**  
 0647 **RP ID**  
 0648 A valid domain string that identifies the Relying Party on whose  
 0649 behalf a given registration or authentication ceremony is being  
 0650 performed. A public key credential can only be used for  
 0651 authentication with the same entity (as identified by RP ID) it  
 0652 was registered with. By default, the RP ID for a WebAuthn  
 0653 operation is set to the caller's origin's effective domain. This  
 0654 default MAY be overridden by the caller, as long as the  
 0655 caller-specified RP ID value is a registrable domain suffix of  
 0656 or is equal to the caller's origin's effective domain. See also  
 0657 5.1.3 Create a new credential - PublicKeyCredential's  
 0658 [[Create]](origin, options, sameOriginWithAncestors) method and  
 0659 5.1.4 Use an existing credential to make an assertion -  
 0660 PublicKeyCredential's [[Get]](options) method.  
 0661  
 0662 Note: A Public key credential's scope is for a Relying Party's  
 0663 origin, with the following restrictions and relaxations:

0693 while this specification gives "credential" the English term's  
 0694 flexibility. This specification uses more specific terms to  
 0695 identify the data related to an [RFC4949] credential:  
 0696  
 0697 "Authentication information" (possibly including a private key)  
 0698 Public key credential source  
 0699  
 0700 "Signed value"  
 0701 Authentication assertion  
 0702  
 0703 [RFC4949] "credential"  
 0704 Credential public key or attestation object  
 0705  
 0706 At registration time, the authenticator creates an asymmetric  
 0707 key pair, and stores its private key portion and information  
 0708 from the Relying Party into a public key credential source. The  
 0709 public key portion is returned to the Relying Party, who then  
 0710 stores it in conjunction with the present user's account.  
 0711 Subsequently, only that Relying Party, as identified by its RP  
 0712 ID, is able to employ the public key credential in  
 0713 authentication ceremonies, via the get() method. The Relying  
 0714 Party uses its stored copy of the credential public key to  
 0715 verify the resultant authentication assertion.  
 0716  
 0717 **Rate Limiting**  
 0718 The process (also known as throttling) by which an authenticator  
 0719 implements controls against brute force attacks by limiting the  
 0720 number of consecutive failed authentication attempts within a  
 0721 given period of time. If the limit is reached, the authenticator  
 0722 should impose a delay that increases exponentially with each  
 0723 successive attempt, or disable the current authentication  
 0724 modality and offer a different authentication factor if  
 0725 available. Rate limiting is often implemented as an aspect of  
 0726 user verification.  
 0727  
 0728 **Registration**  
 0729 The ceremony where a user, a Relying Party, and the user's  
 0730 computing device(s) (containing at least one authenticator) work  
 0731 in concert to create a public key credential and associate it  
 0732 with the user's Relying Party account. Note that this includes  
 0733 employing a test of user presence or user verification.  
 0734  
 0735 **Relying Party**  
 0736 The entity whose web application utilizes the Web Authentication  
 0737 API to register and authenticate users. See Registration and  
 0738 Authentication, respectively.  
 0739  
 0740 Note: While the term Relying Party is used in other contexts  
 0741 (e.g., X.509 and OAuth), an entity acting as a Relying Party in  
 0742 one context is not necessarily a Relying Party in other  
 0743 contexts.  
 0744  
 0745 **Relying Party Identifier**  
 0746 **RP ID**  
 0747 A valid domain string that identifies the Relying Party on whose  
 0748 behalf a given registration or authentication ceremony is being  
 0749 performed. A public key credential can only be used for  
 0750 authentication with the same entity (as identified by RP ID) it  
 0751 was registered with. By default, the RP ID for a WebAuthn  
 0752 operation is set to the caller's origin's effective domain. This  
 0753 default MAY be overridden by the caller, as long as the  
 0754 caller-specified RP ID value is a registrable domain suffix of  
 0755 or is equal to the caller's origin's effective domain. See also  
 0756 5.1.3 Create a new credential - PublicKeyCredential's  
 0757 [[Create]](origin, options, sameOriginWithAncestors) method and  
 0758 5.1.4 Use an existing credential to make an assertion -  
 0759 PublicKeyCredential's [[Get]](options) method.  
 0760  
 0761 Note: A Public key credential's scope is for a Relying Party's  
 0762 origin, with the following restrictions and relaxations:

0664  
 0665 + The scheme is always https (i.e., a restriction), and,  
 0666 + the host may be equal to the Relying Party's origin's  
 0667 effective domain, or it may be equal to a registrable domain  
 0668 suffix of the Relying Party's origin's effective domain (i.e.,  
 0669 an available relaxation), and,  
 0670 + all (TCP) ports on that host (i.e., a relaxation).  
 0671  
 0672 This is done in order to match the behavior of pervasively  
 0673 deployed ambient credentials (e.g., cookies, [RFC6265]). Please  
 0674 note that this is a greater relaxation of "same-origin"  
 0675 restrictions than what document.domain's setter provides.  
 0676  
**Test of User Presence**  
 0677 A test of user presence is a simple form of authorization  
 0678 gesture and technical process where a user interacts with an  
 0679 authenticator by (typically) simply touching it (other  
 0680 modalities may also exist), yielding a boolean result. Note that  
 0681 this does not constitute user verification because a user  
 0682 presence test, by definition, is not capable of biometric  
 0683 recognition, nor does it involve the presentation of a shared  
 0684 secret such as a password or PIN.  
 0685  
**User Consent**  
 0686 User consent means the user agrees with what they are being  
 0687 asked, i.e., it encompasses reading and understanding prompts.  
 0688 An authorization gesture is a ceremony component often employed  
 0689 to indicate user consent.  
 0690  
**User Handle**  
 0691 The user handle is specified by a Relying Party and is a unique  
 0692 identifier for a user account with that Relying Party. A user  
 0693 handle is an opaque byte sequence with a maximum size of 64  
 0694 bytes.  
 0695  
 0696 The user handle is not meant to be displayed to the user, but is  
 0697 used by the Relying Party to control the number of credentials -  
 0698 an authenticator will never contain more than one credential for  
 0699 a given Relying Party under the same user handle.  
 0700  
**User Verification**  
 0701 The technical process by which an authenticator locally  
 0702 authorizes the invocation of the authenticatorMakeCredential and  
 0703 authenticatorGetAssertion operations. User verification **may** be  
 0704 instigated through various authorization gesture modalities; for  
 0705 example, through a touch plus pin code, password entry, or  
 0706 biometric recognition (e.g., presenting a fingerprint)  
 0707 [ISOBiometricVocabulary]. The intent is to be able to  
 0708 distinguish individual users. Note that invocation of the  
 0709 authenticatorMakeCredential and authenticatorGetAssertion  
 0710 operations implies use of key material managed by the  
 0711 authenticator. Note that for security, user verification and use  
 0712 of credential private keys must occur within a single logical  
 0713 security boundary defining the authenticator.  
 0714  
**User Present**  
 0715 UP  
 0716 Upon successful completion of a user presence test, the user is  
 0717 said to be "present".  
 0718  
**User Verified**  
 0719 UV  
 0720 Upon successful completion of a user verification process, the  
 0721 user is said to be "verified".  
 0722  
**WebAuthn Client**  
 0723 Also referred to herein as simply a client. See also Conforming  
 0724 User Agent.  
 0725  
 0726  
 0727  
 0728  
 0729  
 0730  
 0731

0763  
 0764 + The scheme is always https (i.e., a restriction), and,  
 0765 + the host may be equal to the Relying Party's origin's  
 0766 effective domain, or it may be equal to a registrable domain  
 0767 suffix of the Relying Party's origin's effective domain (i.e.,  
 0768 an available relaxation), and,  
 0769 + all (TCP) ports on that host (i.e., a relaxation).  
 0770  
 0771 This is done in order to match the behavior of pervasively  
 0772 deployed ambient credentials (e.g., cookies, [RFC6265]). Please  
 0773 note that this is a greater relaxation of "same-origin"  
 0774 restrictions than what document.domain's setter provides.  
 0775  
**Test of User Presence**  
 0776 A test of user presence is a simple form of authorization  
 0777 gesture and technical process where a user interacts with an  
 0778 authenticator by (typically) simply touching it (other  
 0779 modalities may also exist), yielding a boolean result. Note that  
 0780 this does not constitute user verification because a user  
 0781 presence test, by definition, is not capable of biometric  
 0782 recognition, nor does it involve the presentation of a shared  
 0783 secret such as a password or PIN.  
 0784  
**User Consent**  
 0785 User consent means the user agrees with what they are being  
 0786 asked, i.e., it encompasses reading and understanding prompts.  
 0787 An authorization gesture is a ceremony component often employed  
 0788 to indicate user consent.  
 0789  
**User Handle**  
 0790 The user handle is specified by a Relying Party and is a unique  
 0791 identifier for a user account with that Relying Party. A user  
 0792 handle is an opaque byte sequence with a maximum size of 64  
 0793 bytes.  
 0794  
 0795 The user handle is not meant to be displayed to the user, but is  
 0796 used by the Relying Party to control the number of credentials -  
 0797 an authenticator will never contain more than one credential for  
 0798 a given Relying Party under the same user handle.  
 0799  
**User Verification**  
 0800 The technical process by which an authenticator locally  
 0801 authorizes the invocation of the authenticatorMakeCredential and  
 0802 authenticatorGetAssertion operations. User verification **MAY** be  
 0803 instigated through various authorization gesture modalities; for  
 0804 example, through a touch plus pin code, password entry, or  
 0805 biometric recognition (e.g., presenting a fingerprint)  
 0806 [ISOBiometricVocabulary]. The intent is to be able to  
 0807 distinguish individual users. Note that invocation of the  
 0808 authenticatorMakeCredential and authenticatorGetAssertion  
 0809 operations implies use of key material managed by the  
 0810 authenticator. Note that for security, user verification and use  
 0811 of credential private keys must occur within a single logical  
 0812 security boundary defining the authenticator.  
 0813  
**User Present**  
 0814 UP  
 0815 Upon successful completion of a user presence test, the user is  
 0816 said to be "present".  
 0817  
**User Verified**  
 0818 UV  
 0819 Upon successful completion of a user verification process, the  
 0820 user is said to be "verified".  
 0821  
**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  
 0827  
 0828  
 0829  
 0830  
 0831  
 0832

0732  
0733  
0734  
0735  
0736  
0737  
0738  
0739  
0740  
0741  
0742  
0743  
0744  
0745  
0746  
0747  
0748  
0749  
0750  
0751  
0752  
0753  
0754  
0755  
0756  
0757  
0758  
0759  
0760  
0761  
0762  
0763  
0764  
0765  
0766  
0767  
0768  
0769  
0770  
0771  
0772  
0773  
0774  
0775  
0776  
0777  
0778  
0779  
0780  
0781  
0782  
0783  
0784  
0785  
0786  
0787  
0788  
0789  
0790  
0791  
0792  
0793  
0794

## 5. Web Authentication API

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

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

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

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

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

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

### 5.1. PublicKeyCredential Interface

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

0833  
0834  
0835  
0836  
0837  
0838  
0839  
0840  
0841  
0842  
0843  
0844  
0845  
0846  
0847  
0848  
0849  
0850  
0851  
0852  
0853  
0854  
0855  
0856  
0857  
0858  
0859  
0860  
0861  
0862  
0863  
0864  
0865  
0866  
0867  
0868  
0869  
0870  
0871  
0872  
0873  
0874  
0875  
0876  
0877  
0878  
0879  
0880  
0881  
0882  
0883  
0884  
0885  
0886  
0887  
0888  
0889  
0890  
0891  
0900  
0901  
0902

embodies the implementation of the `[[Create]](origin, options, sameOriginWithAncestors)` and `[[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors)` internal methods. It is responsible for both marshalling the inputs for the underlying authenticator operations, and for returning the results of the latter operations to the Web Authentication API's callers.

## 5. Web Authentication API

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

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

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

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

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

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

### 5.1. PublicKeyCredential Interface

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

```

0795 interface PublicKeyCredential : Credential {
0796   [SameObject] readonly attribute ArrayBuffer rawId;
0797   [SameObject] readonly attribute AuthenticatorResponse response;
0798   AuthenticationExtensions getClientExtensionResults();
0799 };
0800
0801 id
0802   This attribute is inherited from Credential, though
0803   PublicKeyCredential overrides Credential's getter, instead
0804   returning the base64url encoding of the data contained in the
0805   object's [[identifier]] internal slot.
0806
0807 rawId
0808   This attribute returns the ArrayBuffer contained in the
0809   [[identifier]] internal slot.
0810
0811 response, of type AuthenticatorResponse, readonly
0812   This attribute contains the authenticator's response to the
0813   client's request to either create a public key credential, or
0814   generate an authentication assertion. If the PublicKeyCredential
0815   is created in response to create(), this attribute's value will
0816   be an AuthenticatorAttestationResponse, otherwise, the
0817   PublicKeyCredential was created in response to get(), and this
0818   attribute's value will be an AuthenticatorAssertionResponse.
0819
0820 getClientExtensionResults()
0821   This operation returns the value of [[clientExtensionsResults]],
0822   which is a map containing extension identifier -> client
0823   extension output entries produced by the extension's client
0824   extension processing.
0825
0826 [[type]]
0827   The PublicKeyCredential interface object's [[type]] internal
0828   slot's value is the string "public-key".
0829
0830 Note: This is reflected via the type attribute getter inherited
0831 from Credential.
0832
0833 [[discovery]]
0834   The PublicKeyCredential interface object's [[discovery]]
0835   internal slot's value is "remote".
0836
0837 [[identifier]]
0838   This internal slot contains an identifier for the credential,
0839   chosen by the platform with help from the authenticator. This
0840   identifier is used to look up credentials for use, and is
0841   therefore expected to be globally unique with high probability
0842   across all credentials of the same type, across all
0843   authenticators. This API does not constrain the format or length
0844   of this identifier, except that it must be sufficient for the
0845   platform to uniquely select a key. For example, an authenticator
0846   without on-board storage may create identifiers containing a
0847   credential private key wrapped with a symmetric key that is
0848   burned into the authenticator.
0849
0850 [[clientExtensionsResults]]
0851   This internal slot contains the results of processing client
0852   extensions requested by the Relying Party upon the Relying
0853   Party's invocation of either navigator.credentials.create() or
0854   navigator.credentials.get().
0855
0856 PublicKeyCredential's interface object inherits Credential's
0857 implementation of [[CollectFromCredentialStore]](origin, options,
0858 sameOriginWithAncestors), and defines its own implementation of
0859 [[Create]](origin, options, sameOriginWithAncestors),
0860 [[DiscoverFromExternalSource]](origin, options,
0861 sameOriginWithAncestors), and [[Store]](credential,
0862 sameOriginWithAncestors).
0863

```

```

0903 interface PublicKeyCredential : Credential {
0904   [SameObject] readonly attribute ArrayBuffer rawId;
0905   [SameObject] readonly attribute AuthenticatorResponse response;
0906   AuthenticationExtensionsClientOutputs getClientExtensionResults();
0907 };
0908
0909 id
0910   This attribute is inherited from Credential, though
0911   PublicKeyCredential overrides Credential's getter, instead
0912   returning the base64url encoding of the data contained in the
0913   object's [[identifier]] internal slot.
0914
0915 rawId
0916   This attribute returns the ArrayBuffer contained in the
0917   [[identifier]] internal slot.
0918
0919 response, of type AuthenticatorResponse, readonly
0920   This attribute contains the authenticator's response to the
0921   client's request to either create a public key credential, or
0922   generate an authentication assertion. If the PublicKeyCredential
0923   is created in response to create(), this attribute's value will
0924   be an AuthenticatorAttestationResponse, otherwise, the
0925   PublicKeyCredential was created in response to get(), and this
0926   attribute's value will be an AuthenticatorAssertionResponse.
0927
0928 getClientExtensionResults()
0929   This operation returns the value of [[clientExtensionsResults]],
0930   which is a map containing extension identifier -> client
0931   extension output entries produced by the extension's client
0932   extension processing.
0933
0934 [[type]]
0935   The PublicKeyCredential interface object's [[type]] internal
0936   slot's value is the string "public-key".
0937
0938 Note: This is reflected via the type attribute getter inherited
0939 from Credential.
0940
0941 [[discovery]]
0942   The PublicKeyCredential interface object's [[discovery]]
0943   internal slot's value is "remote".
0944
0945 [[identifier]]
0946   This internal slot contains the credential ID, chosen by the
0947   platform with help from the authenticator. The credential ID is
0948   used to look up credentials for use, and is therefore expected
0949   to be globally unique with high probability across all
0950   credentials of the same type, across all authenticators.
0951
0952 Note: This API does not constrain the format or length of this
0953 identifier, except that it MUST be sufficient for the platform
0954 to uniquely select a key. For example, an authenticator without
0955 on-board storage may create identifiers containing a credential
0956 private key wrapped with a symmetric key that is burned into the
0957 authenticator.
0958
0959 [[clientExtensionsResults]]
0960   This internal slot contains the results of processing client
0961   extensions requested by the Relying Party upon the Relying
0962   Party's invocation of either navigator.credentials.create() or
0963   navigator.credentials.get().
0964
0965 PublicKeyCredential's interface object inherits Credential's
0966 implementation of [[CollectFromCredentialStore]](origin, options,
0967 sameOriginWithAncestors), and defines its own implementation of
0968 [[Create]](origin, options, sameOriginWithAncestors),
0969 [[DiscoverFromExternalSource]](origin, options,
0970 sameOriginWithAncestors), and [[Store]](credential,
0971 sameOriginWithAncestors).
0972

```

0864 5.1.1. CredentialCreationOptions Extension  
0865  
0866 To support registration via navigator.credentials.create(), this  
0867 document extends the CredentialCreationOptions dictionary as follows:  
0868 partial dictionary CredentialCreationOptions {  
0869 MakePublicKeyCredentialOptions publicKey;  
0870 };  
0871  
0872 5.1.2. CredentialRequestOptions Extension  
0873  
0874 To support obtaining assertions via navigator.credentials.get(), this  
0875 document extends the CredentialRequestOptions dictionary as follows:  
0876 partial dictionary CredentialRequestOptions {  
0877 PublicKeyCredentialRequestOptions publicKey;  
0878 };  
0879  
0880 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin,  
0881 options, sameOriginWithAncestors) method  
0882  
0883 PublicKeyCredential's interface object's implementation of the  
0884  
0885 [[Create]](origin, options, sameOriginWithAncestors) internal method  
0886 [CREDENTIAL-MANAGEMENT-1] allows Relying Party scripts to call  
0887 navigator.credentials.create() to request the creation of a new public  
0888 key credential source, bound to an authenticator. This  
0889 navigator.credentials.create() operation can be aborted by leveraging  
0890 the AbortController; see DOM 3.3 Using AbortController and AbortSignal  
0891 objects in APIs for detailed instructions.  
0892  
0893 This internal method accepts three arguments:  
0894  
0895 origin  
0896 This argument is the relevant settings object's origin, as  
0897 determined by the calling create() implementation.  
0898  
0899 options  
0900 This argument is a CredentialCreationOptions object whose  
0901 options.publicKey member contains a  
0902 MakePublicKeyCredentialOptions object specifying the desired  
0903 attributes of the to-be-created public key credential.  
0904  
0905 sameOriginWithAncestors  
0906 This argument is a boolean which is true if and only if the  
0907 caller's environment settings object is same-origin with its  
0908 ancestors.  
0909  
0910 Note: This algorithm is synchronous: the Promise resolution/rejection  
0911 is handled by navigator.credentials.create().  
0912  
0913 When this method is invoked, the user agent MUST execute the following  
0914 algorithm:  
0915 1. Assert: options.publicKey is present.  
0916 2. If sameOriginWithAncestors is false, return a "NotAllowedError"  
0917 DOMException.  
0918 Note: This "sameOriginWithAncestors" restriction aims to address  
0919 the concern raised in the Origin Confusion section of  
0920 [CREDENTIAL-MANAGEMENT-1], while allowing Relying Party script  
0921 access to Web Authentication functionality, e.g., when running in a  
0922 secure context framed document that is same-origin with its  
0923 ancestors. However, in the future, this specification (in  
0924 conjunction with [CREDENTIAL-MANAGEMENT-1]) may provide Relying  
0925 Parties with more fine-grained control--e.g., ranging from allowing  
0926 only top-level access to Web Authentication functionality, to  
0927 allowing cross-origin embedded cases--by leveraging  
0928 [Feature-Policy] once the latter specification becomes stably  
0929 implemented in user agents.  
0930 3. Let options be the value of options.publicKey.  
0931 4. If the timeout member of options is present, check if its value  
0932 lies within a reasonable range as defined by the platform and if  
0933 not, correct it to the closest value lying within that range. Set a

0973 5.1.1. CredentialCreationOptions Dictionary Extension  
0974  
0975 To support registration via navigator.credentials.create(), this  
0976 document extends the CredentialCreationOptions dictionary as follows:  
0977 partial dictionary CredentialCreationOptions {  
0978 PublicKeyCredentialCreationOptions publicKey;  
0979 };  
0980  
0981 5.1.2. CredentialRequestOptions Dictionary Extension  
0982  
0983 To support obtaining assertions via navigator.credentials.get(), this  
0984 document extends the CredentialRequestOptions dictionary as follows:  
0985 partial dictionary CredentialRequestOptions {  
0986 PublicKeyCredentialRequestOptions publicKey;  
0987 };  
0988  
0989 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin,  
0990 options, sameOriginWithAncestors) method  
0991  
0992 PublicKeyCredential's interface object's implementation of the  
0993  
0994 [[Create]](origin, options, sameOriginWithAncestors) internal method  
0995 [CREDENTIAL-MANAGEMENT-1] allows Relying Party scripts to call  
0996 navigator.credentials.create() to request the creation of a new public  
0997 key credential source, bound to an authenticator. This  
0998 navigator.credentials.create() operation can be aborted by leveraging  
0999 the AbortController; see DOM 3.3 Using AbortController and AbortSignal  
1000 objects in APIs for detailed instructions.  
1001  
1002 This internal method accepts three arguments:  
1003  
1004 origin  
1005 This argument is the relevant settings object's origin, as  
1006 determined by the calling create() implementation.  
1007  
1008 options  
1009 This argument is a CredentialCreationOptions object whose  
1010 options.publicKey member contains a  
1011 PublicKeyCredentialCreationOptions object specifying the desired  
1012 attributes of the to-be-created public key credential.  
1013  
1014 sameOriginWithAncestors  
1015 This argument is a boolean which is true if and only if the  
1016 caller's environment settings object is same-origin with its  
1017 ancestors.  
1018  
1019 Note: This algorithm is synchronous: the Promise resolution/rejection  
1020 is handled by navigator.credentials.create().  
1021  
1022 When this method is invoked, the user agent MUST execute the following  
1023 algorithm:  
1024 1. Assert: options.publicKey is present.  
1025 2. If sameOriginWithAncestors is false, return a "NotAllowedError"  
1026 DOMException.  
1027 Note: This "sameOriginWithAncestors" restriction aims to address  
1028 the concern raised in the Origin Confusion section of  
1029 [CREDENTIAL-MANAGEMENT-1], while allowing Relying Party script  
1030 access to Web Authentication functionality, e.g., when running in a  
1031 secure context framed document that is same-origin with its  
1032 ancestors. However, in the future, this specification (in  
1033 conjunction with [CREDENTIAL-MANAGEMENT-1]) may provide Relying  
1034 Parties with more fine-grained control--e.g., ranging from allowing  
1035 only top-level access to Web Authentication functionality, to  
1036 allowing cross-origin embedded cases--by leveraging  
1037 [Feature-Policy] once the latter specification becomes stably  
1038 implemented in user agents.  
1039 3. Let options be the value of options.publicKey.  
1040 4. If the timeout member of options is present, check if its value  
1041 lies within a reasonable range as defined by the platform and if  
1042 not, correct it to the closest value lying within that range. Set a

0934 timer lifetimeTimer to this adjusted value. If the timeout member  
 0935 of options is not present, then set lifetimeTimer to a  
 0936 platform-specific default.  
 0937 5. Let callerOrigin be origin. If callerOrigin is an opaque origin,  
 0938 return a DOMException whose name is "NotAllowedError", and  
 0939 terminate this algorithm.  
 0940 6. Let effectiveDomain be the callerOrigin's effective domain. If  
 0941 effective domain is not a valid domain, then return a DOMException  
 0942 whose name is "SecurityError" and terminate this algorithm.  
 0943 Note: An effective domain may resolve to a host, which can be  
 0944 represented in various manners, such as domain, ipv4 address, ipv6  
 0945 address, opaque host, or empty host. Only the domain format of host  
 0946 is allowed here.  
 0947 7. If options.rp.id  
 0948  
 0949 Is present  
 0950 If options.rp.id is not a registrable domain suffix of and  
 0951 is not equal to effectiveDomain, return a DOMException  
 0952 whose name is "SecurityError", and terminate this  
 0953 algorithm.  
 0954  
 0955 Is not present  
 0956 Set options.rp.id to effectiveDomain.  
 0957  
 0958 Note: options.rp.id represents the caller's RP ID. The RP ID  
 0959 defaults to being the caller's origin's effective domain unless the  
 0960 caller has explicitly set options.rp.id when calling create().  
 0961 8. Let credTypesAndPubKeyAlgs be a new list whose items are pairs of  
 0962 PublicKeyCredentialType and a COSEAlgorithmIdentifier.  
 0963 9. For each current of options.pubKeyCredParams:  
 0964 1. If current.type does not contain a PublicKeyCredentialType  
 0965 supported by this implementation, then continue.  
 0966 2. Let alg be current.alg.  
 0967 3. Append the pair of current.type and alg to  
 0968 credTypesAndPubKeyAlgs.  
 0969 10. If credTypesAndPubKeyAlgs is empty and options.pubKeyCredParams is  
 0970 not empty, return a DOMException whose name is "NotSupportedError",  
 0971 and terminate this algorithm.  
 0972 11. Let clientExtensions be a new map and let authenticatorExtensions  
 0973 be a new map.  
 0974 12. If the extensions member of options is present, then for each  
 0975 extensionId -> clientExtensionInput of options.extensions:  
 0976 1. If extensionId is not supported by this client platform or is  
 0977 not a registration extension, then continue.  
 0978 2. Set clientExtensions[extensionId] to clientExtensionInput.  
 0979 3. If extensionId is not an authenticator extension, then  
 0980 continue.  
 0981 4. Let authenticatorExtensionInput be the (CBOR) result of  
 0982 running extensionId's client extension processing algorithm on  
 0983 clientExtensionInput. If the algorithm returned an error,  
 0984 continue.  
 0985 5. Set authenticatorExtensions[extensionId] to the base64url  
 0986 encoding of authenticatorExtensionInput.  
 0987 13. Let collectedClientData be a new CollectedClientData instance whose  
 0988 fields are:  
 0989  
 0990 type  
 0991 The string "webauthn.create".  
 0992  
 0993 challenge  
 0994 The base64url encoding of options.challenge.  
 0995  
 0996 origin  
 0997 The serialization of callerOrigin.  
 0998  
 0999 hashAlgorithm  
 1000 The recognized algorithm name of the hash algorithm  
 1001 selected by the client for generating the hash of the  
 1002 serialized client data.  
 1003

1043 timer lifetimeTimer to this adjusted value. If the timeout member  
 1044 of options is not present, then set lifetimeTimer to a  
 1045 platform-specific default.  
 1046 5. Let callerOrigin be origin. If callerOrigin is an opaque origin,  
 1047 return a DOMException whose name is "NotAllowedError", and  
 1048 terminate this algorithm.  
 1049 6. Let effectiveDomain be the callerOrigin's effective domain. If  
 1050 effective domain is not a valid domain, then return a DOMException  
 1051 whose name is "SecurityError" and terminate this algorithm.  
 1052 Note: An effective domain may resolve to a host, which can be  
 1053 represented in various manners, such as domain, ipv4 address, ipv6  
 1054 address, opaque host, or empty host. Only the domain format of host  
 1055 is allowed here.  
 1056 7. If options.rp.id  
 1057  
 1058 Is present  
 1059 If options.rp.id is not a registrable domain suffix of and  
 1060 is not equal to effectiveDomain, return a DOMException  
 1061 whose name is "SecurityError", and terminate this  
 1062 algorithm.  
 1063  
 1064 Is not present  
 1065 Set options.rp.id to effectiveDomain.  
 1066  
 1067 Note: options.rp.id represents the caller's RP ID. The RP ID  
 1068 defaults to being the caller's origin's effective domain unless the  
 1069 caller has explicitly set options.rp.id when calling create().  
 1070 8. Let credTypesAndPubKeyAlgs be a new list whose items are pairs of  
 1071 PublicKeyCredentialType and a COSEAlgorithmIdentifier.  
 1072 9. For each current of options.pubKeyCredParams:  
 1073 1. If current.type does not contain a PublicKeyCredentialType  
 1074 supported by this implementation, then continue.  
 1075 2. Let alg be current.alg.  
 1076 3. Append the pair of current.type and alg to  
 1077 credTypesAndPubKeyAlgs.  
 1078 10. If credTypesAndPubKeyAlgs is empty and options.pubKeyCredParams is  
 1079 not empty, return a DOMException whose name is "NotSupportedError",  
 1080 and terminate this algorithm.  
 1081 11. Let clientExtensions be a new map and let authenticatorExtensions  
 1082 be a new map.  
 1083 12. If the extensions member of options is present, then for each  
 1084 extensionId -> clientExtensionInput of options.extensions:  
 1085 1. If extensionId is not supported by this client platform or is  
 1086 not a registration extension, then continue.  
 1087 2. Set clientExtensions[extensionId] to clientExtensionInput.  
 1088 3. If extensionId is not an authenticator extension, then  
 1089 continue.  
 1090 4. Let authenticatorExtensionInput be the (CBOR) result of  
 1091 running extensionId's client extension processing algorithm on  
 1092 clientExtensionInput. If the algorithm returned an error,  
 1093 continue.  
 1094 5. Set authenticatorExtensions[extensionId] to the base64url  
 1095 encoding of authenticatorExtensionInput.  
 1096 13. Let collectedClientData be a new CollectedClientData instance whose  
 1097 fields are:  
 1098  
 1099 type  
 1100 The string "webauthn.create".  
 1101  
 1102 challenge  
 1103 The base64url encoding of options.challenge.  
 1104  
 1105 origin  
 1106 The serialization of callerOrigin.  
 1107  
 1108 tokenBinding  
 1109 The status of Token Binding between the client and the  
 1110 callerOrigin, as well as the Token Binding ID associated  
 1111 with callerOrigin, if one is available.

1004 tokenBindingId  
 1005 The Token Binding ID associated with callerOrigin, if one  
 1006 is available.  
 1007  
 1008 clientExtensions  
 1009 clientExtensions  
 1010  
 1011 authenticatorExtensions  
 1012 authenticatorExtensions  
 1013  
 1014 14. Let clientDataJSON be the JSON-serialized client data constructed  
 1015 from collectedClientData.  
 1016 15. Let clientDataHash be the hash of the serialized client data  
 1017 represented by clientDataJSON.  
 1018 16. If the options.signal is present and its aborted flag is set to  
 1019 true, return a DOMException whose name is "AbortError" and  
 1020 terminate this algorithm.  
 1021 17. Start lifetimeTimer.  
 1022 18. Let issuedRequests be a new ordered set.  
 1023 19. For each authenticator that becomes available on this platform  
 1024 during the lifetime of lifetimeTimer, do the following:  
 1025 The definitions of "lifetime of" and "becomes available" are  
 1026 intended to represent how devices are hotplugged into (USB) or  
 1027 discovered by (NFC) browsers, and are under-specified. Resolving  
 1028 this with good definitions or some other means will be addressed by  
 1029 resolving Issue #613.  
 1030 1. If options.authenticatorSelection is present:  
 1031 1. If options.authenticatorSelection.authenticatorAttachment  
 1032 is present and its value is not equal to authenticator's  
 1033 attachment modality, continue.  
 1034 2. If options.authenticatorSelection.requireResidentKey is  
 1035 set to true and the authenticator is not capable of  
 1036 storing a Client-Side-Resident Credential Private Key,  
 1037 continue.  
 1038 3. If options.authenticatorSelection.userVerification is set  
 1039 to required and the authenticator is not capable of  
 1040 performing user verification, continue.  
 1041 2. Let userVerification be the effective user verification  
 1042 requirement for credential creation, a Boolean value, as  
 1043 follows. If options.authenticatorSelection.userVerification  
 1044 is set to required  
 1045 Let userVerification be true.  
 1046  
 1047 is set to preferred  
 1048 If the authenticator  
 1049  
 1050 is capable of user verification  
 1051 Let userVerification be true.  
 1052  
 1053 is not capable of user verification  
 1054 Let userVerification be false.  
 1055  
 1056 is set to discouraged  
 1057 Let userVerification be false.  
 1058  
 1059 3. Let userPresence be a Boolean value set to the inverse of  
 1060 userVerification.  
 1061 4. Let excludeCredentialDescriptorList be a new list.  
 1062 5. For each credential descriptor C in  
 1063 options.excludeCredentials:  
 1064 1. If C.transports is not empty, and authenticator is  
 1065 connected over a transport not mentioned in C.transports,  
 1066 the client MAY continue.  
 1067 2. Otherwise, Append C to excludeCredentialDescriptorList.  
 1068 6. Invoke the authenticatorMakeCredential operation on  
 1069 authenticator with clientDataHash, options.rp, options.user,  
 1070 options.authenticatorSelection.requireResidentKey,  
 1071 userPresence, userVerification, credTypesAndPubKeyAlgs,  
 1072 excludeCredentialDescriptorList, and authenticatorExtensions  
 1073

1112  
 1113 14. Let clientDataJSON be the JSON-serialized client data constructed  
 1114 from collectedClientData.  
 1115 15. Let clientDataHash be the hash of the serialized client data  
 1116 represented by clientDataJSON.  
 1117 16. If the options.signal is present and its aborted flag is set to  
 1118 true, return a DOMException whose name is "AbortError" and  
 1119 terminate this algorithm.  
 1120 17. Start lifetimeTimer.  
 1121 18. Let issuedRequests be a new ordered set.  
 1122 19. For each authenticator that becomes available on this platform  
 1123 during the lifetime of lifetimeTimer, do the following:  
 1124 The definitions of "lifetime of" and "becomes available" are  
 1125 intended to represent how devices are hot-plugged into (USB) or  
 1126 discovered by (NFC) browsers, and are underspecified. Resolving  
 1127 this with good definitions or some other means will be addressed by  
 1128 resolving Issue #613.  
 1129 1. If options.authenticatorSelection is present:  
 1130 1. If options.authenticatorSelection.authenticatorAttachment  
 1131 is present and its value is not equal to authenticator's  
 1132 attachment modality, continue.  
 1133 2. If options.authenticatorSelection.requireResidentKey is  
 1134 set to true and the authenticator is not capable of  
 1135 storing a Client-Side-Resident Credential Private Key,  
 1136 continue.  
 1137 3. If options.authenticatorSelection.userVerification is set  
 1138 to required and the authenticator is not capable of  
 1139 performing user verification, continue.  
 1140 2. Let userVerification be the effective user verification  
 1141 requirement for credential creation, a Boolean value, as  
 1142 follows. If options.authenticatorSelection.userVerification  
 1143 is set to required  
 1144 Let userVerification be true.  
 1145  
 1146 is set to preferred  
 1147 If the authenticator  
 1148  
 1149 is capable of user verification  
 1150 Let userVerification be true.  
 1151  
 1152 is not capable of user verification  
 1153 Let userVerification be false.  
 1154  
 1155 is set to discouraged  
 1156 Let userVerification be false.  
 1157  
 1158 3. Let userPresence be a Boolean value set to the inverse of  
 1159 userVerification.  
 1160 4. Let excludeCredentialDescriptorList be a new list.  
 1161 5. For each credential descriptor C in  
 1162 options.excludeCredentials:  
 1163 1. If C.transports is not empty, and authenticator is  
 1164 connected over a transport not mentioned in C.transports,  
 1165 the client MAY continue.  
 1166 2. Otherwise, Append C to excludeCredentialDescriptorList.  
 1167 6. Invoke the authenticatorMakeCredential operation on  
 1168 authenticator with clientDataHash, options.rp, options.user,  
 1169 options.authenticatorSelection.requireResidentKey,  
 1170 userPresence, userVerification, credTypesAndPubKeyAlgs,  
 1171 excludeCredentialDescriptorList, and authenticatorExtensions  
 1172

1074 as parameters.  
1075 7. Append authenticator to issuedRequests.  
1076 20. While **issuedRequests** is not **empty**, perform the following actions  
1077 depending upon **lifetimeTimer** and responses from the authenticators:  
1078  
1079 If **lifetimeTimer** expires,  
1080 For each authenticator in **issuedRequests** invoke the  
1081 **authenticatorCancel** operation on authenticator and remove  
1082 authenticator from **issuedRequests**.  
1083  
1084 If the **options.signal** is present and its **aborted** flag is set to  
1085 **true**,  
1086 For each authenticator in **issuedRequests** invoke the  
1087 **authenticatorCancel** operation on authenticator and remove  
1088 authenticator from **issuedRequests**. Then return a  
1089 **DOMException** whose name is "AbortError" and terminate this  
1090 algorithm.  
1091  
1092 If any authenticator returns a status indicating that the user  
1093 cancelled the operation,  
1094  
1095 1. Remove authenticator from **issuedRequests**.  
1096 2. For each remaining authenticator in **issuedRequests** invoke  
1097 the **authenticatorCancel** operation on authenticator and  
1098 remove it from **issuedRequests**.  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108 Remove authenticator from **issuedRequests**.  
1109  
1110  
1111  
1112  
1113  
1114  
1115

1173 as parameters.  
1174 7. Append authenticator to issuedRequests.  
1175 20. While **lifetimeTimer** has not **expired**, perform the following actions  
1176 depending upon **lifetimeTimer** and responses from the authenticators:  
1177  
1178 If **lifetimeTimer** expires,  
1179 For each authenticator in **issuedRequests** invoke the  
1180 **authenticatorCancel** operation on authenticator and remove  
1181 authenticator from **issuedRequests**.  
1182  
1183 If the **options.signal** is present and its **aborted** flag is set to  
1184 **true**,  
1185 For each authenticator in **issuedRequests** invoke the  
1186 **authenticatorCancel** operation on authenticator and remove  
1187 authenticator from **issuedRequests**. Then return a  
1188 **DOMException** whose name is "AbortError" and terminate this  
1189 algorithm.  
1190  
1191 If any authenticator returns a status indicating that the user  
1192 cancelled the operation,  
1193  
1194 1. Remove authenticator from **issuedRequests**.  
1195 2. For each remaining authenticator in **issuedRequests** invoke  
1196 the **authenticatorCancel** operation on authenticator and  
1197 remove it from **issuedRequests**.  
1198 **Note: Authenticators may return an indication of "the**  
1199 **user cancelled the entire operation". How a user agent**  
1200 **manifests this state to users is unspecified.**  
1201  
1202 If any authenticator returns an error status **equivalent to**  
1203 **"InvalidStateError"**,  
1204  
1205 1. Remove authenticator from **issuedRequests**.  
1206 2. For each remaining authenticator in **issuedRequests** invoke  
1207 the **authenticatorCancel** operation on authenticator and  
1208 remove it from **issuedRequests**.  
1209 3. Return a **DOMException** whose name is "InvalidStateError"  
1210 and terminate this algorithm.  
1211  
1212 **Note: This error status is handled separately because the**  
1213 **authenticator returns it only if**  
1214 **excludeCredentialDescriptorList** identifies a credential  
1215 bound to the authenticator and the user has consented to  
1216 the operation. Given this explicit consent, it is  
1217 acceptable for this case to be distinguishable to the  
1218 Relying Party.  
1219  
1220 If any authenticator returns an error status not equivalent to  
1221 **"InvalidStateError"**,  
1222 Remove authenticator from **issuedRequests**.  
1223  
1224 **Note: This case does not imply user consent for the**  
1225 **operation, so details about the error must be hidden from**  
1226 **the Relying Party in order to prevent leak of potentially**  
1227 **identifying information. See 14.2 Registration Ceremony**  
1228 **Privacy for details.**  
1229  
1230 If any authenticator indicates success,  
1231  
1232 1. Remove authenticator from **issuedRequests**.  
1233 2. Let **credentialCreationData** be a struct whose items are:  
1234  
1235 **attestationObjectResult**  
1236 whose value is the bytes returned from the  
1237 successful **authenticatorMakeCredential**  
1238 operation.  
1239  
1240 **Note: this value is attObj, as defined in**  
1241 **6.3.4 Generating an Attestation Object.**  
1242

1116 clientDataJSONResult  
 1117 whose value is the bytes of clientDataJSON.  
 1118  
 1119 attestationConveyancePreferenceOption  
 1120 whose value is the value of  
 1121 options.attestation.  
 1122  
 1123 clientExtensionResults  
 1124 whose value is an **AuthenticationExtensions**  
 1125 **object containing extension identifier ->**  
 1126 **client extension output entries. The entries**  
 1127 **are created by running each extension's client**  
 1128  
 1129 extension processing algorithm to create the  
 1130 client extension outputs, for each client  
 1131 extension in clientDataJSON.clientExtensions.  
 1132  
 1133 3. Let constructCredentialAlg be an algorithm that takes a  
 1134 global object global, and whose steps are:  
 1135 1. Let attestationObject be a new **ArrayBuffer**, created  
 1136 **using global's %ArrayBuffer%, containing the bytes**  
 1137 **of credentialCreationData.attestationObjectResult's**  
 1138 **value.**  
 1139 2. If  
 1140 credentialCreationData.attestationConveyancePreferen  
 1141 ceOption's value is  
 1142 "none"  
 1143 Replace potentially uniquely identifying  
 1144 information (such as **AAGUID and**  
 1145 **attestation certificates) in the**  
 1146 **attested credential data and attestation**  
 1147 **statement, respectively, with blinded**  
 1148 **versions of the same data.**  
 1149  
 1150 **need to define "blinding". See also**  
 1151 **#462.**  
 1152 **<[https://github.com/w3c/webauthn/issues/](https://github.com/w3c/webauthn/issues/694)**  
 1153 **694>**

1154 "indirect"  
 1155 The client MAY replace the AAGUID and  
 1156 attestation statement with a more  
 1157 privacy-friendly and/or more easily  
 1158 verifiable version of the same data (for  
 1159 example, by employing a **Privacy CA**).  
 1160

1161 "direct"  
 1162 Convey the authenticator's AAGUID and  
 1163 attestation statement, unaltered, to the  
 1164 RP.  
 1165  
 1166 @balfanz wishes to add to the "direct"  
 1167

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

1168 case: If the authenticator violates the  
 1169 privacy requirements of the attestation  
 1170 type it is using, the client SHOULD  
 1171 terminate this algorithm with a  
 1172 "AttestationNotPrivateError".  
 1173

1174 3. Let id be  
 1175 attestationObject.authData.attestedCredentialData.cred  
 1176 edentialId.  
 1177 4. Let pubKeyCred be a new PublicKeyCredential object  
 1178 associated with global whose fields are:

1179     [[identifier]]  
 1180     id  
 1181  
 1182     response  
 1183     A new AuthenticatorAttestationResponse  
 1184     object associated with global whose  
 1185     fields are:

1186     clientDataJSON  
 1187     A new ArrayBuffer, created using  
 1188     global's %ArrayBuffer%, containing  
 1189     the bytes of  
 1190     credentialCreationData.clientDataJ  
 1191     SONResult.  
 1192  
 1193     attestationObject  
 1194     attestationObject  
 1195  
 1196     [[clientExtensionsResults]]  
 1197     A new ArrayBuffer, created using  
 1198     global's %ArrayBuffer%, containing the  
 1199     bytes of  
 1200     credentialCreationData.clientExtensionRe  
 1201     sults.  
 1202  
 1203     5. Return pubKeyCred.  
 1204  
 1205 4. For each remaining authenticator in issuedRequests invoke  
 1206 the authenticatorCancel operation on authenticator and  
 1207 remove it from issuedRequests.  
 1208 5. Return constructCredentialAlg and terminate this  
 1209 algorithm.  
 1210  
 1211 21. Return a DOMException whose name is "NotAllowedError".  
 1212

1213  
 1214 During the above process, the user agent SHOULD show some UI to the  
 1215 user to guide them in the process of selecting and authorizing an  
 1216 authenticator.  
 1217

1218 5.1.4. Use an existing credential to make an assertion -  
 1219 PublicKeyCredential's [[Get]](options) method  
 1220

1221 Relying Parties call navigator.credentials.get({publicKey:..., ...}) to  
 1222 discover and use an existing public key credential, with the user's  
 1223 consent. Relying Party script optionally specifies some criteria to  
 1224 indicate what credential sources are acceptable to it. The user agent  
 1225 and/or platform locates credential sources matching the specified  
 1226 criteria, and guides the user to pick one that the script will be  
 1227 allowed to use. The user may choose to decline the entire interaction  
 1228 even if a credential source is present, for example to maintain  
 1229 privacy. If the user picks a credential source, the user agent then  
 1230 uses 6.2.2 The authenticatorGetAssertion operation to sign a Relying  
 1231

1306 case: If the authenticator violates the  
 1307 privacy requirements of the attestation  
 1308 type it is using, the client SHOULD  
 1309 terminate this algorithm with an  
 1310 "AttestationNotPrivateError".  
 1311

1312 2. Let attestationObject be a new ArrayBuffer, created  
 1313 using global's %ArrayBuffer%, containing the bytes  
 1314 of credentialCreationData.attestationObjectResult's  
 1315 value.  
 1316 3. Let id be  
 1317 attestationObject.authData.attestedCredentialData.cred  
 1318 edentialId.  
 1319 4. Let pubKeyCred be a new PublicKeyCredential object  
 1320 associated with global whose fields are:

1321     [[identifier]]  
 1322     id  
 1323  
 1324     response  
 1325     A new AuthenticatorAttestationResponse  
 1326     object associated with global whose  
 1327     fields are:

1328     clientDataJSON  
 1329     A new ArrayBuffer, created using  
 1330     global's %ArrayBuffer%, containing  
 1331     the bytes of  
 1332     credentialCreationData.clientDataJ  
 1333     SONResult.  
 1334  
 1335     attestationObject  
 1336     attestationObject  
 1337  
 1338     [[clientExtensionsResults]]  
 1339     A new ArrayBuffer, created using  
 1340     global's %ArrayBuffer%, containing the  
 1341     bytes of  
 1342     credentialCreationData.clientExtensionRe  
 1343     sults.  
 1344  
 1345     5. Return pubKeyCred.  
 1346  
 1347 4. For each remaining authenticator in issuedRequests invoke  
 1348 the authenticatorCancel operation on authenticator and  
 1349 remove it from issuedRequests.  
 1350 5. Return constructCredentialAlg and terminate this  
 1351 algorithm.  
 1352  
 1353 21. Return a DOMException whose name is "NotAllowedError".  
 1354

1355 **prevent information leak that could identify the user without**  
 1356 **consent, this step MUST NOT be executed before lifetimeTimer has**  
 1357 **expired. See 14.3 Authentication Ceremony Privacy for details.**  
 1358

1359 During the above process, the user agent SHOULD show some UI to the  
 1360 user to guide them in the process of selecting and authorizing an  
 1361 authenticator.  
 1362

1363 5.1.4. Use an existing credential to make an assertion -  
 1364 PublicKeyCredential's [[Get]](options) method  
 1365

1366 Relying Parties call navigator.credentials.get({publicKey:..., ...}) to  
 1367 discover and use an existing public key credential, with the user's  
 1368 consent. Relying Party script optionally specifies some criteria to  
 1369 indicate what credential sources are acceptable to it. The user agent  
 1370 and/or platform locates credential sources matching the specified  
 1371 criteria, and guides the user to pick one that the script will be  
 1372 allowed to use. The user may choose to decline the entire interaction  
 1373 even if a credential source is present, for example to maintain  
 1374 privacy. If the user picks a credential source, the user agent then  
 1375 uses 6.2.3 The authenticatorGetAssertion operation to sign a Relying

1231 Party-provided challenge and other collected data into an assertion,  
 1232 which is used as a credential.  
 1233

1234 The get() implementation [CREDENTIAL-MANAGEMENT-1] calls  
 1235 PublicKeyCredential.[[CollectFromCredentialStore]]() to collect any  
 1236 credentials that should be available without user mediation (roughly,  
 1237 this specification's authorization gesture), and if it does not find  
 1238 exactly one of those, it then calls  
 1239 PublicKeyCredential.[[DiscoverFromExternalSource]]() to have the user  
 1240 select a credential source.  
 1241

1242 Since this specification requires an authorization gesture to create  
 1243 any credentials, the  
 1244 PublicKeyCredential.[[CollectFromCredentialStore]](origin, options,  
 1245 sameOriginWithAncestors) internal method inherits the default behavior  
 1246 of Credential.[[CollectFromCredentialStore]](), of returning an empty  
 1247 set.  
 1248

1249 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin,  
 1250 options, sameOriginWithAncestors) method  
 1251

1252 This internal method accepts three arguments:  
 1253

1254 origin  
 1255 This argument is the relevant settings object's origin, as  
 1256 determined by the calling get() implementation, i.e.,  
 1257 CredentialsContainer's Request a Credential abstract operation.  
 1258

1259 options  
 1260 This argument is a CredentialRequestOptions object whose  
 1261 options.publicKey member contains a  
 1262 PublicKeyCredentialRequestOptions object specifying the desired  
 1263 attributes of the public key credential to discover.  
 1264

1265 sameOriginWithAncestors  
 1266 This argument is a boolean which is true if and only if the  
 1267 caller's environment settings object is same-origin with its  
 1268 ancestors.  
 1269

1270 Note: This algorithm is synchronous: the Promise resolution/rejection  
 1271 is handled by navigator.credentials.get().  
 1272

1273 When this method is invoked, the user agent MUST execute the following  
 1274 algorithm:  
 1275 1. Assert: options.publicKey is present.  
 1276 2. If sameOriginWithAncestors is false, return a "NotAllowedError"  
 1277 DOMException.  
 1278 Note: This "sameOriginWithAncestors" restriction aims to address  
 1279 the concern raised in the Origin Confusion section of  
 1280 [CREDENTIAL-MANAGEMENT-1], while allowing Relying Party script  
 1281 access to Web Authentication functionality, e.g., when running in a  
 1282 secure context framed document that is same-origin with its  
 1283 ancestors. However, in the future, this specification (in  
 1284 conjunction with [CREDENTIAL-MANAGEMENT-1]) may provide Relying  
 1285 Parties with more fine-grained control--e.g., ranging from allowing  
 1286 only top-level access to Web Authentication functionality, to  
 1287 allowing cross-origin embedded cases--by leveraging  
 1288 [Feature-Policy] once the latter specification becomes stably  
 1289 implemented in user agents.  
 1290 3. Let options be the value of options.publicKey.  
 1291 4. If the timeout member of options is present, check if its value  
 1292 lies within a reasonable range as defined by the platform and if  
 1293 not, correct it to the closest value lying within that range. Set a  
 1294 timer lifetimeTimer to this adjusted value. If the timeout member  
 1295 of options is not present, then set lifetimeTimer to a  
 1296 platform-specific default.  
 1297 5. Let callerOrigin be origin. If callerOrigin is an opaque origin,  
 1298 return a DOMException whose name is "NotAllowedError", and  
 1299 terminate this algorithm.  
 1300 6. Let effectiveDomain be the callerOrigin's effective domain. If

1376 Party-provided challenge and other collected data into an assertion,  
 1377 which is used as a credential.  
 1378

1379 The get() implementation [CREDENTIAL-MANAGEMENT-1] calls  
 1380 PublicKeyCredential.[[CollectFromCredentialStore]]() to collect any  
 1381 credentials that should be available without user mediation (roughly,  
 1382 this specification's authorization gesture), and if it does not find  
 1383 exactly one of those, it then calls  
 1384 PublicKeyCredential.[[DiscoverFromExternalSource]]() to have the user  
 1385 select a credential source.  
 1386

1387 Since this specification requires an authorization gesture to create  
 1388 any credentials, the  
 1389 PublicKeyCredential.[[CollectFromCredentialStore]](origin, options,  
 1390 sameOriginWithAncestors) internal method inherits the default behavior  
 1391 of Credential.[[CollectFromCredentialStore]](), of returning an empty  
 1392 set.  
 1393

1394 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin,  
 1395 options, sameOriginWithAncestors) method  
 1396

1397 This internal method accepts three arguments:  
 1398

1399 origin  
 1400 This argument is the relevant settings object's origin, as  
 1401 determined by the calling get() implementation, i.e.,  
 1402 CredentialsContainer's Request a Credential abstract operation.  
 1403

1404 options  
 1405 This argument is a CredentialRequestOptions object whose  
 1406 options.publicKey member contains a  
 1407 PublicKeyCredentialRequestOptions object specifying the desired  
 1408 attributes of the public key credential to discover.  
 1409

1410 sameOriginWithAncestors  
 1411 This argument is a boolean which is true if and only if the  
 1412 caller's environment settings object is same-origin with its  
 1413 ancestors.  
 1414

1415 Note: This algorithm is synchronous: the Promise resolution/rejection  
 1416 is handled by navigator.credentials.get().  
 1417

1418 When this method is invoked, the user agent MUST execute the following  
 1419 algorithm:  
 1420 1. Assert: options.publicKey is present.  
 1421 2. If sameOriginWithAncestors is false, return a "NotAllowedError"  
 1422 DOMException.  
 1423 Note: This "sameOriginWithAncestors" restriction aims to address  
 1424 the concern raised in the Origin Confusion section of  
 1425 [CREDENTIAL-MANAGEMENT-1], while allowing Relying Party script  
 1426 access to Web Authentication functionality, e.g., when running in a  
 1427 secure context framed document that is same-origin with its  
 1428 ancestors. However, in the future, this specification (in  
 1429 conjunction with [CREDENTIAL-MANAGEMENT-1]) may provide Relying  
 1430 Parties with more fine-grained control--e.g., ranging from allowing  
 1431 only top-level access to Web Authentication functionality, to  
 1432 allowing cross-origin embedded cases--by leveraging  
 1433 [Feature-Policy] once the latter specification becomes stably  
 1434 implemented in user agents.  
 1435 3. Let options be the value of options.publicKey.  
 1436 4. If the timeout member of options is present, check if its value  
 1437 lies within a reasonable range as defined by the platform and if  
 1438 not, correct it to the closest value lying within that range. Set a  
 1439 timer lifetimeTimer to this adjusted value. If the timeout member  
 1440 of options is not present, then set lifetimeTimer to a  
 1441 platform-specific default.  
 1442 5. Let callerOrigin be origin. If callerOrigin is an opaque origin,  
 1443 return a DOMException whose name is "NotAllowedError", and  
 1444 terminate this algorithm.  
 1445 6. Let effectiveDomain be the callerOrigin's effective domain. If

1301 effective domain is not a valid domain, then return a DOMException  
 1302 whose name is "SecurityError" and terminate this algorithm.  
 1303 Note: An effective domain may resolve to a host, which can be  
 1304 represented in various manners, such as domain, ipv4 address, ipv6  
 1305 address, opaque host, or empty host. Only the domain format of host  
 1306 is allowed here.  
 1307 7. If options.rpld is not present, then set rpld to effectiveDomain.  
 1308 Otherwise:  
 1309 1. If options.rpld is not a registrable domain suffix of and is  
 1310 not equal to effectiveDomain, return a DOMException whose name  
 1311 is "SecurityError", and terminate this algorithm.  
 1312 2. Set rpld to options.rpld.  
 1313 Note: rpld represents the caller's RP ID. The RP ID defaults  
 1314 to being the caller's origin's effective domain unless the  
 1315 caller has explicitly set options.rpld when calling get().  
 1316 8. Let clientExtensions be a new map and let authenticatorExtensions  
 1317 be a new map.  
 1318 9. If the extensions member of options is present, then for each  
 1319 extensionId -> clientExtensionInput of options.extensions:  
 1320 1. If extensionId is not supported by this client platform or is  
 1321 not an authentication extension, then continue.  
 1322 2. Set clientExtensions[extensionId] to clientExtensionInput.  
 1323 3. If extensionId is not an authenticator extension, then  
 1324 continue.  
 1325 4. Let authenticatorExtensionInput be the (CBOR) result of  
 1326 running extensionId's client extension processing algorithm on  
 1327 clientExtensionInput. If the algorithm returned an error,  
 1328 continue.  
 1329 5. Set authenticatorExtensions[extensionId] to the base64url  
 1330 encoding of authenticatorExtensionInput.  
 1331 10. Let collectedClientData be a new CollectedClientData instance whose  
 1332 fields are:  
 1333  
 1334 type  
 1335 The string "webauthn.get".  
 1336  
 1337 challenge  
 1338 The base64url encoding of options.challenge  
 1339  
 1340 origin  
 1341 The serialization of callerOrigin.  
 1342  
 1343 hashAlgorithm  
 1344 The recognized algorithm name of the hash algorithm  
 1345 selected by the client for generating the hash of the  
 1346 serialized client data  
 1347  
 1348 tokenBindingId  
 1349 The Token Binding ID associated with callerOrigin, if one  
 1350 is available.  
 1351  
 1352 clientExtensions  
 1353 clientExtensions  
 1354  
 1355 authenticatorExtensions  
 1356 authenticatorExtensions  
 1357  
 1358 11. Let clientDataJSON be the JSON-serialized client data constructed  
 1359 from collectedClientData.  
 1360 12. Let clientDataHash be the hash of the serialized client data  
 1361 represented by clientDataJSON.  
 1362 13. If the options.signal is present and its aborted flag is set to  
 1363 true, return a DOMException whose name is "AbortError" and  
 1364 terminate this algorithm.  
 1365 14. Let issuedRequests be a new ordered set.  
 1366 15. Let authenticator be a platform-specific handle whose value  
 1367 identifies an authenticator.  
 1368 16. Start lifetimeTimer.  
 1369 17. For each authenticator that becomes available on this platform  
 1370 during the lifetime of lifetimeTimer, perform the following steps:

1446 effective domain is not a valid domain, then return a DOMException  
 1447 whose name is "SecurityError" and terminate this algorithm.  
 1448 Note: An effective domain may resolve to a host, which can be  
 1449 represented in various manners, such as domain, ipv4 address, ipv6  
 1450 address, opaque host, or empty host. Only the domain format of host  
 1451 is allowed here.  
 1452 7. If options.rpld is not present, then set rpld to effectiveDomain.  
 1453 Otherwise:  
 1454 1. If options.rpld is not a registrable domain suffix of and is  
 1455 not equal to effectiveDomain, return a DOMException whose name  
 1456 is "SecurityError", and terminate this algorithm.  
 1457 2. Set rpld to options.rpld.  
 1458 Note: rpld represents the caller's RP ID. The RP ID defaults  
 1459 to being the caller's origin's effective domain unless the  
 1460 caller has explicitly set options.rpld when calling get().  
 1461 8. Let clientExtensions be a new map and let authenticatorExtensions  
 1462 be a new map.  
 1463 9. If the extensions member of options is present, then for each  
 1464 extensionId -> clientExtensionInput of options.extensions:  
 1465 1. If extensionId is not supported by this client platform or is  
 1466 not an authentication extension, then continue.  
 1467 2. Set clientExtensions[extensionId] to clientExtensionInput.  
 1468 3. If extensionId is not an authenticator extension, then  
 1469 continue.  
 1470 4. Let authenticatorExtensionInput be the (CBOR) result of  
 1471 running extensionId's client extension processing algorithm on  
 1472 clientExtensionInput. If the algorithm returned an error,  
 1473 continue.  
 1474 5. Set authenticatorExtensions[extensionId] to the base64url  
 1475 encoding of authenticatorExtensionInput.  
 1476 10. Let collectedClientData be a new CollectedClientData instance whose  
 1477 fields are:  
 1478  
 1479 type  
 1480 The string "webauthn.get".  
 1481  
 1482 challenge  
 1483 The base64url encoding of options.challenge  
 1484  
 1485 origin  
 1486 The serialization of callerOrigin.  
 1487  
 1488 tokenBinding  
 1489 The status of Token Binding between the client and the  
 1490 callerOrigin, as well as the Token Binding ID associated  
 1491 with callerOrigin, if one is available.  
 1492  
 1493 11. Let clientDataJSON be the JSON-serialized client data constructed  
 1494 from collectedClientData.  
 1495 12. Let clientDataHash be the hash of the serialized client data  
 1496 represented by clientDataJSON.  
 1497 13. If the options.signal is present and its aborted flag is set to  
 1498 true, return a DOMException whose name is "AbortError" and  
 1499 terminate this algorithm.  
 1500 14. Let issuedRequests be a new ordered set.  
 1501 15. Let authenticator be a platform-specific handle whose value  
 1502 identifies an authenticator.  
 1503 16. Start lifetimeTimer.  
 1504 17. For each authenticator that becomes available on this platform  
 1505 during the lifetime of lifetimeTimer, perform the following steps:

1371 The definitions of "lifetime of" and "becomes available" are  
 1372 intended to represent how devices are hotplugged into (USB) or  
 1373 discovered by (NFC) browsers, and are under-specified. Resolving  
 1374 this with good definitions or some other means will be addressed by  
 1375 resolving Issue #613.  
 1376 1. If options.userVerification is set to required and the  
 1377 authenticator is not capable of performing user verification,  
 1378 continue.  
 1379 2. Let userVerification be the effective user verification  
 1380 requirement for assertion, a Boolean value, as follows. If  
 1381 options.userVerification  
 1382  
 1383 is set to required  
 1384 Let userVerification be true.  
 1385  
 1386 is set to preferred  
 1387 If the authenticator  
 1388  
 1389 is capable of user verification  
 1390 Let userVerification be true.  
 1391  
 1392 is not capable of user verification  
 1393 Let userVerification be false.  
 1394  
 1395 is set to discouraged  
 1396 Let userVerification be false.  
 1397  
 1398 3. Let userPresence be a Boolean value set to the inverse of  
 1399 userVerification.  
 1400 4. Let allowCredentialDescriptorList be a new list.  
 1401 5. If options.allowCredentials is not empty, execute a  
 1402 platform-specific procedure to determine which, if any, public  
 1403 key credentials described by options.allowCredentials are  
 1404 bound to this authenticator, by matching with rpId,  
 1405 options.allowCredentials.id, and  
 1406 options.allowCredentials.type. Set  
 1407 allowCredentialDescriptorList to this filtered list.  
 1408 6. If allowCredentialDescriptorList  
 1409  
 1410 is not empty  
 1411  
 1412 1. Let distinctTransports be a new ordered set.  
 1413 2. If allowCredentialDescriptorList has exactly one  
 1414  
 1415 value, let savedCredentialId be a new  
 1416 PublicKeyCredentialDescriptor.id and set its value  
 1417 to allowCredentialDescriptorList[0].id's value (see  
 1418 here in 6.2.2 The authenticatorGetAssertion  
 1419 operation for more information).  
 1420  
 1421 The foregoing step `_may_` be incorrect, in that we  
 1422 are attempting to create `savedCredentialId` here and  
 1423 use it later below, and we do not have a global in  
 1424 which to allocate a place for it. Perhaps this is  
 1425 good enough? addendum: @jones feels the above step  
 1426 is likely good enough.  
 1427  
 1428 1. For each credential descriptor C in  
 1429 allowCredentialDescriptorList, append each value, if  
 1430 any, of C.transports to distinctTransports.  
 1431 Note: This will aggregate only distinct values of  
 transports (for this authenticator) in

1506 The definitions of "lifetime of" and "becomes available" are  
 1507 intended to represent how devices are hot-plugged into (USB) or  
 1508 discovered by (NFC) browsers, and are underspecified. Resolving  
 1509 this with good definitions or some other means will be addressed by  
 1510 resolving Issue #613.  
 1511 1. If options.userVerification is set to required and the  
 1512 authenticator is not capable of performing user verification,  
 1513 continue.  
 1514 2. Let userVerification be the effective user verification  
 1515 requirement for assertion, a Boolean value, as follows. If  
 1516 options.userVerification  
 1517  
 1518 is set to required  
 1519 Let userVerification be true.  
 1520  
 1521 is set to preferred  
 1522 If the authenticator  
 1523  
 1524 is capable of user verification  
 1525 Let userVerification be true.  
 1526  
 1527 is not capable of user verification  
 1528 Let userVerification be false.  
 1529  
 1530 is set to discouraged  
 1531 Let userVerification be false.  
 1532  
 1533 3. Let userPresence be a Boolean value set to the inverse of  
 1534 userVerification.  
 1535 4. If options.allowCredentials  
 1536  
 1537 is not empty  
 1538  
 1539 1. Let allowCredentialDescriptorList be a new list.  
 1540 2. Execute a platform-specific procedure to determine  
 1541 which, if any, public key credentials described by  
 1542 options.allowCredentials are bound to this  
 1543 authenticator, by matching with rpId,  
 1544 options.allowCredentials.id, and  
 1545 options.allowCredentials.type. Set  
 1546 allowCredentialDescriptorList to this filtered list.  
 1547 3. If allowCredentialDescriptorList is empty, continue.  
 1548 4. Let distinctTransports be a new ordered set.  
 1549 5. If allowCredentialDescriptorList has exactly one  
 1550 value, let savedCredentialId be a new  
 1551 PublicKeyCredentialDescriptor.id and set its value  
 1552 to allowCredentialDescriptorList[0].id's value (see  
 1553 here in 6.2.3 The authenticatorGetAssertion  
 1554 operation for more information).  
 1555  
 1556 The foregoing step `_may_` be incorrect, in that we  
 1557 are attempting to create `savedCredentialId` here and  
 1558 use it later below, and we do not have a global in  
 1559 which to allocate a place for it. Perhaps this is  
 1560 good enough? addendum: @jones feels the above step  
 1561 is likely good enough.  
 1562  
 1563 1. For each credential descriptor C in  
 1564 allowCredentialDescriptorList, append each value, if  
 1565 any, of C.transports to distinctTransports.  
 1566 Note: This will aggregate only distinct values of  
 transports (for this authenticator) in

1432 distinctTransports due to the properties of ordered  
1433 sets.  
1434 2. If distinctTransports  
1435  
1436 is not empty  
1437 The client selects one transport value  
1438 from distinctTransports, possibly  
1439 incorporating local configuration  
1440 knowledge of the appropriate transport  
1441 to use with authenticator in making its  
1442 selection.  
1443  
1444 Then, using transport, invoke the  
1445 authenticatorGetAssertion operation on  
1446 authenticator, with rpId,  
1447 clientDataHash,  
1448 allowCredentialDescriptorList,  
1449 userPresence, userVerification, and  
1450 authenticatorExtensions as parameters.  
1451  
1452 is empty  
1453 Using local configuration knowledge of  
1454 the appropriate transport to use with  
1455 authenticator, invoke the  
1456 authenticatorGetAssertion operation on  
1457 authenticator with rpId, clientDataHash,  
1458 allowCredentialDescriptorList,  
1459 userPresence, userVerification, and  
1460 clientExtensions as parameters.  
1461  
1462 is empty  
1463 Using local configuration knowledge of the  
1464 appropriate transport to use with authenticator,  
1465 invoke the authenticatorGetAssertion operation on  
1466 authenticator with rpId, clientDataHash,  
1467 userPresence, userVerification and clientExtensions  
1468 as parameters.  
1469  
1470 Note: In this case, the Relying Party did not supply  
1471 a list of acceptable credential descriptors. Thus  
1472 the authenticator is being asked to exercise any  
1473 credential it may possess that is bound to the  
1474 Relying Party, as identified by rpId.  
1475  
1476 7. Append authenticator to issuedRequests.  
1477 18. While issuedRequests is not empty, perform the following actions  
1478 depending upon lifetimeTimer and responses from the authenticators:  
1479  
1480 If lifetimeTimer expires,  
1481 For each authenticator in issuedRequests invoke the  
1482 authenticatorCancel operation on authenticator and remove  
1483 authenticator from issuedRequests.  
1484  
1485 If the signal member is present and the aborted flag is set to  
1486 true,  
1487 For each authenticator in issuedRequests invoke the  
1488 authenticatorCancel operation on authenticator and remove  
1489 authenticator from issuedRequests. Then return a  
1490 DOMException whose name is "AbortError" and terminate this  
1491 algorithm.  
1492  
1493 If any authenticator returns a status indicating that the user  
1494 cancelled the operation,  
1495  
1496 1. Remove authenticator from issuedRequests.  
1497 2. For each remaining authenticator in issuedRequests invoke  
1498 the authenticatorCancel operation on authenticator and  
1499 remove it from issuedRequests.

1568 distinctTransports due to the properties of ordered  
1569 sets.  
1570 2. If distinctTransports  
1571  
1572 is not empty  
1573 The client selects one transport value  
1574 from distinctTransports, possibly  
1575 incorporating local configuration  
1576 knowledge of the appropriate transport  
1577 to use with authenticator in making its  
1578 selection.  
1579  
1580 Then, using transport, invoke the  
1581 authenticatorGetAssertion operation on  
1582 authenticator, with rpId,  
1583 clientDataHash,  
1584 allowCredentialDescriptorList,  
1585 userPresence, userVerification, and  
1586 authenticatorExtensions as parameters.  
1587  
1588 is empty  
1589 Using local configuration knowledge of  
1590 the appropriate transport to use with  
1591 authenticator, invoke the  
1592 authenticatorGetAssertion operation on  
1593 authenticator with rpId, clientDataHash,  
1594 allowCredentialDescriptorList,  
1595 userPresence, userVerification, and  
1596 clientExtensions as parameters.  
1597  
1598 is empty  
1599 Using local configuration knowledge of the  
1600 appropriate transport to use with authenticator,  
1601 invoke the authenticatorGetAssertion operation on  
1602 authenticator with rpId, clientDataHash,  
1603 userPresence, userVerification and clientExtensions  
1604 as parameters.  
1605  
1606 Note: In this case, the Relying Party did not supply  
1607 a list of acceptable credential descriptors. Thus,  
1608 the authenticator is being asked to exercise any  
1609 credential it may possess that is bound to the  
1610 Relying Party, as identified by rpId.  
1611  
1612 5. Append authenticator to issuedRequests.  
1613 18. While lifetimeTimer has not expired, perform the following actions  
1614 depending upon lifetimeTimer and responses from the authenticators:  
1615  
1616 If lifetimeTimer expires,  
1617 For each authenticator in issuedRequests invoke the  
1618 authenticatorCancel operation on authenticator and remove  
1619 authenticator from issuedRequests.  
1620  
1621 If the signal member is present and the aborted flag is set to  
1622 true,  
1623 For each authenticator in issuedRequests invoke the  
1624 authenticatorCancel operation on authenticator and remove  
1625 authenticator from issuedRequests. Then return a  
1626 DOMException whose name is "AbortError" and terminate this  
1627 algorithm.  
1628  
1629 If any authenticator returns a status indicating that the user  
1630 cancelled the operation,  
1631  
1632 1. Remove authenticator from issuedRequests.  
1633 2. For each remaining authenticator in issuedRequests invoke  
1634 the authenticatorCancel operation on authenticator and  
1635 remove it from issuedRequests.  
1636 Note: Authenticators may return an indication of "the  
1637 user cancelled the entire operation". How a user agent

1500 If any authenticator returns an error status,  
 1501 Remove authenticator from issuedRequests.  
 1502  
 1503 If any authenticator indicates success,  
 1504  
 1505 1. Remove authenticator from issuedRequests.  
 1506 2. Let assertionCreationData be a struct whose items are:  
 1507  
 1508 credentialIdResult  
 1509 If savedCredentialId exists, set the value of  
 1510 credentialIdResult to be the bytes of  
 1511 savedCredentialId. Otherwise, set the value of  
 1512 credentialIdResult to be the bytes of the  
 1513 credential ID returned from the successful  
 1514 authenticatorGetAssertion operation, as  
 1515 defined in 6.2.2 The  
 1516 authenticatorGetAssertion operation.  
 1517  
 1518 clientDataJSONResult  
 1519 whose value is the bytes of clientDataJSON.  
 1520  
 1521 authenticatorDataResult  
 1522 whose value is the bytes of the authenticator  
 1523 data returned by the authenticator.  
 1524  
 1525 signatureResult  
 1526 whose value is the bytes of the signature  
 1527 value returned by the authenticator.  
 1528  
 1529 userHandleResult  
 1530 whose value is the bytes of the user handle  
 1531 returned by the authenticator.  
 1532  
 1533  
 1534 clientExtensionResults  
 1535 whose value is an AuthenticationExtensions  
 1536 object containing extension identifier ->  
 1537 client extension output entries. The entries  
 1538 are created by running each extension's client  
 1539 extension processing algorithm to create the  
 1540 client extension outputs, for each client  
 1541 extension in clientDataJSON.clientExtensions.  
 1542  
 1543 3. Let constructAssertionAlg be an algorithm that takes a  
 1544 global object global, and whose steps are:  
 1545 1. Let pubKeyCred be a new PublicKeyCredential object  
 1546 associated with global whose fields are:  
 1547  
 1548 [[identifier]]  
 1549 A new ArrayBuffer, created using  
 1550 global's %ArrayBuffer%, containing the  
 1551 bytes of  
 1552 assertionCreationData.credentialIdResult  
 1553  
 1554  
 1555 response  
 1556 A new AuthenticatorAssertionResponse  
 1557 object associated with global whose  
 1558 fields are:  
 1559  
 1560 clientDataJSON  
 1561 A new ArrayBuffer, created using  
 1562 global's %ArrayBuffer%, containing  
 1563 the bytes of  
 1564 assertionCreationData.clientDataJS  
 1565 ONResult.

1638 manifests this state to users is unspecified.  
 1639  
 1640 If any authenticator returns an error status,  
 1641 Remove authenticator from issuedRequests.  
 1642  
 1643 If any authenticator indicates success,  
 1644  
 1645 1. Remove authenticator from issuedRequests.  
 1646 2. Let assertionCreationData be a struct whose items are:  
 1647  
 1648 credentialIdResult  
 1649 If savedCredentialId exists, set the value of  
 1650 credentialIdResult to be the bytes of  
 1651 savedCredentialId. Otherwise, set the value of  
 1652 credentialIdResult to be the bytes of the  
 1653 credential ID returned from the successful  
 1654 authenticatorGetAssertion operation, as  
 1655 defined in 6.2.3 The  
 1656 authenticatorGetAssertion operation.  
 1657  
 1658 clientDataJSONResult  
 1659 whose value is the bytes of clientDataJSON.  
 1660  
 1661 authenticatorDataResult  
 1662 whose value is the bytes of the authenticator  
 1663 data returned by the authenticator.  
 1664  
 1665 signatureResult  
 1666 whose value is the bytes of the signature  
 1667 value returned by the authenticator.  
 1668  
 1669 userHandleResult  
 1670 If the authenticator returned a user handle,  
 1671 set the value of userHandleResult to be the  
 1672 bytes of the returned user handle. Otherwise,  
 1673 set the value of userHandleResult to null.  
 1674  
 1675 clientExtensionResults  
 1676 whose value is an  
 1677 AuthenticationExtensionsClientOutputs object  
 1678 containing extension identifier -> client  
 1679 extension output entries. The entries are  
 1680 created by running each extension's client  
 1681 extension processing algorithm to create the  
 1682 client extension outputs, for each client  
 1683 extension in clientDataJSON.clientExtensions.  
 1684  
 1685 3. Let constructAssertionAlg be an algorithm that takes a  
 1686 global object global, and whose steps are:  
 1687 1. Let pubKeyCred be a new PublicKeyCredential object  
 1688 associated with global whose fields are:  
 1689  
 1690 [[identifier]]  
 1691 A new ArrayBuffer, created using  
 1692 global's %ArrayBuffer%, containing the  
 1693 bytes of  
 1694 assertionCreationData.credentialIdResult  
 1695  
 1696  
 1697 response  
 1698 A new AuthenticatorAssertionResponse  
 1699 object associated with global whose  
 1700 fields are:  
 1701  
 1702 clientDataJSON  
 1703 A new ArrayBuffer, created using  
 1704 global's %ArrayBuffer%, containing  
 1705 the bytes of  
 1706 assertionCreationData.clientDataJS  
 1707 ONResult.

```

1566 authenticatorData
1567   A new ArrayBuffer, created using
1568   global's %ArrayBuffer%, containing
1569   the bytes of
1570   assertionCreationData.authenticato
1571   rDataResult.
1572
1573 signature
1574   A new ArrayBuffer, created using
1575   global's %ArrayBuffer%, containing
1576   the bytes of
1577   assertionCreationData.signatureRes
1578   ult.
1579
1580 userHandle
1581   A new ArrayBuffer, created using
1582
1583
1584   global's %ArrayBuffer%, containing
1585   the bytes of
1586   assertionCreationData.userHandleRe
1587   sult.
1588
1589 [[clientExtensionsResults]]
1590   A new ArrayBuffer, created using
1591   global's %ArrayBuffer%, containing the
1592   bytes of
1593   assertionCreationData.clientExtensionRes
1594   ults.
1595
1596 2. Return pubKeyCred.
1597 4. For each remaining authenticator in issuedRequests invoke
1598   the authenticatorCancel operation on authenticator and
1599   remove it from issuedRequests.
1600 5. Return constructAssertionAlg and terminate this
1601   algorithm.
1602
1603 19. Return a DOMException whose name is "NotAllowedError".
1604
1605 During the above process, the user agent SHOULD show some UI to the
1606 user to guide them in the process of selecting and authorizing an
1607 authenticator with which to complete the operation.
1608
1609 5.1.5. Store an existing credential - PublicKeyCredential's
1610 [[Store]](credential, sameOriginWithAncestors) method
1611
1612 The [[Store]](credential, sameOriginWithAncestors) method is not
1613 supported for Web Authentication's PublicKeyCredential type, so it
1614 always returns an error.
1615
1616 Note: This algorithm is synchronous; the Promise resolution/rejection
1617 is handled by navigator.credentials.store().
1618
1619 This internal method accepts two arguments:
1620
1621 credential
1622   This argument is a PublicKeyCredential object.
1623
1624 sameOriginWithAncestors
1625   This argument is a boolean which is true if and only if the
1626   caller's environment settings object is same-origin with its
1627   ancestors.
1628
1629 When this method is invoked, the user agent MUST execute the following

```

```

1708 authenticatorData
1709   A new ArrayBuffer, created using
1710   global's %ArrayBuffer%, containing
1711   the bytes of
1712   assertionCreationData.authenticato
1713   rDataResult.
1714
1715 signature
1716   A new ArrayBuffer, created using
1717   global's %ArrayBuffer%, containing
1718   the bytes of
1719   assertionCreationData.signatureRes
1720   ult.
1721
1722 userHandle
1723   If
1724   assertionCreationData.userHandleRe
1725   sult is null, set this field to
1726   null. Otherwise, set this field to
1727   a new ArrayBuffer, created using
1728   global's %ArrayBuffer%, containing
1729   the bytes of
1730   assertionCreationData.userHandleRe
1731   sult.
1732
1733 [[clientExtensionsResults]]
1734   A new ArrayBuffer, created using
1735   global's %ArrayBuffer%, containing the
1736   bytes of
1737   assertionCreationData.clientExtensionRes
1738   ults.
1739
1740 2. Return pubKeyCred.
1741 4. For each remaining authenticator in issuedRequests invoke
1742   the authenticatorCancel operation on authenticator and
1743   remove it from issuedRequests.
1744 5. Return constructAssertionAlg and terminate this
1745   algorithm.
1746
1747 19. Return a DOMException whose name is "NotAllowedError". In order to
1748   prevent information leak that could identify the user without
1749   consent, this step MUST NOT be executed before lifetimeTimer has
1750   expired. See 14.3 Authentication Ceremony Privacy for details.
1751
1752 During the above process, the user agent SHOULD show some UI to the
1753 user to guide them in the process of selecting and authorizing an
1754 authenticator with which to complete the operation.
1755
1756 5.1.5. Store an existing credential - PublicKeyCredential's
1757 [[Store]](credential, sameOriginWithAncestors) method
1758
1759 The [[Store]](credential, sameOriginWithAncestors) method is not
1760 supported for Web Authentication's PublicKeyCredential type, so it
1761 always returns an error.
1762
1763 Note: This algorithm is synchronous; the Promise resolution/rejection
1764 is handled by navigator.credentials.store().
1765
1766 This internal method accepts two arguments:
1767
1768 credential
1769   This argument is a PublicKeyCredential object.
1770
1771 sameOriginWithAncestors
1772   This argument is a boolean which is true if and only if the
1773   caller's environment settings object is same-origin with its
1774   ancestors.
1775
1776 When this method is invoked, the user agent MUST execute the following

```

1625 algorithm:  
 1630 1. Return a DOMException whose name is "NotSupportedError", and  
 1631 terminate this algorithm  
 1632  
 1633 5.1.6. Availability of User-Verifying Platform Authenticator -

1634 PublicKeyCredential's isUserVerifyingPlatformAuthenticatorAvailable() method  
 1635  
 1636 Relying Parties use this method to determine whether they can create a  
 1637 new credential using a user-verifying platform authenticator. Upon  
 1638 invocation, the client employs a platform-specific procedure to  
 1639 discover available user-verifying platform authenticators. If  
 1640 successful, the client then assesses whether the user is willing to  
 1641 create a credential using one of the available user-verifying platform  
 1642 authenticators. This assessment may include various factors, such as:  
 1643 \* Whether the user is running in private or incognito mode.  
 1644 \* Whether the user has configured the client to not create such  
 1645 credentials.  
 1646 \* Whether the user has previously expressed an unwillingness to  
 1647 create a new credential for this Relying Party, either through  
 1648 configuration or by declining a user interface prompt.  
 1649 \* The user's explicitly stated intentions, determined through user  
 1650 interaction.

1651 If this assessment is affirmative, the promise is resolved with the  
 1652 value of True. Otherwise, the promise is resolved with the value of  
 1653 False. Based on the result, the Relying Party can take further actions  
 1654 to guide the user to create a credential.  
 1655  
 1656 This method has no arguments and returns a boolean value.  
 1657  
 1658 If the promise will return False, the client SHOULD wait a fixed period  
 1659 of time from the invocation of the method before returning False. This  
 1660 is done so that callers can not distinguish between the case where the  
 1661 user was unwilling to create a credential using one of the available  
 1662 user-verifying platform authenticators and the case where no  
 1663 user-verifying platform authenticator exists. Trying to make these  
 1664 cases indistinguishable is done in an attempt to not provide additional  
 1665 information that could be used for fingerprinting. A timeout value on  
 1666 the order of 10 minutes is recommended; this is enough time for  
 1667 successful user interactions to be performed but short enough that the  
 1668 dangling promise will still be resolved in a reasonably timely fashion.

```
1670 partial interface PublicKeyCredential {
1671   static Promise < boolean > isUserVerifyingPlatformAuthenticatorAvailable();
1672 };
```

1673  
 1674 5.2. Authenticator Responses (interface AuthenticatorResponse)  
 1675  
 1676 Authenticators respond to Relying Party requests by returning an object  
 1677 derived from the AuthenticatorResponse interface:  
 1678 [SecureContext, Exposed=Window]  
 1679 interface AuthenticatorResponse {  
 1680 [SameObject] readonly attribute ArrayBuffer clientDataJSON;  
 1681 };

1682  
 1683 clientDataJSON, of type ArrayBuffer, readonly  
 1684 This attribute contains a JSON serialization of the client data  
 1685 passed to the authenticator by the client in its call to either  
 1686 create() or get().

1778 algorithm:  
 1779 1. Return a DOMException whose name is "NotSupportedError", and  
 1780 terminate this algorithm  
 1781  
 1782 5.1.6. Preventing silent access to an existing credential -  
 1783 PublicKeyCredential's [[preventSilentAccess]](credential,  
 1784 sameOriginWithAncestors) method  
 1785  
 1786 Calling the [[preventSilentAccess]](credential,  
 1787 sameOriginWithAncestors) method will have no effect on authenticators  
 1788 that require an authorization gesture, but setting that flag may  
 1789 potentially exclude authenticators that can operate without user  
 1790 intervention.  
 1791  
 1792 This internal method accepts no arguments.  
 1793  
 1794 5.1.7. Availability of User-Verifying Platform Authenticator -  
 1795 PublicKeyCredential's isUserVerifyingPlatformAuthenticatorAvailable() method  
 1796  
 1797 Relying Parties use this method to determine whether they can create a  
 1798 new credential using a user-verifying platform authenticator. Upon  
 1799 invocation, the client employs a platform-specific procedure to  
 1800 discover available user-verifying platform authenticators. If  
 1801 successful, the client then assesses whether the user is willing to  
 1802 create a credential using one of the available user-verifying platform  
 1803 authenticators. This assessment may include various factors, such as:  
 1804 \* Whether the user is running in private or incognito mode.  
 1805 \* Whether the user has configured the client to not create such  
 1806 credentials.  
 1807 \* Whether the user has previously expressed an unwillingness to  
 1808 create a new credential for this Relying Party, either through  
 1809 configuration or by declining a user interface prompt.  
 1810 \* The user's explicitly stated intentions, determined through user  
 1811 interaction.

1812 If this assessment is affirmative, the promise is resolved with the  
 1813 value of True. Otherwise, the promise is resolved with the value of  
 1814 False. Based on the result, the Relying Party can take further actions  
 1815 to guide the user to create a credential.  
 1816  
 1817 This method has no arguments and returns a boolean value.  
 1818  
 1819 If the promise will return False, the client SHOULD wait a fixed period  
 1820 of time from the invocation of the method before returning False. This  
 1821 is done so that callers cannot distinguish between the case where the  
 1822 user was unwilling to create a credential using one of the available  
 1823 user-verifying platform authenticators and the case where no  
 1824 user-verifying platform authenticator exists. Trying to make these  
 1825 cases indistinguishable is done in an attempt to not provide additional  
 1826 information that could be used for fingerprinting. A timeout value on  
 1827 the order of 10 minutes is recommended; this is enough time for  
 1828 successful user interactions to be performed but short enough that the  
 1829 dangling promise will still be resolved in a reasonably timely fashion.

```
1830 partial interface PublicKeyCredential {
1831   static Promise < boolean > isUserVerifyingPlatformAuthenticatorAvailable();
1832 };
```

1833  
 1834 5.2. Authenticator Responses (interface AuthenticatorResponse)  
 1835  
 1836 Authenticators respond to Relying Party requests by returning an object  
 1837 derived from the AuthenticatorResponse interface:  
 1838 [SecureContext, Exposed=Window]  
 1839 interface AuthenticatorResponse {  
 1840 [SameObject] readonly attribute ArrayBuffer clientDataJSON;  
 1841 };

1842  
 1843 clientDataJSON, of type ArrayBuffer, readonly  
 1844 This attribute contains a JSON serialization of the client data  
 1845 passed to the authenticator by the client in its call to either  
 1846 create() or get().

1687  
 1688 5.2.1. Information about Public Key Credential (interface  
 1689 AuthenticatorAttestationResponse)  
 1690  
 1691 The AuthenticatorAttestationResponse interface represents the  
 1692 authenticator's response to a client's request for the creation of a  
 1693 new public key credential. It contains information about the new  
 1694 credential that can be used to identify it for later use, and metadata  
 1695 that can be used by the Relying Party to assess the characteristics of  
 1696 the credential during registration.  
 1697 [SecureContext, Exposed=Window]  
 1698 interface AuthenticatorAttestationResponse : AuthenticatorResponse {  
 1699 [SameObject] readonly attribute ArrayBuffer attestationObject;  
 1700 };  
 1701  
 1702 clientDataJSON  
 1703 This attribute, inherited from AuthenticatorResponse, contains  
 1704 the JSON-serialized client data (see 6.3 Attestation) passed to  
 1705 the authenticator by the client in order to generate this  
 1706 credential. The exact JSON serialization must be preserved, as  
 1707 the hash of the serialized client data has been computed over  
 1708 it.  
 1709  
 1710 attestationObject, of type ArrayBuffer, readonly  
 1711 This attribute contains an attestation object, which is opaque  
 1712 to, and cryptographically protected against tampering by, the  
 1713 client. The attestation object contains both authenticator data  
 1714 and an attestation statement. The former contains the AAGUID, a  
 1715 unique credential ID, and the credential public key. The  
 1716 contents of the attestation statement are determined by the  
 1717 attestation statement format used by the authenticator. It also  
 1718 contains any additional information that the Relying Party's  
 1719 server requires to validate the attestation statement, as well  
 1720 as to decode and validate the authenticator data along with the  
 1721 JSON-serialized client data. For more details, see 6.3  
 1722 Attestation, 6.3.4 Generating an Attestation Object, and Figure  
 1723 3.  
 1724  
 1725 5.2.2. Web Authentication Assertion (interface  
 1726 AuthenticatorAssertionResponse)  
 1727  
 1728 The AuthenticatorAssertionResponse interface represents an  
 1729 authenticator's response to a client's request for generation of a new  
 1730 authentication assertion given the Relying Party's challenge and  
 1731 optional list of credentials it is aware of. This response contains a  
 1732 cryptographic signature proving possession of the credential private  
 1733 key, and optionally evidence of user consent to a specific transaction.  
 1734 [SecureContext, Exposed=Window]  
 1735 interface AuthenticatorAssertionResponse : AuthenticatorResponse {  
 1736 [SameObject] readonly attribute ArrayBuffer authenticatorData;  
 1737 [SameObject] readonly attribute ArrayBuffer signature;  
 1738 [SameObject] readonly attribute ArrayBuffer userHandle;  
 1739 };  
 1740  
 1741 clientDataJSON  
 1742 This attribute, inherited from AuthenticatorResponse, contains  
 1743 the JSON-serialized client data (see 5.8.1 Client data used in  
 1744 WebAuthn signatures (dictionary CollectedClientData)) passed to  
 1745 the authenticator by the client in order to generate this  
 1746 assertion. The exact JSON serialization **must** be preserved, as  
 1747 the hash of the serialized client data has been computed over  
 1748 it.  
 1749  
 1750 authenticatorData, of type ArrayBuffer, readonly  
 1751 This attribute contains the authenticator data returned by the  
 1752 authenticator. See 6.1 Authenticator data.  
 1753  
 1754 signature, of type ArrayBuffer, readonly  
 1755 This attribute contains the raw signature returned from the  
 1756 authenticator. See 6.2.2 The authenticatorGetAssertion

1848  
 1849 5.2.1. Information about Public Key Credential (interface  
 1850 AuthenticatorAttestationResponse)  
 1851  
 1852 The AuthenticatorAttestationResponse interface represents the  
 1853 authenticator's response to a client's request for the creation of a  
 1854 new public key credential. It contains information about the new  
 1855 credential that can be used to identify it for later use, and metadata  
 1856 that can be used by the Relying Party to assess the characteristics of  
 1857 the credential during registration.  
 1858 [SecureContext, Exposed=Window]  
 1859 interface AuthenticatorAttestationResponse : AuthenticatorResponse {  
 1860 [SameObject] readonly attribute ArrayBuffer attestationObject;  
 1861 };  
 1862  
 1863 clientDataJSON  
 1864 This attribute, inherited from AuthenticatorResponse, contains  
 1865 the JSON-serialized client data (see 6.3 Attestation) passed to  
 1866 the authenticator by the client in order to generate this  
 1867 credential. The exact JSON serialization must be preserved, as  
 1868 the hash of the serialized client data has been computed over  
 1869 it.  
 1870  
 1871 attestationObject, of type ArrayBuffer, readonly  
 1872 This attribute contains an attestation object, which is opaque  
 1873 to, and cryptographically protected against tampering by, the  
 1874 client. The attestation object contains both authenticator data  
 1875 and an attestation statement. The former contains the AAGUID, a  
 1876 unique credential ID, and the credential public key. The  
 1877 contents of the attestation statement are determined by the  
 1878 attestation statement format used by the authenticator. It also  
 1879 contains any additional information that the Relying Party's  
 1880 server requires to validate the attestation statement, as well  
 1881 as to decode and validate the authenticator data along with the  
 1882 JSON-serialized client data. For more details, see 6.3  
 1883 Attestation, 6.3.4 Generating an Attestation Object, and Figure  
 1884 3.  
 1885  
 1886 5.2.2. Web Authentication Assertion (interface  
 1887 AuthenticatorAssertionResponse)  
 1888  
 1889 The AuthenticatorAssertionResponse interface represents an  
 1890 authenticator's response to a client's request for generation of a new  
 1891 authentication assertion given the Relying Party's challenge and  
 1892 optional list of credentials it is aware of. This response contains a  
 1893 cryptographic signature proving possession of the credential private  
 1894 key, and optionally evidence of user consent to a specific transaction.  
 1895 [SecureContext, Exposed=Window]  
 1896 interface AuthenticatorAssertionResponse : AuthenticatorResponse {  
 1897 [SameObject] readonly attribute ArrayBuffer authenticatorData;  
 1898 [SameObject] readonly attribute ArrayBuffer signature;  
 1899 [SameObject] readonly attribute ArrayBuffer userHandle;  
 1900 };  
 1901  
 1902 clientDataJSON  
 1903 This attribute, inherited from AuthenticatorResponse, contains  
 1904 the JSON-serialized client data (see 5.10.1 Client data used in  
 1905 WebAuthn signatures (dictionary CollectedClientData)) passed to  
 1906 the authenticator by the client in order to generate this  
 1907 assertion. The exact JSON serialization **MUST** be preserved, as  
 1908 the hash of the serialized client data has been computed over  
 1909 it.  
 1910  
 1911 authenticatorData, of type ArrayBuffer, readonly  
 1912 This attribute contains the authenticator data returned by the  
 1913 authenticator. See 6.1 Authenticator data.  
 1914  
 1915 signature, of type ArrayBuffer, readonly  
 1916 This attribute contains the raw signature returned from the  
 1917 authenticator. See 6.2.3 The authenticatorGetAssertion

1757 operation.  
 1758  
 1759 userHandle, of type ArrayBuffer, readonly  
 1760 This attribute contains the user handle returned from the  
 1761 authenticator. See 6.2.2 The authenticator **GetAssertion**  
 1762 **operation**.  
 1763  
 1764 5.3. Parameters for Credential Generation (dictionary  
 1765 PublicKeyCredentialParameters)  
 1766  
 1767 dictionary PublicKeyCredentialParameters {  
 1768 required PublicKeyCredentialType type;  
 1769 required COSEAlgorithmIdentifier alg;  
 1770 };  
 1771  
 1772 This dictionary is used to supply additional parameters when creating a  
 1773 new credential.  
 1774  
 1775 The type member specifies the type of credential to be created.  
 1776  
 1777 The alg member specifies the cryptographic signature algorithm with  
 1778 which the newly generated credential will be used, and thus also the  
 1779 type of asymmetric key pair to be generated, e.g., RSA or Elliptic  
 1780 Curve.  
 1781  
 1782 Note: we use "alg" as the latter member name, rather than spelling-out  
 1783 "algorithm", because it will be serialized into a message to the  
 1784 authenticator, which may be sent over a low-bandwidth link.  
 1785  
 1786 5.4. Options for Credential Creation (dictionary  
 1787 MakePublicKeyCredentialOptions)  
 1788  
 1789 dictionary MakePublicKeyCredentialOptions {  
 1790 required PublicKeyCredentialRpEntity rp;  
 1791 required PublicKeyCredentialUserEntity user;  
 1792  
 1793 required BufferSource challenge;  
 1794 required sequence<PublicKeyCredentialParameters> pubKeyCredParams;  
 1795  
 1796 unsigned long timeout;  
 1797 sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];  
 1798 AuthenticatorSelectionCriteria authenticatorSelection;  
 1799 AttestationConveyancePreference attestation = "none";  
 1800 AuthenticationExtensions extensions;  
 1801 };  
 1802  
 1803 rp, of type PublicKeyCredentialRpEntity  
 1804 This member contains data about the Relying Party responsible  
 1805 for the request.  
 1806  
 1807 Its value's name member contains the friendly name of the  
 1808 Relying Party (e.g. "Acme Corporation", "Widgets, Inc.", or  
 1809 "Awesome Site").  
 1810  
 1811 Its value's id member specifies the relying party identifier  
 1812 with which the credential should be associated. If omitted, its  
 1813 value will be the CredentialsContainer object's relevant  
 1814 settings object's origin's effective domain.  
 1815  
 1816 user, of type PublicKeyCredentialUserEntity  
 1817 This member contains data about the user account for which the  
 1818 Relying Party is requesting attestation.  
 1819  
 1820 Its value's name member contains a name for the user account  
 1821 (e.g., "john.p.smith@example.com" or "+14255551234").  
 1822  
 1823 Its value's displayName member contains a friendly name for the  
 1824 user account (e.g., "John P. Smith").  
 1825  
 1826 Its value's id member contains the user handle for the account,

1918 operation.  
 1919  
 1920 userHandle, of type ArrayBuffer, readonly, **nullable**  
 1921 This attribute contains the user handle returned from the  
 1922 authenticator, or null if the authenticator did not return a  
 1923 user handle. See 6.2.3 The authenticator **GetAssertion**  
 1924 **operation**.  
 1925  
 1926 5.3. Parameters for Credential Generation (dictionary  
 1927 PublicKeyCredentialParameters)  
 1928  
 1929 dictionary PublicKeyCredentialParameters {  
 1930 required PublicKeyCredentialType type;  
 1931 required COSEAlgorithmIdentifier alg;  
 1932 };  
 1933  
 1934 This dictionary is used to supply additional parameters when creating a  
 1935 new credential.  
 1936  
 1937 The type member specifies the type of credential to be created.  
 1938  
 1939 The alg member specifies the cryptographic signature algorithm with  
 1940 which the newly generated credential will be used, and thus also the  
 1941 type of asymmetric key pair to be generated, e.g., RSA or Elliptic  
 1942 Curve.  
 1943  
 1944 Note: we use "alg" as the latter member name, rather than spelling-out  
 1945 "algorithm", because it will be serialized into a message to the  
 1946 authenticator, which may be sent over a low-bandwidth link.  
 1947  
 1948 5.4. Options for Credential Creation (dictionary  
 1949 PublicKeyCredentialCreationOptions)  
 1950  
 1951 dictionary PublicKeyCredentialCreationOptions {  
 1952 required PublicKeyCredentialRpEntity rp;  
 1953 required PublicKeyCredentialUserEntity user;  
 1954  
 1955 required BufferSource challenge;  
 1956 required sequence<PublicKeyCredentialParameters> pubKeyCredParams;  
 1957  
 1958 unsigned long timeout;  
 1959 sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];  
 1960 AuthenticatorSelectionCriteria authenticatorSelection;  
 1961 AttestationConveyancePreference attestation = "none";  
 1962 AuthenticationExtensionsClientInputs extensions;  
 1963 };  
 1964  
 1965 rp, of type PublicKeyCredentialRpEntity  
 1966 This member contains data about the Relying Party responsible  
 1967 for the request.  
 1968  
 1969 Its value's name member is required.  
 1970  
 1971 Its value's id member specifies the relying party identifier  
 1972 with which the credential should be associated. If omitted, its  
 1973 value will be the CredentialsContainer object's relevant  
 1974 settings object's origin's effective domain.  
 1975  
 1976 user, of type PublicKeyCredentialUserEntity  
 1977 This member contains data about the user account for which the  
 1978 Relying Party is requesting attestation.  
 1979  
 1980 Its value's name, displayName and id members are required.

1827 specified by the Relying Party.  
 1828  
 1829 challenge, of type BufferSource  
 1830 This member contains a challenge intended to be used for  
 1831 generating the newly created credential's attestation object.  
 1832  
 1833 pubKeyCredParams, of type sequence<PublicKeyCredentialParameters>  
 1834 This member contains information about the desired properties of  
 1835 the credential to be created. The sequence is ordered from most  
 1836 preferred to least preferred. The platform makes a best-effort  
 1837 to create the most preferred credential that it can.  
 1838  
 1839 timeout, of type unsigned long  
 1840 This member specifies a time, in milliseconds, that the caller  
 1841 is willing to wait for the call to complete. This is treated as  
 1842 a hint, and **may** be overridden by the platform.  
 1843  
 1844 excludeCredentials, of type sequence<PublicKeyCredentialDescriptor>,  
 1845 defaulting to None  
 1846 This member is intended for use by Relying Parties that wish to  
 1847 limit the creation of multiple credentials for the same account  
 1848 on a single authenticator. The platform is requested to return  
 1849 an error if the new credential would be created on an  
 1850 authenticator that also contains one of the credentials  
 1851 enumerated in this parameter.  
 1852  
 1853 authenticatorSelection, of type AuthenticatorSelectionCriteria  
 1854 This member is intended for use by Relying Parties that wish to  
 1855 select the appropriate authenticators to participate in the  
 1856 create() operation.  
 1857  
 1858 attestation, of type AttestationConveyancePreference, defaulting to  
 1859 "none"  
 1860 This member is intended for use by Relying Parties that wish to  
 1861 express their preference for attestation conveyance. The default  
 1862 is none.  
 1863  
 1864 extensions, of type AuthenticationExtensions  
 1865 This member contains additional parameters requesting additional  
 1866 processing by the client and authenticator. For example, the  
 1867 caller may request that only authenticators with certain  
 1868 capabilities be used to create the credential, or that **particular**  
 1869 **information be returned in the attestation object. Some**  
 1870 **extensions are defined in 9 WebAuthn Extensions; consult the**  
 1871 **IANA "WebAuthn Extension Identifier" registry established by**  
 1872 **[WebAuthn-Registries] for an up-to-date list of registered**  
 1873 **WebAuthn Extensions.**  
 1874  
 1875 5.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)  
 1876  
 1877 The PublicKeyCredentialEntity dictionary describes a user account, or a  
 1878 Relying Party, with which a public key credential is associated.  
 1879 dictionary PublicKeyCredentialEntity {  
 1880 required DOMString name;  
 1881 USVString icon;  
 1882 };  
 1883  
 1884 name, of type DOMString  
 1885 A human-friendly identifier for the entity. **For example, this**  
 1886 **could be a company name for a Relying Party, or a user's name.**  
 1887 **This identifier is intended for display. Authenticators MUST**  
 1888 **accept and store a 64 byte minimum length for a name members's**  
 1889 **value. Authenticators MAY truncate a name member's value to a**  
 1890 **length equal to or greater than 64 bytes.**

1980  
 1981 challenge, of type BufferSource  
 1982 This member contains a challenge intended to be used for  
 1983 generating the newly created credential's attestation object.  
 1984  
 1985 pubKeyCredParams, of type sequence<PublicKeyCredentialParameters>  
 1986 This member contains information about the desired properties of  
 1987 the credential to be created. The sequence is ordered from most  
 1988 preferred to least preferred. The platform makes a best-effort  
 1989 to create the most preferred credential that it can.  
 1990  
 1991 timeout, of type unsigned long  
 1992 This member specifies a time, in milliseconds, that the caller  
 1993 is willing to wait for the call to complete. This is treated as  
 1994 a hint, and **MAY** be overridden by the platform.  
 1995  
 1996 excludeCredentials, of type sequence<PublicKeyCredentialDescriptor>,  
 1997 defaulting to None  
 1998 This member is intended for use by Relying Parties that wish to  
 1999 limit the creation of multiple credentials for the same account  
 2000 on a single authenticator. The platform is requested to return  
 2001 an error if the new credential would be created on an  
 2002 authenticator that also contains one of the credentials  
 2003 enumerated in this parameter.  
 2004  
 2005 authenticatorSelection, of type AuthenticatorSelectionCriteria  
 2006 This member is intended for use by Relying Parties that wish to  
 2007 select the appropriate authenticators to participate in the  
 2008 create() operation.  
 2009  
 2010 attestation, of type AttestationConveyancePreference, defaulting to  
 2011 "none"  
 2012 This member is intended for use by Relying Parties that wish to  
 2013 express their preference for attestation conveyance. The default  
 2014 is none.  
 2015  
 2016 extensions, of type AuthenticationExtensions**ClientInputs**  
 2017 This member contains additional parameters requesting additional  
 2018 processing by the client and authenticator. For example, the  
 2019 caller may request that only authenticators with certain  
 2020 capabilities be used to create the credential, or that  
 2021 **particular information be returned in the attestation object.**  
 2022 **Some extensions are defined in 9 WebAuthn Extensions; consult**  
 2023 **the IANA "WebAuthn Extension Identifier" registry established by**  
 2024 **[WebAuthn-Registries] for an up-to-date list of registered**  
 2025 **WebAuthn Extensions.**  
 2026  
 2027 5.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)  
 2028  
 2029 The PublicKeyCredentialEntity dictionary describes a user account, or a  
 2030 Relying Party, with which a public key credential is associated.  
 2031 dictionary PublicKeyCredentialEntity {  
 2032 required DOMString name;  
 2033 USVString icon;  
 2034 };  
 2035  
 2036 name, of type DOMString  
 2037 A human-readable name for the entity. **Its function depends on**  
 2038 **what the PublicKeyCredentialEntity represents:**  
 2039  
 2040 **+ When inherited by PublicKeyCredentialRpEntity it is a**  
 2041 **human-friendly identifier for the Relying Party, intended only**  
 2042 **for display. For example, "ACME Corporation", "Wonderful**  
 2043 **Widgets, Inc." or "Awesome Site".**  
 2044 **+ When inherited by PublicKeyCredentialUserEntity, it is a**  
 2045 **human-palatable identifier for a user account. It is intended**  
 2046 **only for display, and SHOULD allow the user to easily tell the**  
 2047 **difference between user accounts with similar displayNames.**  
 2048 **For example, "alex", "alex.p.mueller@example.com" or**

1891 icon, of type USVString  
 1892 A serialized URL which resolves to an image associated with the  
 1893 entity. For example, this could be a user's avatar or a Relying  
 1894 Party's logo. This URL MUST be an a priori authenticated URL.  
 1895 Authenticators MUST accept and store a 128 byte minimum length  
 1896 for a icon members's value. Authenticators MAY ignore a icon  
 1897 members's value if its length is greater than 128 bytes.  
 1898  
 1900 5.4.2. RP Parameters for Credential Generation (dictionary  
 1901 PublicKeyCredentialRpEntity)  
 1902  
 1903 The PublicKeyCredentialRpEntity dictionary is used to supply additional  
 1904 Relying Party attributes when creating a new credential.  
 1905 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {  
 1906 DOMString id;  
 1907 };  
 1908  
 1909 id, of type DOMString  
 1910 A unique identifier for the Relying Party entity, which sets the  
 1911 RP ID.  
 1912  
 1913 5.4.3. User Account Parameters for Credential Generation (dictionary  
 1914 PublicKeyCredentialUserEntity)  
 1915  
 1916 The PublicKeyCredentialUserEntity dictionary is used to supply  
 1917 additional user account attributes when creating a new credential.  
 1918 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {  
 1919 required BufferSource id;  
 1920 required DOMString displayName;  
 1921 };  
 1922  
 1923 id, of type BufferSource  
 1924 The user handle of the user account entity.  
 1925  
 1926 displayName, of type DOMString  
 1927 A friendly name for the user account (e.g., "John P. Smith").  
 1928 Authenticators MUST accept and store a 64 byte minimum length  
 1929 for a displayName members's value. Authenticators MAY truncate a  
 1930  
 1931 displayName member's value to a length equal to or greater than  
 1932 64 bytes.  
 1933  
 1934 5.4.4. Authenticator Selection Criteria (dictionary  
 1935 AuthenticatorSelectionCriteria)  
 1936  
 1937 Relying Parties may use the AuthenticatorSelectionCriteria dictionary  
 1938 to specify their requirements regarding authenticator attributes.  
 1939 dictionary AuthenticatorSelectionCriteria {  
 1940 AuthenticatorAttachment authenticatorAttachment;  
 1941 boolean requireResidentKey = false;  
 1942 UserVerificationRequirement userVerification = "preferred";  
 1943 };  
 1944  
 1945 authenticatorAttachment, of type AuthenticatorAttachment  
 1946 If this member is present, eligible authenticators are filtered  
 1947 to only authenticators attached with the specified 5.4.5  
 Authenticator Attachment enumeration (enum

2049 "+14255551234". The Relying Party MAY let the user choose  
 2050 this, and MAY restrict the choice as needed or appropriate.  
 2051 For example, a Relying Party might choose to map  
 2052 human-palatable username account identifiers to the name  
 2053 member of PublicKeyCredentialUserEntity.  
 2054  
 2055 Authenticators MUST accept and store a 64-byte minimum length  
 2056 for a name member's value. Authenticators MAY truncate a name  
 2057 member's value to a length equal to or greater than 64 bytes.  
 2058  
 2059 icon, of type USVString  
 2060 A serialized URL which resolves to an image associated with the  
 2061 entity. For example, this could be a user's avatar or a Relying  
 2062 Party's logo. This URL MUST be an a priori authenticated URL.  
 2063 Authenticators MUST accept and store a 128-byte minimum length  
 2064 for an icon member's value. Authenticators MAY ignore an icon  
 2065 member's value if its length is greater than 128 bytes.  
 2066  
 2067 5.4.2. RP Parameters for Credential Generation (dictionary  
 2068 PublicKeyCredentialRpEntity)  
 2069  
 2070 The PublicKeyCredentialRpEntity dictionary is used to supply additional  
 2071 Relying Party attributes when creating a new credential.  
 2072 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {  
 2073 DOMString id;  
 2074 };  
 2075  
 2076 id, of type DOMString  
 2077 A unique identifier for the Relying Party entity, which sets the  
 2078 RP ID.  
 2079  
 2080 5.4.3. User Account Parameters for Credential Generation (dictionary  
 2081 PublicKeyCredentialUserEntity)  
 2082  
 2083 The PublicKeyCredentialUserEntity dictionary is used to supply  
 2084 additional user account attributes when creating a new credential.  
 2085 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {  
 2086 required BufferSource id;  
 2087 required DOMString displayName;  
 2088 };  
 2089  
 2090 id, of type BufferSource  
 2091 The user handle of the user account entity.  
 2092  
 2093 displayName, of type DOMString  
 2094 A human-friendly name for the user account, intended only for  
 2095 display. For example, "Alex P. Miller" or ". The Relying  
 2096 Party SHOULD let the user choose this, and SHOULD NOT restrict  
 2097 the choice more than necessary.  
 2098  
 2099 Authenticators MUST accept and store a 64-byte minimum length  
 2100 for a displayName member's value. Authenticators MAY truncate a  
 2101 displayName member's value to a length equal to or greater than  
 2102 64 bytes.  
 2103  
 2104 5.4.4. Authenticator Selection Criteria (dictionary  
 2105 AuthenticatorSelectionCriteria)  
 2106  
 2107 Relying Parties may use the AuthenticatorSelectionCriteria dictionary  
 2108 to specify their requirements regarding authenticator attributes.  
 2109 dictionary AuthenticatorSelectionCriteria {  
 2110 AuthenticatorAttachment authenticatorAttachment;  
 2111 boolean requireResidentKey = false;  
 2112 UserVerificationRequirement userVerification = "preferred";  
 2113 };  
 2114  
 2115 authenticatorAttachment, of type AuthenticatorAttachment  
 2116 If this member is present, eligible authenticators are filtered  
 2117 to only authenticators attached with the specified 5.4.5  
 2118 Authenticator Attachment enumeration (enum

1948 AuthenticatorAttachment).

1949

1950 requireResidentKey, of type boolean, defaulting to false

1951 This member describes the Relying Parties' requirements

1952 regarding availability of the Client-side-resident Credential

1953 Private Key. If the parameter is set to true, the authenticator

1954 MUST create a Client-side-resident Credential Private Key when

1955 creating a public key credential.

1956

1957 userVerification, of type UserVerificationRequirement, defaulting to

1958 "preferred"

1959 This member describes the Relying Party's requirements regarding

1960 user verification for the create() operation. Eligible

1961 authenticators are filtered to only those capable of satisfying

1962 this requirement.

1963

1964 5.4.5. Authenticator Attachment enumeration (enum AuthenticatorAttachment)

1965

1966 enum AuthenticatorAttachment {

1967 "platform", // Platform attachment

1968 "cross-platform" // Cross-platform attachment

1969 };

1970

1971 Clients **may** communicate with authenticators using a variety of

1972 mechanisms. For example, a client **may** use a platform-specific API to

1973 communicate with an authenticator which is physically bound to a

1974 platform. On the other hand, a client **may** use a variety of standardized

1975 cross-platform transport protocols such as Bluetooth (see 5.8.4

1976 Authenticator Transport enumeration (enum AuthenticatorTransport)) to

1977 discover and communicate with cross-platform attached authenticators.

1978 Therefore, we use AuthenticatorAttachment to describe an

1979 authenticator's attachment modality. We define authenticators that are

1980 part of the client's platform as having a platform attachment, and

1981 refer to them as platform authenticators. While those that are

1982 reachable via cross-platform transport protocols are defined as having

1983 cross-platform attachment, and refer to them as roaming authenticators.

1984 \* platform attachment - the respective authenticator is attached

1985 using platform-specific transports. Usually, authenticators of this

1986 class are non-removable from the platform.

1987

1988 \* cross-platform attachment - the respective authenticator is

1989 attached using cross-platform transports. Authenticators of this

1990 class are removable from, and can "roam" among, client platforms.

1991

1992 This distinction is important because there are use-cases where only

1993 platform authenticators are acceptable to a Relying Party, and

1994 conversely ones where only roaming authenticators are employed. As a

1995 concrete example of the former, a credential **on a platform**

1996 **authenticator may be used by Relying Parties to quickly and**

1997 **conveniently reauthenticate the user with a minimum of friction, e.g.,**

1998 **the user will not have to dig around in their pocket for their key fob**

1999 **or phone. As a concrete example of the latter, when the user is**

2000 **accessing the Relying Party from a given client for the first time,**

2001 **they may be required to use a roaming authenticator which was**

originally registered with the Relying Party using a different client.

2002

2003 5.4.6. Attestation Conveyance Preference enumeration (enum

2119 AuthenticatorAttachment).

2120

2121 requireResidentKey, of type boolean, defaulting to false

2122 This member describes the Relying Parties' requirements

2123 regarding availability of the Client-side-resident Credential

2124 Private Key. If the parameter is set to true, the authenticator

2125 MUST create a Client-side-resident Credential Private Key when

2126 creating a public key credential.

2127

2128 userVerification, of type UserVerificationRequirement, defaulting to

2129 "preferred"

2130 This member describes the Relying Party's requirements regarding

2131 user verification for the create() operation. Eligible

2132 authenticators are filtered to only those capable of satisfying

2133 this requirement.

2134

2135 5.4.5. Authenticator Attachment enumeration (enum AuthenticatorAttachment)

2136

2137 enum AuthenticatorAttachment {

2138 "platform", // Platform attachment

2139 "cross-platform" // Cross-platform attachment

2140 };

2141

2142 Clients **can** communicate with authenticators using a variety of

2143 mechanisms. For example, a client **MAY** use a platform-specific API to

2144 communicate with an authenticator which is physically bound to a

2145 platform. On the other hand, a client **can** use a variety of standardized

2146 cross-platform transport protocols such as Bluetooth (see 5.10.4

2147 Authenticator Transport enumeration (enum AuthenticatorTransport)) to

2148 discover and communicate with cross-platform attached authenticators.

2149 Therefore, we use AuthenticatorAttachment to describe an

2150 authenticator's attachment modality. We define authenticators that are

2151 part of the client's platform as having a platform attachment, and

2152 refer to them as platform authenticators. While those that are

2153 reachable via cross-platform transport protocols are defined as having

2154 cross-platform attachment, and refer to them as roaming authenticators.

2155 \* platform attachment - the respective authenticator is attached

2156 using platform-specific transports. Usually, authenticators of this

2157 class are non-removable from the platform. **A public key credential**

2158 **bound to a platform authenticator is called a platform credential.**

2159 \* cross-platform attachment - the respective authenticator is

2160 attached using cross-platform transports. Authenticators of this

2161 class are removable from, and can "roam" among, client platforms. **A**

2162 **public key credential bound to a roaming authenticator is called a**

2163 **roaming credential.**

2164

2165 This distinction is important because there are use-cases where only

2166 platform authenticators are acceptable to a Relying Party, and

2167 conversely ones where only roaming authenticators are employed. As a

2168 concrete example of the former, a **platform credential may be used by**

2169 **Relying Parties to quickly and conveniently reauthenticate the user**

2170 **with a minimum of friction, e.g., the user will not have to dig around**

2171 **in their pocket for their key fob or phone. As a concrete example of**

2172 **the latter, when the user is accessing the Relying Party from a given**

2173 **client for the first time, they may be asked to use a roaming**

2174 **credential which was originally registered with the Relying Party using**

2175 **a different client.**

2176

2177 **Note: An attachment modality selection option is available only in the**

2178 **[[Create]](origin, options, sameOriginWithAncestors) operation. The**

2179 **Relying Party may use it to, for example, ensure the user has a roaming**

2180 **credential for authenticating using other clients; or to specifically**

2181 **register a platform credential for easier reauthentication using a**

2182 **particular client. The [[DiscoverFromExternalSource]](origin, options,**

2183 **sameOriginWithAncestors) operation has no attachment modality selection**

2184 **option, so the Relying Party should accept any of the user's registered**

2185 **credentials. The client and user will then use whichever is available**

2186 **and convenient at the time.**

2187

2188 5.4.6. Attestation Conveyance Preference enumeration (enum

```

2004 AttestationConveyancePreference)
2005
2006 Relying Parties may use AttestationConveyancePreference to specify
2007 their preference regarding attestation conveyance during credential
2008 generation.
2009 enum AttestationConveyancePreference {
2010     "none",
2011     "indirect",
2012     "direct"
2013 };
2014
2015 * none - indicates that the Relying Party is not interested in
2016 authenticator attestation. The client may replace the AAGUID and
2017 attestation statement generated by the authenticator with
2018 meaningless client-generated values. For example, in order to avoid
2019 having to obtain user consent to relay uniquely identifying
2020 information to the Relying Party, or to save a roundtrip to a
2021 Privacy CA.
2022 This is the default value.
2023 * indirect - indicates that the Relying Party prefers an attestation
2024 conveyance yielding verifiable attestation statements, but allows
2025 the client to decide how to obtain such attestation statements. The
2026 client may replace the authenticator-generated attestation
2027 statements with attestation statements generated by a Privacy CA,
2028 in order to protect the user's privacy, or to assist Relying
2029 Parties with attestation verification in a heterogeneous ecosystem.
2030
2031 Note: There is no guarantee that the Relying Party will obtain a
2032 verifiable attestation statement in this case. For example, in the
2033 case that the authenticator employs self attestation.
2034 * direct - indicates that the Relying Party wants to receive the
2035 attestation statement as generated by the authenticator.
2036
2037 5.5. Options for Assertion Generation (dictionary
2038 PublicKeyCredentialRequestOptions)
2039
2040 The PublicKeyCredentialRequestOptions dictionary supplies get() with
2041 the data it needs to generate an assertion. Its challenge member must
2042 be present, while its other members are optional.
2043 dictionary PublicKeyCredentialRequestOptions {
2044     required BufferSource challenge;
2045     unsigned long timeout;
2046     USVString rpId;
2047     sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
2048     UserVerificationRequirement userVerification = "preferred";
2049     AuthenticationExtensions extensions;
2050 };
2051
2052 challenge, of type BufferSource
2053 This member represents a challenge that the selected
2054 authenticator signs, along with other data, when producing an
2055 authentication assertion. See the 13.1 Cryptographic Challenges
2056 security consideration.
2057
2058 timeout, of type unsigned long
2059 This optional member specifies a time, in milliseconds, that the
2060 caller is willing to wait for the call to complete. The value is
2061 treated as a hint, and may be overridden by the platform.
2062
2063 rpId, of type USVString
2064 This optional member specifies the relying party identifier
2065 claimed by the caller. If omitted, its value will be the
2066 CredentialsContainer object's relevant settings object's
2067 origin's effective domain.
2068
2069 allowCredentials, of type sequence<PublicKeyCredentialDescriptor>,
2070 defaulting to None
2071 This optional member contains a list of
2072 PublicKeyCredentialDescriptor objects representing public key
2073 credentials acceptable to the caller, in descending order of the
    
```

```

2189 AttestationConveyancePreference)
2190
2191 Relying Parties may use AttestationConveyancePreference to specify
2192 their preference regarding attestation conveyance during credential
2193 generation.
2194 enum AttestationConveyancePreference {
2195     "none",
2196     "indirect",
2197     "direct"
2198 };
2199
2200 * none - indicates that the Relying Party is not interested in
2201 authenticator attestation. For example, in order to potentially
2202 avoid having to obtain user consent to relay identifying
2203 information to the Relying Party, or to save a roundtrip to an
2204 Attestation CA.
2205
2206 This is the default value.
2207 * indirect - indicates that the Relying Party prefers an attestation
2208 conveyance yielding verifiable attestation statements, but allows
2209 the client to decide how to obtain such attestation statements. The
2210 client MAY replace the authenticator-generated attestation
2211 statements with attestation statements generated by an
2212 Anonymization CA, in order to protect the user's privacy, or to
2213 assist Relying Parties with attestation verification in a
2214 heterogeneous ecosystem.
2215 Note: There is no guarantee that the Relying Party will obtain a
2216 verifiable attestation statement in this case. For example, in the
2217 case that the authenticator employs self attestation.
2218 * direct - indicates that the Relying Party wants to receive the
2219 attestation statement as generated by the authenticator.
2220
2221 5.5. Options for Assertion Generation (dictionary
2222 PublicKeyCredentialRequestOptions)
2223
2224 The PublicKeyCredentialRequestOptions dictionary supplies get() with
2225 the data it needs to generate an assertion. Its challenge member MUST
2226 be present, while its other members are OPTIONAL.
2227 dictionary PublicKeyCredentialRequestOptions {
2228     required BufferSource challenge;
2229     unsigned long timeout;
2230     USVString rpId;
2231     sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
2232     UserVerificationRequirement userVerification = "preferred";
2233     AuthenticationExtensionsClientInputs extensions;
2234 };
2235
2236 challenge, of type BufferSource
2237 This member represents a challenge that the selected
2238 authenticator signs, along with other data, when producing an
2239 authentication assertion. See the 13.1 Cryptographic Challenges
2240 security consideration.
2241
2242 timeout, of type unsigned long
2243 This OPTIONAL member specifies a time, in milliseconds, that the
2244 caller is willing to wait for the call to complete. The value is
2245 treated as a hint, and MAY be overridden by the platform.
2246
2247 rpId, of type USVString
2248 This optional member specifies the relying party identifier
2249 claimed by the caller. If omitted, its value will be the
2250 CredentialsContainer object's relevant settings object's
2251 origin's effective domain.
2252
2253 allowCredentials, of type sequence<PublicKeyCredentialDescriptor>,
2254 defaulting to None
2255 This optional member contains a list of
2256 PublicKeyCredentialDescriptor objects representing public key
2257 credentials acceptable to the caller, in descending order of the
    
```

2073 caller's preference (the first item in the list is the most  
 2074 preferred credential, and so on down the list).  
 2075  
 2076 userVerification, of type UserVerificationRequirement, defaulting to  
 2077 "preferred"  
 2078 This member describes the Relying Party's requirements regarding  
 2079 user verification for the get() operation. Eligible  
 2080 authenticators are filtered to only those capable of satisfying  
 2081 this requirement.  
 2082  
 2083 extensions, of type AuthenticationExtensions  
 2084 This optional member contains additional parameters requesting  
 2085 additional processing by the client and authenticator. For  
 2086 example, if transaction confirmation is sought from the user,  
 2087 then the prompt string might be included as an extension.  
 2088  
 2089 5.6. Abort operations with AbortSignal  
 2090  
 2091 Developers are encouraged to leverage the AbortController to manage the  
 2092 [[Create]](origin, options, sameOriginWithAncestors) and  
 2093 [[DiscoverFromExternalSource]](origin, options,  
 2094 sameOriginWithAncestors) operations. See DOM 3.3 Using AbortController  
 2095 and AbortSignal objects in APIs section for detailed instructions.  
 2096  
 2097 Note: DOM 3.3 Using AbortController and AbortSignal objects in APIs  
 2098 section specifies that web platform APIs integrating with the  
 2099 AbortController must reject the promise immediately once the aborted  
 2100 flag is set. Given the complex inheritance and parallelization  
 2101 structure of the [[Create]](origin, options, sameOriginWithAncestors)  
 2102 and [[DiscoverFromExternalSource]](origin, options,  
 2103 sameOriginWithAncestors) methods, the algorithms for the two APIs  
 2104 fulfills this requirement by checking the aborted flag in three places.  
 2105 In the case of [[Create]](origin, options, sameOriginWithAncestors),  
 2106 the aborted flag is checked first in Credential Management 1 2.5.4  
 2107 Create a Credential immediately before calling [[Create]](origin,  
 2108 options, sameOriginWithAncestors), then in 5.1.3 Create a new  
 2109 credential - PublicKeyCredential's [[Create]](origin, options,  
 2110 sameOriginWithAncestors) method right before authenticator sessions  
 2111 start, and finally during authenticator sessions. The same goes for  
 2112 [[DiscoverFromExternalSource]](origin, options,  
 2113 sameOriginWithAncestors).  
 2114  
 2115 The visibility and focus state of the Window object determines whether  
 2116 the [[Create]](origin, options, sameOriginWithAncestors) and  
 2117 [[DiscoverFromExternalSource]](origin, options,  
 2118 sameOriginWithAncestors) operations should continue. When the Window  
 2119 object associated with the [Document] loses focus, [[Create]](origin,  
 2120 options, sameOriginWithAncestors) and  
 2121 [[DiscoverFromExternalSource]](origin, options,  
 2122 sameOriginWithAncestors) operations SHOULD be aborted.  
 2123  
 2124 The WHATWG HTML WG is discussing whether to provide a hook when a  
 2125 browsing context gains or loses focus. If a hook is provided, the  
 2126 above paragraph will be updated to include the hook. See WHATWG HTML WG  
 2127 Issue #2711 for more details.  
 2128  
 2129 5.7. Authentication Extensions (typedef AuthenticationExtensions)  
 2130  
 2131 typedef record<DOMString, any> AuthenticationExtensions;  
 2132  
 2133 This is a dictionary containing zero or more WebAuthn extensions, as  
 2134 defined in 9 WebAuthn Extensions. An AuthenticationExtensions instance  
 2135 can contain either client extensions or authenticator extensions,  
 2136 depending upon context.  
 2137  
 2138 5.8. Supporting Data Structures

2257 caller's preference (the first item in the list is the most  
 2258 preferred credential, and so on down the list).  
 2259  
 2260 userVerification, of type UserVerificationRequirement, defaulting to  
 2261 "preferred"  
 2262 This member describes the Relying Party's requirements regarding  
 2263 user verification for the get() operation. Eligible  
 2264 authenticators are filtered to only those capable of satisfying  
 2265 this requirement.  
 2266  
 2267 extensions, of type AuthenticationExtensionsClientInputs  
 2268 This OPTIONAL member contains additional parameters requesting  
 2269 additional processing by the client and authenticator. For  
 2270 example, if transaction confirmation is sought from the user,  
 2271 then the prompt string might be included as an extension.  
 2272  
 2273 5.6. Abort operations with AbortSignal  
 2274  
 2275 Developers are encouraged to leverage the AbortController to manage the  
 2276 [[Create]](origin, options, sameOriginWithAncestors) and  
 2277 [[DiscoverFromExternalSource]](origin, options,  
 2278 sameOriginWithAncestors) operations. See DOM 3.3 Using AbortController  
 2279 and AbortSignal objects in APIs section for detailed instructions.  
 2280  
 2281 Note: DOM 3.3 Using AbortController and AbortSignal objects in APIs  
 2282 section specifies that web platform APIs integrating with the  
 2283 AbortController must reject the promise immediately once the aborted  
 2284 flag is set. Given the complex inheritance and parallelization  
 2285 structure of the [[Create]](origin, options, sameOriginWithAncestors)  
 2286 and [[DiscoverFromExternalSource]](origin, options,  
 2287 sameOriginWithAncestors) methods, the algorithms for the two APIs  
 2288 fulfills this requirement by checking the aborted flag in three places.  
 2289 In the case of [[Create]](origin, options, sameOriginWithAncestors),  
 2290 the aborted flag is checked first in Credential Management 1 2.5.4  
 2291 Create a Credential immediately before calling [[Create]](origin,  
 2292 options, sameOriginWithAncestors), then in 5.1.3 Create a new  
 2293 credential - PublicKeyCredential's [[Create]](origin, options,  
 2294 sameOriginWithAncestors) method right before authenticator sessions  
 2295 start, and finally during authenticator sessions. The same goes for  
 2296 [[DiscoverFromExternalSource]](origin, options,  
 2297 sameOriginWithAncestors).  
 2298  
 2299 The visibility and focus state of the Window object determines whether  
 2300 the [[Create]](origin, options, sameOriginWithAncestors) and  
 2301 [[DiscoverFromExternalSource]](origin, options,  
 2302 sameOriginWithAncestors) operations should continue. When the Window  
 2303 object associated with the [Document] loses focus, [[Create]](origin,  
 2304 options, sameOriginWithAncestors) and  
 2305 [[DiscoverFromExternalSource]](origin, options,  
 2306 sameOriginWithAncestors) operations SHOULD be aborted.  
 2307  
 2308 The WHATWG HTML WG is discussing whether to provide a hook when a  
 2309 browsing context gains or loses focus. If a hook is provided, the  
 2310 above paragraph will be updated to include the hook. See WHATWG HTML WG  
 2311 Issue #2711 for more details.  
 2312  
 2313 5.7. Authentication Extensions Client Inputs (typedef  
 2314 AuthenticationExtensionsClientInputs)  
 2315  
 2316 dictionary AuthenticationExtensionsClientInputs {  
 2317 };  
 2318  
 2319 This is a dictionary containing the client extension input values for  
 2320 zero or more WebAuthn extensions, as defined in 9 WebAuthn Extensions.  
 2321  
 2322 5.8. Authentication Extensions Client Outputs (typedef  
 2323 AuthenticationExtensionsClientOutputs)  
 2324

2139  
 2140  
 2141 The public key credential type uses certain data structures that are  
 2142 specified in supporting specifications. These are as follows.  
 2143  
 2144 **5.8.1. Client data used in WebAuthn signatures (dictionary**  
 2145 **CollectedClientData)**  
 2146  
 2147 The client data represents the contextual bindings of both the Relying  
 2148 Party and the client platform. It is a key-value mapping with  
 2149 string-valued keys. Values **may** be any type that has a valid encoding in  
 2150 JSON. Its structure is defined by the following Web IDL.  
 2151 dictionary CollectedClientData {  
 2152 required DOMString type;  
 2153 required DOMString challenge;  
 2154 required DOMString origin;  
 2155 required DOMString hashAlgorithm;  
 2156 DOMString tokenBindingId;  
 2157 AuthenticationExtensions clientExtensions;  
 2158 AuthenticationExtensions authenticatorExtensions;  
 2159 };  
 2160  
 2161 The type member contains the string "webauthn.create" when creating new  
 2162 credentials, and "webauthn.get" when getting an assertion from an  
 2163 existing credential. The purpose of this member is to prevent certain  
 2164 types of signature confusion attacks (where an attacker substitutes one  
 2165 legitimate signature for another).  
 2166  
 2167 The challenge member contains the base64url encoding of the challenge  
 2168 provided by the RP. See the 13.1 Cryptographic Challenges security  
 2169 consideration.  
 2170  
 2171 The origin member contains the fully qualified origin of the requester,  
 2172 as provided to the authenticator by the client, in the syntax defined  
 2173 by [RFC6454].  
 2174  
 2175 The hashAlgorithm member is a recognized algorithm name that supports  
 2176 the "digest" operation, which specifies the algorithm used to compute  
 2177 the hash of the serialized client data. This algorithm is chosen by the  
 2178 client at its sole discretion.  
 2179  
 2180 The tokenBindingId member contains the base64url encoding of the Token  
 2181 Binding ID that this client uses for the Token Binding protocol when  
 2182 communicating with the Relying Party. This can be omitted if no Token  
 2183 Binding has been negotiated between the client and the Relying Party.  
 2184  
 2185 The optional clientExtensions and authenticatorExtensions members  
 2186 contain additional parameters generated by processing the extensions  
 2187 passed in by the Relying Party. WebAuthn extensions are detailed in  
 Section 9 WebAuthn Extensions.

2325 dictionary AuthenticationExtensionsClientOutputs {  
 2326 };  
 2327  
 2328 This is a dictionary containing the client extension output values for  
 2329 zero or more WebAuthn extensions, as defined in 9 WebAuthn Extensions.  
 2330  
 2331 **5.9. Authentication Extensions Authenticator Inputs (typedef**  
 2332 **AuthenticationExtensionsAuthenticatorInputs)**  
 2333  
 2334 typedef record<DOMString, DOMString> AuthenticationExtensionsAuthenticatorInputs  
 2335 ;  
 2336  
 2337 This is a dictionary containing the authenticator extension input  
 2338 values for zero or more WebAuthn extensions, as defined in 9 WebAuthn  
 2339 Extensions.  
 2340  
 2341 **5.10. Supporting Data Structures**  
 2342  
 2343 The public key credential type uses certain data structures that are  
 2344 specified in supporting specifications. These are as follows.  
 2345  
 2346 **5.10.1. Client data used in WebAuthn signatures (dictionary**  
 2347 **CollectedClientData)**  
 2348  
 2349 The client data represents the contextual bindings of both the Relying  
 2350 Party and the client platform. It is a key-value mapping with  
 2351 string-valued keys. Values **can** be any type that has a valid encoding in  
 2352 JSON. Its structure is defined by the following Web IDL.  
 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  
 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.

218E This structure is used by the client to compute the following  
 218F quantities:  
 219C  
 2191 JSON-serialized client data  
 2192 This is the UTF-8 encoding of the result of calling the initial  
 2193 value of JSON.stringify on a CollectedClientData dictionary.  
 2194  
 2195 Hash of the serialized client data  
 2196 This is the hash (computed using [hashAlgorithm](#)) of the  
 2197 JSON-serialized client data, as constructed by the client.  
 2198  
 2199  
 2200 5.8.2. Credential Type enumeration (enum PublicKeyCredentialType)  
 2201  
 2202 enum PublicKeyCredentialType {  
 2203 "public-key"  
 2204 };  
 2205  
 2206 This enumeration defines the valid credential types. It is an extension  
 2207 point; values [may](#) be added to it in the future, as more credential  
 2208 types are defined. The values of this enumeration are used for  
 2209 versioning the Authentication Assertion and attestation structures  
 2210 according to the type of the authenticator.  
 2211  
 2212 Currently one credential type is defined, namely "public-key".  
 2213  
 2214 5.8.3. Credential Descriptor (dictionary PublicKeyCredentialDescriptor)  
 2215  
 2216 dictionary PublicKeyCredentialDescriptor {  
 2217 required PublicKeyCredentialType type;  
 2218 required BufferSource id;  
 2219 sequence<AuthenticatorTransport> transports;  
 2220 };  
 2221  
 2222 This dictionary contains the attributes that are specified by a caller  
 2223 when referring to a credential as an input parameter to the [create\(\)](#) or  
 2224 [get\(\)](#) methods. It mirrors the fields of the [PublicKeyCredential object](#)  
 2225 returned by the latter methods.  
 2226  
 2227 The type member contains the type of the credential the [caller is](#)  
 2228 referring to.  
 2229  
 2230 The id member contains the [identifier](#) of the credential [that the caller](#)  
 2231 is referring to.  
 2232  
 2233 5.8.4. Authenticator Transport enumeration (enum AuthenticatorTransport)  
 2234  
 2235 enum AuthenticatorTransport {  
 2236 "usb",  
 2237 "nfc",  
 2238 "ble"  
 2239 };  
 2240  
 2241 Authenticators may communicate with [Clients](#) using a variety of  
 2242 transports. This enumeration defines a hint as to how [Clients](#) might  
 2243 communicate with a particular [Authenticator](#) in order to obtain an  
 2244 assertion for a specific credential. Note that these hints represent  
 2245 the Relying Party's best belief as to how an [Authenticator](#) may be  
 2246 reached. A Relying Party may obtain a list of transports hints from  
 2247 some attestation statement formats or via some out-of-band mechanism;  
 2248 it is outside the scope of this specification to define that mechanism.  
 2249 \* usb - the respective [Authenticator](#) [may](#) be contacted over USB.  
 2250 \* nfc - the respective [Authenticator](#) [may](#) be contacted over Near Field  
 2251 Communication (NFC).  
 2252 \* ble - the respective [Authenticator](#) [may](#) be contacted over Bluetooth  
 2253 Smart (Bluetooth Low Energy / BLE).  
 2254  
 2255 5.8.5. Cryptographic Algorithm Identifier (typedef COSEAlgorithmIdentifier)  
 2256  
 2257 typedef long COSEAlgorithmIdentifier;

239C This structure is used by the client to compute the following  
 239D quantities:  
 239E  
 239F JSON-serialized client data  
 239G This is the UTF-8 encoding of the result of calling the initial  
 239H value of JSON.stringify on a CollectedClientData dictionary.  
 239I  
 239J Hash of the serialized client data  
 239K This is the hash (computed using [SHA-256](#)) of the [JSON-serialized](#)  
 239L client data, as constructed by the client.  
 2400  
 2401  
 2402 5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)  
 2403  
 2404 enum PublicKeyCredentialType {  
 2405 "public-key"  
 2406 };  
 2407  
 2408 This enumeration defines the valid credential types. It is an extension  
 2409 point; values [can](#) be added to it in the future, as more credential  
 2410 types are defined. The values of this enumeration are used for  
 2411 versioning the Authentication Assertion and attestation structures  
 2412 according to the type of the authenticator.  
 2413  
 2414 Currently one credential type is defined, namely "public-key".  
 2415  
 2416 5.10.3. Credential Descriptor (dictionary PublicKeyCredentialDescriptor)  
 2417  
 2418 dictionary PublicKeyCredentialDescriptor {  
 2419 required PublicKeyCredentialType type;  
 2420 required BufferSource id;  
 2421 sequence<AuthenticatorTransport> transports;  
 2422 };  
 2423  
 2424 This dictionary contains the attributes that are specified by a caller  
 2425 when referring to a [public key](#) credential as an input parameter to the  
 2426 [create\(\)](#) or [get\(\)](#) methods. It mirrors the fields of the  
 2427 [PublicKeyCredential object](#) returned by the latter methods.  
 2428  
 2429 The type member contains the type of the [public key](#) credential the  
 2430 [caller is](#) referring to.  
 2431  
 2432 The id member contains the [credential ID](#) of the [public key](#) credential  
 2433 [that the caller](#) is referring to.  
 2434  
 2435 5.10.4. Authenticator Transport enumeration (enum AuthenticatorTransport)  
 2436  
 2437 enum AuthenticatorTransport {  
 2438 "usb",  
 2439 "nfc",  
 2440 "ble"  
 2441 };  
 2442  
 2443 Authenticators may communicate with [clients](#) using a variety of  
 2444 transports. This enumeration defines a hint as to how [clients](#) might  
 2445 communicate with a particular [authenticator](#) in order to obtain an  
 2446 assertion for a specific credential. Note that these hints represent  
 2447 the Relying Party's best belief as to how an [authenticator](#) may be  
 2448 reached. A Relying Party may obtain a list of transports hints from  
 2449 some attestation statement formats or via some out-of-band mechanism;  
 2450 it is outside the scope of this specification to define that mechanism.  
 2451 \* usb - the respective [authenticator](#) [can](#) be contacted over USB.  
 2452 \* nfc - the respective [authenticator](#) [can](#) be contacted over Near Field  
 2453 Communication (NFC).  
 2454 \* ble - the respective [authenticator](#) [can](#) be contacted over Bluetooth  
 2455 Smart (Bluetooth Low Energy / BLE).  
 2456  
 2457 5.10.5. Cryptographic Algorithm Identifier (typedef COSEAlgorithmIdentifier)  
 2458  
 2459 typedef long COSEAlgorithmIdentifier;

225E A COSEAlgorithmIdentifier's value is a number identifying a  
 225F cryptographic algorithm. The algorithm identifiers SHOULD be values  
 2260 registered in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG],  
 2261 for instance, -7 for "ES256" and -257 for "RS256".  
 2262  
 2263

#### 2264 5.8.6. User Verification Requirement enumeration (enum 2265 UserVerificationRequirement)

```
2266 enum UserVerificationRequirement {  

  2267 "required",  

  2268 "preferred",  

  2269 "discouraged"  

  2270 };  

  2271
```

2272 A Relying Party may require user verification for some of its  
 2273 operations but not for others, and may use this type to express its  
 2274 needs.  
 2275

2276 The value required indicates that the Relying Party requires user  
 2277 verification for the operation and will fail the operation if the  
 2278 response does not have the UV flag set.

2281 The value preferred indicates that the Relying Party prefers user  
 2282 verification for the operation if possible, but will not fail the  
 2283 operation if the response does not have the UV flag set.  
 2284

2285 The value discouraged indicates that the Relying Party does not want  
 2286 user verification employed during the operation (e.g., in the interest  
 2287 of minimizing disruption to the user interaction flow).  
 2288

### 2289 6. WebAuthn Authenticator model

2290 The API defined in this specification implies a specific abstract  
 2291 functional model for an authenticator. This section describes the  
 2292 authenticator model.  
 2293

2294 Client platforms **may** implement and expose this abstract model in any  
 2295 way desired. However, the behavior of the client's Web Authentication  
 2296 API implementation, when operating on the authenticators supported by  
 2297 that platform, **MUST** be indistinguishable from the behavior specified in  
 2298 5 Web Authentication API.  
 2299

2300 For authenticators, this model defines the logical operations that they  
 2301 **must** support, and the data formats that they expose to the client and  
 2302 the Relying Party. However, it does not define the details of how  
 2303 authenticators communicate with the client platform, unless they are  
 2304 **required** for interoperability with Relying Parties. For instance, this  
 2305 abstract model does not define protocols for connecting authenticators  
 2306 to clients over transports such as USB or NFC. Similarly, this abstract  
 2307 model does not define specific error codes or methods of returning  
 2308 them; however, it does define error behavior in terms of the needs of  
 2309 the client. Therefore, specific error codes are mentioned as a means of  
 2310 showing which error conditions must be distinguishable (or not) from  
 2311 each other in order to enable a compliant and secure client  
 2312 implementation.  
 2313  
 2314

2315 In this abstract model, the authenticator provides key management and  
 2316 cryptographic signatures. It **may** be embedded in the WebAuthn client, or  
 2317 housed in a separate device entirely. The authenticator **may** itself  
 2318 contain a cryptographic module which operates at a higher security  
 2319 level than the rest of the authenticator. This is particularly

2460 A COSEAlgorithmIdentifier's value is a number identifying a  
 2461 cryptographic algorithm. The algorithm identifiers SHOULD be values  
 2462 registered in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG],  
 2463 for instance, -7 for "ES256" and -257 for "RS256".  
 2464  
 2465

#### 2466 5.10.6. User Verification Requirement enumeration (enum 2467 UserVerificationRequirement)

```
2468 enum UserVerificationRequirement {  

  2469 "required",  

  2470 "preferred",  

  2471 "discouraged"  

  2472 };  

  2473
```

2474 A Relying Party may require user verification for some of its  
 2475 operations but not for others, and may use this type to express its  
 2476 needs.  
 2477

2478 The value required indicates that the Relying Party requires user  
 2479 verification for the operation and will fail the operation if the  
 2480 response does not have the UV flag set.

2481 The value preferred indicates that the Relying Party prefers user  
 2482 verification for the operation if possible, but will not fail the  
 2483 operation if the response does not have the UV flag set.  
 2484

2485 The value discouraged indicates that the Relying Party does not want  
 2486 user verification employed during the operation (e.g., in the interest  
 2487 of minimizing disruption to the user interaction flow).  
 2488

### 2489 6. WebAuthn Authenticator Model

2490 The Web Authentication API implies a specific abstract functional model  
 2491 for an authenticator. This section describes that authenticator model.  
 2492

2493 Client platforms **MAY** implement and expose this abstract model in any  
 2494 way desired. However, the behavior of the client's Web Authentication  
 2495 API implementation, when operating on the authenticators supported by  
 2496 that platform, **MUST** be indistinguishable from the behavior specified in  
 2497 5 Web Authentication API.  
 2498

2499 For authenticators, this model defines the logical operations that they  
 2500 **MUST** support, and the data formats that they expose to the client and  
 2501 the Relying Party. However, it does not define the details of how  
 2502 authenticators communicate with the client platform, unless they are  
 2503 **necessary** for interoperability with Relying Parties. For instance, this  
 2504 abstract model does not define protocols for connecting authenticators  
 2505 to clients over transports such as USB or NFC. Similarly, this abstract  
 2506 model does not define specific error codes or methods of returning  
 2507 them; however, it does define error behavior in terms of the needs of  
 2508 the client. Therefore, specific error codes are mentioned as a means of  
 2509 showing which error conditions must be distinguishable (or not) from  
 2510 each other in order to enable a compliant and secure client  
 2511 implementation.  
 2512  
 2513

2514 Relying Parties may influence authenticator selection, if they deem  
 2515 necessary, by stipulating various authenticator characteristics when  
 2516 creating credentials and/or when generating assertions, through use of  
 2517 credential creation options or assertion generation options,  
 2518 respectively. The algorithms underlying the WebAuthn API marshal these  
 2519 options and pass them to the applicable authenticator operations  
 2520 defined below.  
 2521

2522 In this abstract model, the authenticator provides key management and  
 2523 cryptographic signatures. It **can** be embedded in the WebAuthn client or  
 2524 housed in a separate device entirely. The authenticator itself **can**  
 2525 contain a cryptographic module which operates at a higher security  
 2526 level than the rest of the authenticator. This is particularly

2320 important for authenticators that are embedded in the WebAuthn client,  
2321 as in those cases this cryptographic module (which may, for example, be  
2322 a TPM) could be considered more trustworthy than the rest of the  
2323 authenticator.  
2324

2325 Each authenticator stores some number of public key credentials. Each  
2326 public key credential has an identifier which is unique (or extremely  
2327 unlikely to be duplicated) among all public key credentials. Each  
2328 credential is also associated with a Relying Party, whose identity is  
2329 represented by a Relying Party Identifier (RP ID).  
2330

2331 Each authenticator has an AAGUID, which is a 128-bit identifier that  
2332 indicates the type (e.g. make and model) of the authenticator. The  
2333 AAGUID MUST be chosen by the manufacturer to be identical across all  
2334 substantially identical authenticators made by that manufacturer, and  
2335 different (with probability  $1-2^{-128}$  or greater) from the AAGUIDs of  
2336 all other types of authenticators. The RP MAY use the AAGUID to infer  
2337 certain properties of the authenticator, such as certification level  
2338 and strength of key protection, using information from other sources.  
2339

2340 The primary function of the authenticator is to provide WebAuthn  
2341 signatures, which are bound to various contextual data. These data are  
2342 observed, and added at different levels of the stack as a signature  
2343 request passes from the server to the authenticator. In verifying a  
2344 signature, the server checks these bindings against expected values.  
2345 These contextual bindings are divided in two: Those added by the RP or  
2346 the client, referred to as client data; and those added by the  
2347 authenticator, referred to as the authenticator data. The authenticator  
2348 signs over the client data, but is otherwise not interested in its  
2349 contents. To save bandwidth and processing requirements on the  
2350 authenticator, the client hashes the client data and sends only the  
2351 result to the authenticator. The authenticator signs over the  
2352 combination of the hash of the serialized client data, and its own  
2353 authenticator data.  
2354

- 2355 The goals of this design can be summarized as follows.
- 2356 \* The scheme for generating signatures should accommodate cases where  
2357 the link between the client platform and authenticator is very  
2358 limited, in bandwidth and/or latency. Examples include Bluetooth  
2359 Low Energy and Near-Field Communication.
  - 2360 \* The data processed by the authenticator should be small and easy to  
2361 interpret in low-level code. In particular, authenticators should  
2362 not have to parse high-level encodings such as JSON.
  - 2363 \* Both the client platform and the authenticator should have the  
2364 flexibility to add contextual bindings as needed.
  - 2365 \* The design aims to reuse as much as possible of existing encoding  
2366 formats in order to aid adoption and implementation.

2367 Authenticators produce cryptographic signatures for two distinct  
2368 purposes:

- 2370 1. An attestation signature is produced when a new public key  
2371 credential is created via an authenticatorMakeCredential operation.  
2372 An attestation signature provides cryptographic proof of certain  
2373 properties of the the authenticator and the credential. For  
2374 instance, an attestation signature asserts the authenticator type  
2375 (as denoted by its AAGUID) and the credential public key. The  
2376 attestation signature is signed by an attestation private key,  
2377 which is chosen depending on the type of attestation desired. For  
2378 more details on attestation, see 6.3 Attestation.
- 2380 2. An assertion signature is produced when the  
2381 authenticatorGetAssertion method is invoked. It represents an  
2382 assertion by the authenticator that the user has consented to a  
2383 specific transaction, such as logging in, or completing a purchase.  
2384 Thus, an assertion signature asserts that the authenticator  
2385 possessing a particular credential private key has established, to  
2386 the best of its ability, that the user requesting this transaction  
2387 is the same user who consented to creating that particular public  
2388 key credential. It also asserts additional information, termed  
2389 client data, that may be useful to the caller, such as the means by

2529 important for authenticators that are embedded in the WebAuthn client,  
2530 as in those cases this cryptographic module (which may, for example, be  
2531 a TPM) could be considered more trustworthy than the rest of the  
2532 authenticator.  
2533

2534 Each authenticator stores a credentials map, a map from (rpId,  
2535 [userHandle]) to public key credential source.

2536 Additionally, each authenticator has an AAGUID, which is a 128-bit  
2537 identifier indicating the type (e.g. make and model) of the  
2538 authenticator. The AAGUID MUST be chosen by the manufacturer to be  
2539 identical across all substantially identical authenticators made by  
2540 that manufacturer, and different (with probability  $1-2^{-128}$  or greater)  
2541 from the AAGUIDs of all other types of authenticators. The RP MAY use  
2542 the AAGUID to infer certain properties of the authenticator, such as  
2543 certification level and strength of key protection, using information  
2544 from other sources.  
2545

2546 The primary function of the authenticator is to provide WebAuthn  
2547 signatures, which are bound to various contextual data. These data are  
2548 observed and added at different levels of the stack as a signature  
2549 request passes from the server to the authenticator. In verifying a  
2550 signature, the server checks these bindings against expected values.  
2551 These contextual bindings are divided in two: Those added by the RP or  
2552 the client, referred to as client data; and those added by the  
2553 authenticator, referred to as the authenticator data. The authenticator  
2554 signs over the client data, but is otherwise not interested in its  
2555 contents. To save bandwidth and processing requirements on the  
2556 authenticator, the client hashes the client data and sends only the  
2557 result to the authenticator. The authenticator signs over the  
2558 combination of the hash of the serialized client data, and its own  
2559 authenticator data.  
2560

- 2561 The goals of this design can be summarized as follows.
- 2562 \* The scheme for generating signatures should accommodate cases where  
2563 the link between the client platform and authenticator is very  
2564 limited, in bandwidth and/or latency. Examples include Bluetooth  
2565 Low Energy and Near-Field Communication.
  - 2566 \* The data processed by the authenticator should be small and easy to  
2567 interpret in low-level code. In particular, authenticators should  
2568 not have to parse high-level encodings such as JSON.
  - 2569 \* Both the client platform and the authenticator should have the  
2570 flexibility to add contextual bindings as needed.
  - 2571 \* The design aims to reuse as much as possible of existing encoding  
2572 formats in order to aid adoption and implementation.

2573 Authenticators produce cryptographic signatures for two distinct  
2574 purposes:

- 2576 1. An attestation signature is produced when a new public key  
2577 credential is created via an authenticatorMakeCredential operation.  
2578 An attestation signature provides cryptographic proof of certain  
2579 properties of the authenticator and the credential. For instance,  
2580 an attestation signature asserts the authenticator type (as denoted  
2581 by its AAGUID) and the credential public key. The attestation  
2582 signature is signed by an attestation private key, which is chosen  
2583 depending on the type of attestation desired. For more details on  
2584 attestation, see 6.3 Attestation.
- 2586 2. An assertion signature is produced when the  
2587 authenticatorGetAssertion method is invoked. It represents an  
2588 assertion by the authenticator that the user has consented to a  
2589 specific transaction, such as logging in, or completing a purchase.  
2590 Thus, an assertion signature asserts that the authenticator  
2591 possessing a particular credential private key has established, to  
2592 the best of its ability, that the user requesting this transaction  
2593 is the same user who consented to creating that particular public  
2594 key credential. It also asserts additional information, termed  
2595 client data, that may be useful to the caller, such as the means by

238E which user consent was provided, and the prompt shown to the user  
 239C by the authenticator. The assertion signature format is illustrated  
 2391 in Figure 2, below.  
 2392  
 2393 The formats of these signatures, as well as the procedures for  
 2394 generating them, are specified below.  
 2395  
 2396 6.1. Authenticator data  
 2397  
 2398 The authenticator data structure encodes contextual bindings made by  
 2399 the authenticator. These bindings are controlled by the authenticator  
 2400 itself, and derive their trust from the Relying Party's assessment of  
 2401 the security properties of the authenticator. In one extreme case, the  
 2402 authenticator may be embedded in the client, and its bindings may be no  
 2403 more trustworthy than the client data. At the other extreme, the  
 2404 authenticator may be a discrete entity with high-security hardware and  
 2405 software, connected to the client over a secure channel. In both cases,  
 2406 the Relying Party receives the authenticator data in the same format,  
 2407 and uses its knowledge of the authenticator to make trust decisions.  
 2408  
 2409 The authenticator data has a compact but extensible encoding. This is  
 2410 desired since authenticators can be devices with limited capabilities  
 2411 and low power requirements, with much simpler software stacks than the  
 2412 client platform components.  
 2413  
 2414 The authenticator data structure is a byte array of 37 bytes or more,  
 2415 as follows.  
 2416  
 2417 Name Length (in bytes) Description  
 2418 rpIdHash 32 SHA-256 hash of the RP ID associated with the credential.  
 2419 flags 1 Flags (bit 0 is the least significant bit):  
 2420 \* Bit 0: User Present (UP) result.  
 2421 + 1 means the user is present.  
 2422 + 0 means the user is not present.  
 2423 \* Bit 1: Reserved for future use (RFU1).  
 2424 \* Bit 2: User Verified (UV) result.  
 2425 + 1 means the user is verified.  
 2426 + 0 means the user is not verified.  
 2427 \* Bits 3-5: Reserved for future use (RFU2).  
 2428 \* Bit 6: Attested credential data included (AT).  
 2429 + Indicates whether the authenticator added attested credential  
 2430 data.  
 2431 \* Bit 7: Extension data included (ED).  
 2432 + Indicates if the authenticator data has extensions.  
 2433  
 2434 signCount 4 Signature counter, 32-bit unsigned big-endian integer.  
 2435 attestedCredentialData variable (if present) attested credential data  
 2436 (if present). See 6.3.1 Attested credential data for details. Its  
 2437 length depends on the length of the credential ID and credential public  
 2438 key being attested.  
 2439 extensions variable (if present) Extension-defined authenticator data.  
 2440 This is a CBOR [RFC7049] map with extension identifiers as keys, and  
 2441 authenticator extension outputs as values. See 9 WebAuthn Extensions  
 2442 for details.  
 2443  
 2444 NOTE: The names in the Name column in the above table are only for  
 2445 reference within this document, and are not present in the actual  
 2446 representation of the authenticator data.  
 2447  
 2448 The RP ID is originally received from the client when the credential is  
 2449 created, and again when an assertion is generated. However, it differs  
 2450 from other client data in some important ways. First, unlike the client  
 2451 data, the RP ID of a credential does not change between operations but  
 2452 instead remains the same for the lifetime of that credential. Secondly,  
 2453 it is validated by the authenticator during the  
 2454 authenticatorGetAssertion operation, by verifying that the RP ID  
 2455 associated with the requested credential exactly matches the RP ID  
 2456 supplied by the client, and that the RP ID is a registrable domain  
 2457 suffix of or is equal to the effective domain of the RP's origin's  
 2458 effective domain.

2596 which user consent was provided, and the prompt shown to the user  
 2597 by the authenticator. The assertion signature format is illustrated  
 2598 in Figure 2, below.  
 2599  
 2600 The formats of these signatures, as well as the procedures for  
 2601 generating them, are specified below.  
 2602  
 2603 6.1. Authenticator data  
 2604  
 2605 The authenticator data structure encodes contextual bindings made by  
 2606 the authenticator. These bindings are controlled by the authenticator  
 2607 itself, and derive their trust from the Relying Party's assessment of  
 2608 the security properties of the authenticator. In one extreme case, the  
 2609 authenticator may be embedded in the client, and its bindings may be no  
 2610 more trustworthy than the client data. At the other extreme, the  
 2611 authenticator may be a discrete entity with high-security hardware and  
 2612 software, connected to the client over a secure channel. In both cases,  
 2613 the Relying Party receives the authenticator data in the same format,  
 2614 and uses its knowledge of the authenticator to make trust decisions.  
 2615  
 2616 The authenticator data has a compact but extensible encoding. This is  
 2617 desired since authenticators can be devices with limited capabilities  
 2618 and low power requirements, with much simpler software stacks than the  
 2619 client platform components.  
 2620  
 2621 The authenticator data structure is a byte array of 37 bytes or more,  
 2622 as follows.  
 2623  
 2624 Name Length (in bytes) Description  
 2625 rpIdHash 32 SHA-256 hash of the RP ID associated with the credential.  
 2626 flags 1 Flags (bit 0 is the least significant bit):  
 2627 \* Bit 0: User Present (UP) result.  
 2628 + 1 means the user is present.  
 2629 + 0 means the user is not present.  
 2630 \* Bit 1: Reserved for future use (RFU1).  
 2631 \* Bit 2: User Verified (UV) result.  
 2632 + 1 means the user is verified.  
 2633 + 0 means the user is not verified.  
 2634 \* Bits 3-5: Reserved for future use (RFU2).  
 2635 \* Bit 6: Attested credential data included (AT).  
 2636 + Indicates whether the authenticator added attested credential  
 2637 data.  
 2638 \* Bit 7: Extension data included (ED).  
 2639 + Indicates if the authenticator data has extensions.  
 2640  
 2641 signCount 4 Signature counter, 32-bit unsigned big-endian integer.  
 2642 attestedCredentialData variable (if present) attested credential data  
 2643 (if present). See 6.3.1 Attested credential data for details. Its  
 2644 length depends on the length of the credential ID and credential public  
 2645 key being attested.  
 2646 extensions variable (if present) Extension-defined authenticator data.  
 2647 This is a CBOR [RFC7049] map with extension identifiers as keys, and  
 2648 authenticator extension outputs as values. See 9 WebAuthn Extensions  
 2649 for details.  
 2650  
 2651 NOTE: The names in the Name column in the above table are only for  
 2652 reference within this document, and are not present in the actual  
 2653 representation of the authenticator data.  
 2654  
 2655 The RP ID is originally received from the client when the credential is  
 2656 created, and again when an assertion is generated. However, it differs  
 2657 from other client data in some important ways. First, unlike the client  
 2658 data, the RP ID of a credential does not change between operations but  
 2659 instead remains the same for the lifetime of that credential. Secondly,  
 2660 it is validated by the authenticator during the  
 2661 authenticatorGetAssertion operation, by verifying that the RP ID  
 2662 associated with the requested credential exactly matches the RP ID  
 2663 supplied by the client, and that the RP ID is a registrable domain  
 2664 suffix of or is equal to the effective domain of the RP's origin's  
 2665 effective domain.

245E The UP flag SHALL be set if and only if the authenticator detected a  
 246C user through an authenticator specific gesture. The RFU bits SHALL be  
 2461 set to zero.  
 2462  
 2463 For attestation signatures, the authenticator MUST set the AT flag and  
 2464 include the attestedCredentialData. For authentication signatures, the  
 2465 AT flag MUST NOT be set and the attestedCredentialData MUST NOT be  
 2466 included.  
 2467  
 2468 If the authenticator does not include any extension data, it MUST set  
 2469 the ED flag to zero, and to one if extension data is included.  
 247C  
 2471 The figure below shows a visual representation of the authenticator  
 2472 data structure.  
 2473 [Authenticator data layout](#) Authenticator data layout.  
 2474  
 2475 Note that the authenticator data describes its own length: If the AT  
 2476 and ED flags are not set, it is always 37 bytes long. The attested  
 2477 credential data (which is only present if the AT flag is set) describes  
 2478 its own length. If the ED flag is set, then the total length is 37  
 2479 bytes plus the length of the attested credential data, plus the length  
 248C of the CBOR map that follows.  
 2481  
 2482  
 2483 6.1.1. Signature Counter Considerations  
 2484  
 2485 Authenticators MUST implement a signature counter feature. The  
 2486 signature counter is incremented for each successful  
 2487 authenticatorGetAssertion operation by some positive value, and its  
 2488 value is returned to the Relying Party within the authenticator data.  
 2489 The signature counter's purpose is to aid Relying Parties in detecting  
 249C cloned authenticators. Clone detection is more important for  
 2491 authenticators with limited protection measures.  
 2492  
 2493 An Relying Party stores the signature counter of the most recent  
 2494 authenticatorGetAssertion operation. Upon a new  
 2495 authenticatorGetAssertion operation, the Relying Party compares the  
 2496 stored signature counter value with the new signCount value returned in  
 2497 the assertion's authenticator data. If this new signCount value is less  
 2498 than or equal to the stored value, a cloned authenticator may exist, or  
 2499 the authenticator may be malfunctioning.  
 250C  
 2501 Detecting a signature counter mismatch does not indicate whether the  
 2502 current operation was performed by a cloned authenticator or the  
 2503 original authenticator. Relying Parties should address this situation  
 2504 appropriately relative to their individual situations, i.e., their risk  
 2505 tolerance.  
 2506  
 2507 Authenticators:  
 2508 \* should implement per-RP ID signature counters. This prevents the  
 2509 signature counter value from being shared between Relying Parties  
 251C and being possibly employed as a correlation handle for the user.  
 2511 Authenticators may implement a global signature counter, i.e., on a  
 2512 per-authenticator basis, but this is less privacy-friendly for  
 2513 users.  
 2514 \* should ensure that the signature counter value does not  
 2515 accidentally decrease (e.g., due to hardware failures).  
 2516  
 2517 6.2. Authenticator operations  
 2518  
 2519 A **client must** connect to an authenticator in order to invoke **any of the**  
 252C operations of that authenticator. This connection defines an  
 2521 authenticator session. An authenticator must maintain isolation between  
 2522 sessions. It may do this by only allowing one session to exist at any  
 2523 particular time, or by providing more complicated session management.  
 2524  
 2525 The following operations can be invoked by the client in an  
 2526 authenticator session.  
 2527  
 2528 6.2.1. [The authenticatorMakeCredential operation](#)

266E The UP flag SHALL be set if and only if the authenticator detected a  
 2667 user through an authenticator specific gesture. The RFU bits SHALL be  
 2668 set to zero.  
 2669  
 267C For attestation signatures, the authenticator MUST set the AT flag and  
 2671 include the attestedCredentialData. For authentication signatures, the  
 2672 AT flag MUST NOT be set and the attestedCredentialData MUST NOT be  
 2673 included.  
 2674  
 2675 If the authenticator does not include any extension data, it MUST set  
 2676 the ED flag to zero, and to one if extension data is included.  
 2677  
 2678 The figure below shows a visual representation of the authenticator  
 2679 data structure.  
 268C [\[fido-signature-formats-figure1.svg\]](#) Authenticator data layout.  
 2681  
 2682 Note that the authenticator data describes its own length: If the AT  
 2683 and ED flags are not set, it is always 37 bytes long. The attested  
 2684 credential data (which is only present if the AT flag is set) describes  
 2685 its own length. If the ED flag is set, then the total length is 37  
 2686 bytes plus the length of the attested credential data, plus the length  
 2687 of the CBOR map that follows.  
 2688  
 2689  
 269C 6.1.1. Signature Counter Considerations  
 2691  
 2692 Authenticators MUST implement a signature counter feature. The  
 2693 signature counter is incremented for each successful  
 2694 authenticatorGetAssertion operation by some positive value, and its  
 2695 value is returned to the Relying Party within the authenticator data.  
 2696 The signature counter's purpose is to aid Relying Parties in detecting  
 2697 cloned authenticators. Clone detection is more important for  
 2698 authenticators with limited protection measures.  
 2699  
 270C An Relying Party stores the signature counter of the most recent  
 2701 authenticatorGetAssertion operation. Upon a new  
 2702 authenticatorGetAssertion operation, the Relying Party compares the  
 2703 stored signature counter value with the new signCount value returned in  
 2704 the assertion's authenticator data. If this new signCount value is less  
 2705 than or equal to the stored value, a cloned authenticator may exist, or  
 2706 the authenticator may be malfunctioning.  
 2707  
 2708 Detecting a signature counter mismatch does not indicate whether the  
 2709 current operation was performed by a cloned authenticator or the  
 271C original authenticator. Relying Parties should address this situation  
 2711 appropriately relative to their individual situations, i.e., their risk  
 2712 tolerance.  
 2713  
 2714 Authenticators:  
 2715 \* should implement per-RP ID signature counters. This prevents the  
 2716 signature counter value from being shared between Relying Parties  
 2717 and being possibly employed as a correlation handle for the user.  
 2718 Authenticators may implement a global signature counter, i.e., on a  
 2719 per-authenticator basis, but this is less privacy-friendly for  
 272C users.  
 2721 \* should ensure that the signature counter value does not  
 2722 accidentally decrease (e.g., due to hardware failures).  
 2723  
 2724 6.2. Authenticator operations  
 2725  
 2726 A **WebAuthn Client MUST** connect to an authenticator in order to invoke  
 2727 **any of the** operations of that authenticator. This connection defines an  
 2728 authenticator session. An authenticator must maintain isolation between  
 2729 sessions. It may do this by only allowing one session to exist at any  
 273C particular time, or by providing more complicated session management.  
 2731  
 2732 The following operations can be invoked by the client in an  
 2733 authenticator session.  
 2734  
 2735 6.2.1. [Lookup Credential Source by Credential ID algorithm](#)

2529 It takes the following input parameters:

2530

2531 hash

2532 The hash of the serialized client data, provided by the client.

2533

2534 rpEntity

2535 The Relying Party's PublicKeyCredentialRpEntity.

2536

2537 userEntity

2538 The user account's PublicKeyCredentialUserEntity, containing the

2539 user handle given by the Relying Party.

2540

2541 requireResidentKey

2542 The authenticatorSelection.requireResidentKey value given by the

2543 Relying Party.

2544

2545 requireUserPresence

2546 A Boolean value provided by the client, which in invocations

2547 from a WebAuthn Client's [[Create]](origin, options,

2548 sameOriginWithAncestors) method is always set to the inverse of

2549 requireUserVerification.

2550

2551 requireUserVerification

2552 The effective user verification requirement for credential

2553 creation, a Boolean value provided by the client.

2554

2555 credTypesAndPubKeyAlgs

2556 A sequence of pairs of PublicKeyCredentialType and public key

2557 algorithms (COSEAlgorithmIdentifier) requested by the Relying

2558 Party. This sequence is ordered from most preferred to least

2559 preferred. The platform makes a best-effort to create the most

2560 preferred credential that it can.

2561

2562 excludeCredentialDescriptorList

2563 An optional list of PublicKeyCredentialDescriptor objects

2564 provided by the Relying Party with the intention that, if any of

2565 these are known to the authenticator, it should not create a new

2566 credential. excludeCredentialDescriptorList contains a list of

2567 known credentials.

2568

2569 extensions

2570 A map from extension identifiers to their authenticator

2571 extension inputs, created by the client based on the extensions

2572 requested by the Relying Party, if any.

2573

2574 Note: Before performing this operation, all other operations in

2575 progress in the authenticator session **must** be aborted by running the

2576 authenticatorCancel operation.

2577

2578 When this operation is invoked, the authenticator **must** perform the

2579 following procedure:

2580

2581 1. Check if all the supplied parameters are syntactically well-formed

2582 and of the correct length. If not, return an error code equivalent

2583 to "UnknownError" and terminate the operation.

2584

2585 2. Check if at least one of the specified combinations of

PublicKeyCredentialType and cryptographic parameters in

2736

2737 The result of looking up a credential id credentialId in an

2738 authenticator authenticator is the result of the following algorithm:

2739

2740 1. If authenticator can decrypt credentialId into a public key

2741 credential source credSource:

2742 1. Set credSource.id to credentialId.

2743 2. Return credSource.

2744

2745 2. For each public key credential source credSource of authenticator's

2746 credentials map:

2747 1. If credSource.id is credentialId, return credSource.

2748 3. Return null.

2749

2750 6.2.2. The authenticatorMakeCredential operation

2751

2752 It takes the following input parameters:

2753

2754 hash

2755 The hash of the serialized client data, provided by the client.

2756

2757 rpEntity

2758 The Relying Party's PublicKeyCredentialRpEntity.

2759

2760 userEntity

2761 The user account's PublicKeyCredentialUserEntity, containing the

2762 user handle given by the Relying Party.

2763

2764 requireResidentKey

2765 The authenticatorSelection.requireResidentKey value given by the

2766 Relying Party.

2767

2768 requireUserPresence

2769 A Boolean value provided by the client, which in invocations

2770 from a WebAuthn Client's [[Create]](origin, options,

2771 sameOriginWithAncestors) method is always set to the inverse of

2772 requireUserVerification.

2773

2774 requireUserVerification

2775 The effective user verification requirement for credential

2776 creation, a Boolean value provided by the client.

2777

2778 credTypesAndPubKeyAlgs

2779 A sequence of pairs of PublicKeyCredentialType and public key

2780 algorithms (COSEAlgorithmIdentifier) requested by the Relying

2781 Party. This sequence is ordered from most preferred to least

2782 preferred. The platform makes a best-effort to create the most

2783 preferred credential that it can.

2784

2785 excludeCredentialDescriptorList

2786 An optional list of PublicKeyCredentialDescriptor objects

2787 provided by the Relying Party with the intention that, if any of

2788 these are known to the authenticator, it should not create a new

2789 credential. excludeCredentialDescriptorList contains a list of

2790 known credentials.

2791

2792 extensions

2793 A **CBOR** map from extension identifiers to their authenticator

2794 extension inputs, created by the client based on the extensions

2795 requested by the Relying Party, if any.

2796

2797 Note: Before performing this operation, all other operations in

2798 progress in the authenticator session **MUST** be aborted by running the

2799 authenticatorCancel operation.

2800

2801 When this operation is invoked, the authenticator **MUST** perform the

2802 following procedure:

2803

2804 1. Check if all the supplied parameters are syntactically well-formed

2805 and of the correct length. If not, return an error code equivalent

to "UnknownError" and terminate the operation.

2. Check if at least one of the specified combinations of

PublicKeyCredentialType and cryptographic parameters in

2586 credTypesAndPubKeyAlgs is supported. If not, return an error code  
2587 equivalent to "NotSupportedError" and terminate the operation.  
2588 3. Check if any credential bound to this authenticator matches an item  
2589 of excludeCredentialDescriptorList. A match occurs if a credential  
2590 matches rpEntity.id and an excludeCredentialDescriptorList item's  
2591 excludeCredentialDescriptorList.id and  
2592 excludeCredentialDescriptorList.type. If so, return an error code  
2593 equivalent to "NotAllowedError" and terminate the operation.

- 2594 4. If requireResidentKey is true and the authenticator cannot store a  
2595 Client-side-resident Credential Private Key, return an error code  
2596 equivalent to "ConstraintError" and terminate the operation.
- 2597 5. If requireUserVerification is true and the authenticator cannot  
2598 perform user verification, return an error code equivalent to  
2599 "ConstraintError" and terminate the operation.
- 2600 6. Obtain user consent for creating a new credential. The prompt for  
2601 obtaining this consent is shown by the authenticator if it has its  
2602 own output capability, or by the user agent otherwise. The prompt  
2603 SHOULD display rpEntity.id, rpEntity.name, userEntity.name and  
2604 userEntity.displayName, if possible.  
2605 If requireUserVerification is true, the method of obtaining user  
2606 consent MUST include user verification.  
2607 If requireUserPresence is true, the method of obtaining user  
2608 consent MUST include a test of user presence.  
2609 If the user denies consent or if user verification fails, return an  
2610 error code equivalent to "NotAllowedError" and terminate the  
2611 operation.
- 2612 7. Once user consent has been obtained, generate a new credential  
2613 object:
  - 2614 1. Let (publicKey,privateKey) be a new pair of cryptographic keys  
2615 using the combination of PublicKeyCredentialType and  
2616 cryptographic parameters represented by the first item in  
2617 credTypesAndPubKeyAlgs that is supported by this  
2618 authenticator.
  - 2619 2. Let credentialId be a new identifier for this credential that  
2620 is globally unique with high probability across all  
2621 credentials with the same type across all authenticators.
  - 2622 3. Let userHandle be userEntity.id.
  - 2623 4. Associate the credentialId and privateKey with rpEntity.id and  
2624 userHandle.
  - 2625 5. Delete any older credentials with the same rpEntity.id and  
2626 userHandle that are stored locally by the authenticator.

2806 credTypesAndPubKeyAlgs is supported. If not, return an error code  
2807 equivalent to "NotSupportedError" and terminate the operation.  
2808 3. For each descriptor of excludeCredentialDescriptorList:  
2809 1. If looking up descriptor.id in this authenticator returns  
2810 non-null, and the returned item's RP ID and type match  
2811 rpEntity.id and excludeCredentialDescriptorList.type  
2812 respectively, then obtain user consent for creating a new  
2813 credential. The method of obtaining user consent MUST include  
2814 a test of user presence. If the user

2815  
2816 confirms consent to create a new credential  
2817 return an error code equivalent to  
2818 "InvalidStateError" and terminate the operation.  
2819

2820 does not consent to create a new credential  
2821 return an error code equivalent to "NotAllowedError"  
2822 and terminate the operation.  
2823

- 2824 4. If requireResidentKey is true and the authenticator cannot store a  
2825 Client-side-resident Credential Private Key, return an error code  
2826 equivalent to "ConstraintError" and terminate the operation.
- 2827 5. If requireUserVerification is true and the authenticator cannot  
2828 perform user verification, return an error code equivalent to  
2829 "ConstraintError" and terminate the operation.
- 2830 6. Obtain user consent for creating a new credential. The prompt for  
2831 obtaining this consent is shown by the authenticator if it has its  
2832 own output capability, or by the user agent otherwise. The prompt  
2833 SHOULD display rpEntity.id, rpEntity.name, userEntity.name and  
2834 userEntity.displayName, if possible.  
2835 If requireUserVerification is true, the method of obtaining user  
2836 consent MUST include user verification.  
2837 If requireUserPresence is true, the method of obtaining user  
2838 consent MUST include a test of user presence.  
2839 If the user does not consent or if user verification fails, return  
2840 an error code equivalent to "NotAllowedError" and terminate the  
2841 operation.
- 2842 7. Once user consent has been obtained, generate a new credential  
2843 object:
  - 2844 1. Let (publicKey,privateKey) be a new pair of cryptographic  
2845 keys using the combination of PublicKeyCredentialType and  
2846 cryptographic parameters represented by the first item in  
2847 credTypesAndPubKeyAlgs that is supported by this  
2848 authenticator.
  - 2849 2. Let userHandle be userEntity.id.
  - 2850 3. Let credentialSource be a new public key credential source  
2851 with the fields:
    - 2852 type  
2853 public-key.
    - 2854 privateKey  
2855 privateKey
    - 2856 rpId  
2857 rpEntity.id
    - 2858 userHandle  
2859 userHandle
    - 2860 otherUI  
2861 Any other information the authenticator chooses to  
2862 include.
  - 2863 4. If requireResidentKey is true or the authenticator chooses to  
2864 create a Client-side-resident Credential Private Key:
    - 2865 1. Let credentialId be a new credential id.
    - 2866 2. Set credentialSource.id to credentialId.
    - 2867 3. Let credentials be this authenticator's credentials map.
    - 2868 4. Set credentials[(rpEntity.id, userHandle)] to  
2869 credentialSource.

2627 8. If any error occurred while creating the new credential object,  
 2628 return an error code equivalent to "UnknownError" and terminate the  
 2629 operation.  
 2630 9. Let processedExtensions be the result of authenticator extension  
 2631 processing for each supported extension identifier/**input pair in**  
 2632 **extensions.**  
 2633 10. If the authenticator supports:  
 2634  
 2635 a per-RP ID signature counter  
 2636 allocate the counter, associate it with the RP ID, and  
 2637 initialize the counter value as zero.  
 2638  
 2639 a global signature counter  
 2640 Use the global signature counter's actual value when  
 2641 generating authenticator data.  
 2642  
 2643 a per credential signature counter  
 2644 allocate the counter, associate it with the new  
 2645 credential, and initialize the counter value as zero.  
 2646  
 2647 11. Let attestedCredentialData be the attested credential data byte  
 2648 array including the credentialId and publicKey.  
 2649 12. Let authenticatorData be the byte array specified in 6.1  
 2650 Authenticator data, including attestedCredentialData as the  
 2651 attestedCredentialData and processedExtensions, if any, as the  
 2652 extensions.  
 2653 13. Return the attestation object for the new credential created by the  
 2654 procedure specified in 6.3.4 Generating an Attestation Object  
 2655 using an authenticator-chosen attestation statement format,  
 2656 authenticatorData, and hash. For more details on attestation, see  
 2657 6.3 Attestation.  
 2658  
 2659 On successful completion of this operation, the authenticator returns  
 2660 the attestation object to the client.  
 2661  
 2662 **6.2.2. The authenticatorGetAssertion operation**  
 2663  
 2664 It takes the following input parameters:  
 2665  
 2666 rpld  
 2667 The caller's RP ID, as determined by the user agent and the  
 2668 client.  
 2669  
 2670 hash  
 2671 The hash of the serialized client data, provided by the client.  
 2672  
 2673 allowCredentialDescriptorList  
 2674 An optional list of PublicKeyCredentialDescriptors describing  
 2675 credentials acceptable to the Relying Party (possibly filtered  
 2676 by the client), if any.  
 2677  
 2678 requireUserPresence  
 2679 A Boolean value provided by the client, which in invocations  
 2680 from a WebAuthn Client's [[DiscoverFromExternalSource]](origin,  
 2681 options, sameOriginWithAncestors) method is always set to the  
 2682 inverse of requireUserVerification.  
 2683  
 2684 requireUserVerification  
 2685 The effective user verification requirement for assertion, a  
 2686 Boolean value provided by the client.  
 2687  
 2688 extensions  
 2689 A map from extension identifiers to their authenticator  
 2690 extension inputs, created by the client based on the extensions  
 2691 requested by the Relying Party, if any.  
 2692

2876 5. Otherwise:  
 2877 1. Let credentialId be the result of serializing and  
 2878 encrypting credentialSource so that only this  
 2879 authenticator can decrypt it.  
 2880 8. If any error occurred while creating the new credential object,  
 2881 return an error code equivalent to "UnknownError" and terminate the  
 2882 operation.  
 2883 9. Let processedExtensions be the result of authenticator extension  
 2884 processing for each supported extension identifier -> authenticator  
 2885 extension input in extensions.  
 2886 10. If the authenticator supports:  
 2887  
 2888 a per-RP ID signature counter  
 2889 allocate the counter, associate it with the RP ID, and  
 2890 initialize the counter value as zero.  
 2891  
 2892 a global signature counter  
 2893 Use the global signature counter's actual value when  
 2894 generating authenticator data.  
 2895  
 2896 a per credential signature counter  
 2897 allocate the counter, associate it with the new  
 2898 credential, and initialize the counter value as zero.  
 2899  
 2900 11. Let attestedCredentialData be the attested credential data byte  
 2901 array including the credentialId and publicKey.  
 2902 12. Let authenticatorData be the byte array specified in 6.1  
 2903 Authenticator data, including attestedCredentialData as the  
 2904 attestedCredentialData and processedExtensions, if any, as the  
 2905 extensions.  
 2906 13. Return the attestation object for the new credential created by the  
 2907 procedure specified in 6.3.4 Generating an Attestation Object  
 2908 using an authenticator-chosen attestation statement format,  
 2909 authenticatorData, and hash. For more details on attestation, see  
 2910 6.3 Attestation.  
 2911  
 2912 On successful completion of this operation, the authenticator returns  
 2913 the attestation object to the client.  
 2914  
 2915 **6.2.3. The authenticatorGetAssertion operation**  
 2916  
 2917 It takes the following input parameters:  
 2918  
 2919 rpld  
 2920 The caller's RP ID, as determined by the user agent and the  
 2921 client.  
 2922  
 2923 hash  
 2924 The hash of the serialized client data, provided by the client.  
 2925  
 2926 allowCredentialDescriptorList  
 2927 An optional list of PublicKeyCredentialDescriptors describing  
 2928 credentials acceptable to the Relying Party (possibly filtered  
 2929 by the client), if any.  
 2930  
 2931 requireUserPresence  
 2932 A Boolean value provided by the client, which in invocations  
 2933 from a WebAuthn Client's [[DiscoverFromExternalSource]](origin,  
 2934 options, sameOriginWithAncestors) method is always set to the  
 2935 inverse of requireUserVerification.  
 2936  
 2937 requireUserVerification  
 2938 The effective user verification requirement for assertion, a  
 2939 Boolean value provided by the client.  
 2940  
 2941 extensions  
 2942 A CBOR map from extension identifiers to their authenticator  
 2943 extension inputs, created by the client based on the extensions  
 2944 requested by the Relying Party, if any.  
 2945

2693 Note: Before performing this operation, all other operations in  
 2694 progress in the authenticator session must be aborted by running the  
 2695 authenticatorCancel operation.  
 2696

2697 When this method is invoked, the authenticator **must** perform the  
 2698 following procedure:  
 2699 1. Check if all the supplied parameters are syntactically well-formed  
 2700 and of the correct length. If not, return an error code equivalent  
 2701 to "UnknownError" and terminate the operation.  
 2702 2. If requireUserVerification is true and the authenticator cannot  
 2703 perform user verification, return an error code equivalent to  
 2704 "ConstraintError" and terminate the operation.  
 2705 3. If allowCredentialDescriptorList was not supplied, set it to a list  
 2706 of all credentials stored for rpId (as determined by an exact match  
 2707 of rpId).  
 2708 4. Remove any items from allowCredentialDescriptorList that do not  
 2709 match a credential bound to this authenticator. A match occurs if a  
 2710 credential matches rpId and an allowCredentialDescriptorList item's  
 2711 id and type members.  
 2712 5. If allowCredentialDescriptorList is now empty, return an error code  
 2713 equivalent to "NotAllowedError" and terminate the operation.  
 2714 6. Let selectedCredential be a credential as follows. If the size of  
 2715 allowCredentialDescriptorList  
 2716 is exactly 1  
 2717 Let selectedCredential be the credential matching  
 2718 allowCredentialDescriptorList[0].  
 2719 is greater than 1  
 2720 Prompt the user to select selectedCredential from the  
 2721 credentials matching the items in  
 2722 allowCredentialDescriptorList.  
 2723  
 2724 7. Obtain user consent for using selectedCredential. The prompt for  
 2725 obtaining this consent may be shown by the authenticator if it has  
 2726 its own output capability, or by the user agent otherwise. The  
 2727 prompt SHOULD display the rpId and any additional displayable data  
 2728 associated with selectedCredential, if possible.  
 2729 If requireUserVerification is true, the method of obtaining user  
 2730 consent MUST include user verification.  
 2731 If requireUserPresence is true, the method of obtaining user  
 2732 consent MUST include a test of user presence.  
 2733 If the user denies consent or if user verification fails, return an  
 2734 error code equivalent to "NotAllowedError" and terminate the  
 2735 operation.  
 2736 8. Let processedExtensions be the result of authenticator extension  
 2737 processing for each supported extension identifier/input pair in  
 2738 extensions.  
 2739 9. Increment the RP ID-associated signature counter or the global  
 2740 signature counter value, depending on which approach is implemented  
 2741 by the authenticator, by some positive value.  
 2742 10. Let authenticatorData be the byte array specified in 6.1  
 2743 Authenticator data including processedExtensions, if any, as the  
 2744 extensions and excluding attestedCredentialData.  
 2745 11. Let signature be the assertion signature of the concatenation  
 2746 authenticatorData || hash using the private key of  
 2747 selectedCredential as shown in Figure 2, below. A simple,  
 2748 undelimited concatenation is safe to use here because the  
 2749 authenticator data describes its own length. The hash of the  
 2750 serialized client data (which potentially has a variable length) is  
 2751 always the last element.  
 2752 Generating an assertion signature Generating an assertion  
 2753 signature.  
 2754 12. If any error occurred while generating the assertion signature,  
 2755 return an error code equivalent to "UnknownError" and terminate the  
 2756 operation.  
 2757 13. Return to the user agent:  
 2758 + selectedCredential's credential ID, if either a list of  
 2759 credentials of size 2 or greater was supplied by the client,  
 2760 or no such list was supplied. Otherwise, return only the below  
 2761  
 2762

2946 Note: Before performing this operation, all other operations in  
 2947 progress in the authenticator session must be aborted by running the  
 2948 authenticatorCancel operation.  
 2949

2950 When this method is invoked, the authenticator **MUST** perform the  
 2951 following procedure:  
 2952 1. Check if all the supplied parameters are syntactically well-formed  
 2953 and of the correct length. If not, return an error code equivalent  
 2954 to "UnknownError" and terminate the operation.  
 2955 2. Let credentialOptions be a new empty set of public key credential  
 2956 sources.  
 2957 3. If allowCredentialDescriptorList was supplied, then for each  
 2958 descriptor of allowCredentialDescriptorList:  
 2959 1. Let credSource be the result of looking up descriptor.id in  
 2960 this authenticator.  
 2961 2. If credSource is not null, append it to credentialOptions.  
 2962 4. Otherwise (allowCredentialDescriptorList was not supplied), for  
 2963 each key -> credSource of this authenticator's credentials map,  
 2964 append credSource to credentialOptions.  
 2965 5. Remove any items from credentialOptions whose rpId is not equal to  
 2966 rpId.  
 2967 6. If credentialOptions is now empty, return an error code equivalent  
 2968 to "NotAllowedError" and terminate the operation.  
 2969 7. Prompt the user to select a public key credential source  
 2970 selectedCredential from credentialOptions. Obtain user consent for  
 2971 using selectedCredential. The prompt for obtaining this consent may  
 2972 be shown by the authenticator if it has its own output capability,  
 2973 or by the user agent otherwise.  
 2974  
 2975 If requireUserVerification is true, the method of obtaining user  
 2976 consent MUST include user verification.  
 2977 If requireUserPresence is true, the method of obtaining user  
 2978 consent MUST include a test of user presence.  
 2979 If the user does not consent, return an error code equivalent to  
 2980 "NotAllowedError" and terminate the operation.  
 2981 8. Let processedExtensions be the result of authenticator extension  
 2982 processing for each supported extension identifier -> authenticator  
 2983 extension input in extensions.  
 2984 9. Increment the RP ID-associated signature counter or the global  
 2985 signature counter value, depending on which approach is implemented  
 2986 by the authenticator, by some positive value.  
 2987 10. Let authenticatorData be the byte array specified in 6.1  
 2988 Authenticator data including processedExtensions, if any, as the  
 2989 extensions and excluding attestedCredentialData.  
 2990 11. Let signature be the assertion signature of the concatenation  
 2991 authenticatorData || hash using the privateKey of  
 2992 selectedCredential as shown in Figure 2, below. A simple,  
 2993 undelimited concatenation is safe to use here because the  
 2994 authenticator data describes its own length. The hash of the  
 2995 serialized client data (which potentially has a variable length) is  
 2996 always the last element.  
 2997 [fido-signature-formats-figure2.svg] Generating an assertion  
 2998 signature.  
 2999 12. If any error occurred while generating the assertion signature,  
 3000 return an error code equivalent to "UnknownError" and terminate the  
 3001 operation.  
 3002 13. Return to the user agent:  
 3003 + selectedCredential.id, if either a list of credentials (i.e.,  
 3004 allowCredentialDescriptorList) of length 2 or greater was  
 3005 supplied by the client, or no such list was supplied.

2763 values.  
 2764 Note: If the client supplies a list of exactly one credential  
 2765 and it was successfully employed, then its credential ID is  
 2766 not returned since the client already knows it. This saves  
 2767 transmitting these bytes over what may be a constrained  
 2768 connection in what is likely a common case.  
 2769 + authenticatorData  
 2770 + signature  
 2771 + The user handle associated with selectedCredential.

2772  
 2773 If the authenticator cannot find any credential corresponding to the  
 2774 specified Relying Party that matches the specified criteria, it  
 2775 terminates the operation and returns an error.  
 2776

2777 6.2.3. The authenticatorCancel operation

2778 This operation takes no input parameters and returns no result.

2781 When this operation is invoked by the client in an authenticator  
 2782 session, it has the effect of terminating any  
 2783 authenticatorMakeCredential or authenticatorGetAssertion operation  
 2784 currently in progress in that authenticator session. The authenticator  
 2785 stops prompting for, or accepting, any user input related to  
 2786 authorizing the canceled operation. The client ignores any further  
 2787 responses from the authenticator for the canceled operation.  
 2788

2789 This operation is ignored if it is invoked in an authenticator session  
 2790 which does not have an authenticatorMakeCredential or  
 2791 authenticatorGetAssertion operation currently in progress.  
 2792

2793 6.3. Attestation

2795 Authenticators **must** also provide some form of attestation. The basic  
 2796 requirement is that the authenticator can produce, for each credential  
 2797 public key, an attestation statement verifiable by the Relying Party.  
 2798 Typically, this attestation statement contains a signature by an  
 2799 attestation private key over the attested credential public key and a  
 2800 challenge, as well as a certificate or similar data providing  
 2801 provenance information for the attestation public key, enabling the  
 2802 Relying Party to make a trust decision. However, if an attestation key  
 2803 pair is not available, then the authenticator **MUST** perform self  
 2804 attestation of the credential public key with the corresponding  
 2805 credential private key. All this information is returned by  
 2806 authenticators any time a new public key credential is generated, in  
 2807 the overall form of an attestation object. The relationship of the  
 2808 attestation object with authenticator data (containing attested  
 2809 credential data) and the attestation statement is illustrated in figure  
 2810 3, below.

2811 Attestation object layout illustrating the included authenticator data  
 2812 (containing attested credential data) and the attestation statement.  
 2813 Attestation object layout illustrating the included authenticator data  
 2814 (containing attested credential data) and the attestation statement.

2815 This figure illustrates only the packed attestation statement format.  
 2816 Several additional attestation statement formats are defined in 8  
 2817 Defined Attestation Statement Formats.  
 2818  
 2819

2820 An important component of the attestation object is the attestation  
 2821 statement. This is a specific type of signed data object, containing  
 2822 statements about a public key credential itself and the authenticator  
 2823 that created it. It contains an attestation signature created using the  
 2824 key of the attesting authority (except for the case of self  
 2825 attestation, when it is created using the credential private key). In  
 2826 order to correctly interpret an attestation statement, a Relying Party  
 2827 needs to understand these two aspects of attestation:

- 2828 1. The attestation statement format is the manner in which the  
 2829 signature is represented and the various contextual bindings are  
 2830 incorporated into the attestation statement by the authenticator.  
 2831

3005 Note: If, within allowCredentialDescriptorList, the client  
 3006 supplied exactly one credential and it was successfully  
 3007 employed, then its credential ID is not returned since the  
 3008 client already knows it. This saves transmitting these bytes  
 3009 over what may be a constrained connection in what is likely a  
 3010 common case.  
 3011 + authenticatorData  
 3012 + signature  
 3013 + selectedCredential.userHandle  
 3014 Note: the returned userHandle value may be null, see:  
 3015 userHandleResult.  
 3016

3017 If the authenticator cannot find any credential corresponding to the  
 3018 specified Relying Party that matches the specified criteria, it  
 3019 terminates the operation and returns an error.  
 3020

3021 6.2.4. The authenticatorCancel operation

3022 This operation takes no input parameters and returns no result.

3024 When this operation is invoked by the client in an authenticator  
 3025 session, it has the effect of terminating any  
 3026 authenticatorMakeCredential or authenticatorGetAssertion operation  
 3027 currently in progress in that authenticator session. The authenticator  
 3028 stops prompting for, or accepting, any user input related to  
 3029 authorizing the canceled operation. The client ignores any further  
 3030 responses from the authenticator for the canceled operation.  
 3031

3032 This operation is ignored if it is invoked in an authenticator session  
 3033 which does not have an authenticatorMakeCredential or  
 3034 authenticatorGetAssertion operation currently in progress.  
 3035

3036 6.3. Attestation

3038 Authenticators **MUST** also provide some form of attestation. The basic  
 3039 requirement is that the authenticator can produce, for each credential  
 3040 public key, an attestation statement verifiable by the Relying Party.  
 3041 Typically, this attestation statement contains a signature by an  
 3042 attestation private key over the attested credential public key and a  
 3043 challenge, as well as a certificate or similar data providing  
 3044 provenance information for the attestation public key, enabling the  
 3045 Relying Party to make a trust decision. However, if an attestation key  
 3046 pair is not available, then the authenticator **MUST** perform self  
 3047 attestation of the credential public key with the corresponding  
 3048 credential private key. All this information is returned by  
 3049 authenticators any time a new public key credential is generated, in  
 3050 the overall form of an attestation object. The relationship of the  
 3051 attestation object with authenticator data (containing attested  
 3052 credential data) and the attestation statement is illustrated in figure  
 3053 3, below.

3054 Attestation Object Layout diagram Attestation object layout  
 3055 illustrating the included authenticator data (containing attested  
 3056 credential data) and the attestation statement.  
 3057

3058 This figure illustrates only the packed attestation statement format.  
 3059 Several additional attestation statement formats are defined in 8  
 3060 Defined Attestation Statement Formats.  
 3061  
 3062

3063 An important component of the attestation object is the attestation  
 3064 statement. This is a specific type of signed data object, containing  
 3065 statements about a public key credential itself and the authenticator  
 3066 that created it. It contains an attestation signature created using the  
 3067 key of the attesting authority (except for the case of self  
 3068 attestation, when it is created using the credential private key). In  
 3069 order to correctly interpret an attestation statement, a Relying Party  
 3070 needs to understand these two aspects of attestation:

- 3071 1. The attestation statement format is the manner in which the  
 3072 signature is represented and the various contextual bindings are  
 3073 incorporated into the attestation statement by the authenticator.

2831 In other words, this defines the syntax of the statement. Various  
 2832 existing devices and platforms (such as TPMs and the Android OS)  
 2833 have previously defined attestation statement formats. This  
 2834 specification supports a variety of such formats in an extensible  
 2835 way, as defined in 6.3.2 Attestation Statement Formats.  
 2836 2. The attestation type defines the semantics of attestation  
 2837 statements and their underlying trust models. Specifically, it  
 2838 defines how a Relying Party establishes trust in a particular  
 2839 attestation statement, after verifying that it is cryptographically  
 2840 valid. This specification supports a number of attestation types,  
 2841 as described in 6.3.3 Attestation Types.  
 2842  
 2843 In general, there is no simple mapping between attestation statement  
 2844 formats and attestation types. For example, the "packed" attestation  
 2845 statement format defined in 8.2 Packed Attestation Statement Format  
 2846 can be used in conjunction with all attestation types, while other  
 2847 formats and types have more limited applicability.  
 2848  
 2849 The privacy, security and operational characteristics of attestation  
 2850 depend on:  
 2851 \* The attestation type, which determines the trust model,  
 2852 \* The attestation statement format, which **may** constrain the strength  
 2853 of the attestation by limiting what can be expressed in an  
 2854 attestation statement, and  
 2855 \* The characteristics of the individual authenticator, such as its  
 2856 construction, whether part or all of it runs in a secure operating  
 2857 environment, and so on.  
 2858  
 2859 It is expected that most authenticators will support a small number of  
 2860 attestation types and attestation statement formats, while Relying  
 2861 Parties will decide what attestation types are acceptable to them by  
 2862 policy. Relying Parties will also need to understand the  
 2863 characteristics of the authenticators that they trust, based on  
 2864 information they have about these authenticators. For example, the FIDO  
 2865 Metadata Service [FIDOMetadataService] provides one way to access such  
 2866 information.  
 2867  
 2868 6.3.1. Attested credential data  
 2869  
 2870 Attested credential data is a variable-length byte array added to the  
 2871 authenticator data when generating an attestation object for a given  
 2872 credential. It has the following format:  
 2873  
 2874 Name Length (in bytes) Description  
 2875 aaguid 16 The AAGUID of the authenticator.  
 2876 credentialIdLength 2 Byte length L of Credential ID  
 2877 credentialId L Credential ID  
 2878 credentialPublicKey variable The credential public key encoded in  
 2879 COSE\_Key format, as defined in Section 7 of [RFC8152]. **The encoded**  
 2880 **credential public key MUST contain the "alg" parameter and MUST NOT**  
 2881 **contain any other optional parameters. The "alg" parameter MUST contain**  
 2882 **a COSEAlgorithmIdentifier value.**

2883  
 2884 NOTE: The names in the Name column in the above table are only for  
 2885 reference within this document, and are not present in the actual  
 2886 representation of the attested credential data.  
 2887

3074 In other words, this defines the syntax of the statement. Various  
 3075 existing devices and platforms (such as TPMs and the Android OS)  
 3076 have previously defined attestation statement formats. This  
 3077 specification supports a variety of such formats in an extensible  
 3078 way, as defined in 6.3.2 Attestation Statement Formats.  
 3079 2. The attestation type defines the semantics of attestation  
 3080 statements and their underlying trust models. Specifically, it  
 3081 defines how a Relying Party establishes trust in a particular  
 3082 attestation statement, after verifying that it is cryptographically  
 3083 valid. This specification supports a number of attestation types,  
 3084 as described in 6.3.3 Attestation Types.  
 3085  
 3086 In general, there is no simple mapping between attestation statement  
 3087 formats and attestation types. For example, the "packed" attestation  
 3088 statement format defined in 8.2 Packed Attestation Statement Format  
 3089 can be used in conjunction with all attestation types, while other  
 3090 formats and types have more limited applicability.  
 3091  
 3092 The privacy, security and operational characteristics of attestation  
 3093 depend on:  
 3094 \* The attestation type, which determines the trust model,  
 3095 \* The attestation statement format, which **MAY** constrain the strength  
 3096 of the attestation by limiting what can be expressed in an  
 3097 attestation statement, and  
 3098 \* The characteristics of the individual authenticator, such as its  
 3099 construction, whether part or all of it runs in a secure operating  
 3100 environment, and so on.  
 3101  
 3102 It is expected that most authenticators will support a small number of  
 3103 attestation types and attestation statement formats, while Relying  
 3104 Parties will decide what attestation types are acceptable to them by  
 3105 policy. Relying Parties will also need to understand the  
 3106 characteristics of the authenticators that they trust, based on  
 3107 information they have about these authenticators. For example, the FIDO  
 3108 Metadata Service [FIDOMetadataService] provides one way to access such  
 3109 information.  
 3110  
 3111 6.3.1. Attested credential data  
 3112  
 3113 Attested credential data is a variable-length byte array added to the  
 3114 authenticator data when generating an attestation object for a given  
 3115 credential. It has the following format:  
 3116  
 3117 Name Length (in bytes) Description  
 3118 aaguid 16 The AAGUID of the authenticator.  
 3119 credentialIdLength 2 Byte length L of Credential ID, **16-bit unsigned**  
 3120 **big-endian integer.**  
 3121 credentialId L Credential ID  
 3122 credentialPublicKey variable The credential public key encoded in  
 3123 COSE\_Key format, as defined in Section 7 of [RFC8152], **using the CTAP2**  
 3124 **canonical CBOR encoding form. The COSE\_Key-encoded credential public**  
 3125 **key MUST contain the optional "alg" parameter and MUST NOT contain any**  
 3126 **other optional parameters. The "alg" parameter MUST contain a**  
 3127 **COSEAlgorithmIdentifier value. The encoded credential public key MUST**  
 3128 **also contain any additional required parameters stipulated by the**  
 3129 **relevant key type specification, i.e., required for the key type "kty"**  
 3130 **and algorithm "alg" (see Section 8 of [RFC8152]).**

3131  
 3132 NOTE: The names in the Name column in the above table are only for  
 3133 reference within this document, and are not present in the actual  
 3134 representation of the attested credential data.  
 3135

3136 **6.3.1.1. Examples of credentialPublicKey Values encoded in COSE\_Key format**

3137  
 3138 This section provides examples of COSE\_Key-encoded Elliptic Curve and  
 3139 RSA public keys for the ES256, PS256, and RS256 signature algorithms.  
 3140 These examples adhere to the rules defined above for the  
 3141 credentialPublicKey value, and are presented in [CDDL] for clarity.

3142  
 3143 [RFC8152] Section 7 defines the general framework for all

3144 COSE\_Key-encoded keys. Specific key types for specific algorithms are  
3145 defined in other sections of [RFC8152] as well as in other  
3146 specifications, as noted below.  
3147  
3148 Below is an example of a COSE\_Key-encoded Elliptic Curve public key in  
3149 EC2 format (see [RFC8152] Section 13.1), on the P-256 curve, to be used  
3150 with the ES256 signature algorithm (ECDSA w/ SHA-256, see [RFC8152]  
3151 Section 8.1):  
3152 {  
3153 1: 2, ; kty: EC2 key type  
3154 3: -7, ; alg: ES256 signature algorithm  
3155 -1: 1, ; crv: P-256 curve  
3156 -2: x, ; x-coordinate as byte string 32 bytes in length  
3157 ; e.g., in hex: 65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de108d  
3158 e439c08551d  
3159 -3: y ; y-coordinate as byte string 32 bytes in length  
3160 ; e.g., in hex: 1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9e  
3161 ecd0084d19c  
3162 }  
3163  
3164 Below is the above Elliptic Curve public key encoded in the CTAP2  
3165 canonical CBOR encoding form, whitespace and line breaks are included  
3166 here for clarity and to match the [CDDL] presentation above:  
3167 A5  
3168 01 02  
3169  
3170 03 26  
3171  
3172 20 01  
3173  
3174 21 58 20 65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de108de439c08551d  
3175  
3176 22 58 20 1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9eecd0084d19c  
3177  
3178 Below is an example of a COSE\_Key-encoded 2048-bit RSA public key (see  
3179 [RFC8230] Section 4), to be used with the PS256 signature algorithm  
3180 (RSASSA-PSS with SHA-256, see [RFC8230] Section 2):  
3181 {  
3182 1: 3, ; kty: RSA key type  
3183 3: -37, ; alg: PS256  
3184 -1: n, ; n: RSA modulus n byte string 256 bytes in length  
3185 ; e.g., in hex (middle bytes elided for brevity): DB5F651550...6  
3186 DC6548ACC3  
3187 -2: e ; e: RSA public exponent e byte string 3 bytes in length  
3188 ; e.g., in hex: 010001  
3189 }  
3190  
3191 Below is an example of the same COSE\_Key-encoded RSA public key as  
3192 above, to be used with the RS256 signature algorithm (RSASSA-PKCS1-v1\_5  
3193 with SHA-256, see 11.3 COSE Algorithm Registrations):  
3194 {  
3195 1: 3, ; kty: RSA key type  
3196 3: -257, ; alg: RS256  
3197 -1: n, ; n: RSA modulus n byte string 256 bytes in length  
3198 ; e.g., in hex (middle bytes elided for brevity): DB5F651550...6  
3199 DC6548ACC3  
3200 -2: e ; e: RSA public exponent e byte string 3 bytes in length  
3201 ; e.g., in hex: 010001  
3202 }  
3203  
3204  
3205  
3206  
3207  
3208  
3209  
3210  
3211  
3212  
3213

2888 6.3.2. Attestation Statement Formats

2889 As described above, an attestation statement format is a data format  
2890 which represents a cryptographic signature by an authenticator over a  
2891 set of contextual bindings. Each attestation statement format MUST be  
2892 defined using the following template:  
2893

- 2894 \* Attestation statement format identifier:
- 2895 \* Supported attestation types:
- 2896 \* Syntax: The syntax of an attestation statement produced in this  
2897 format, defined using [CDDL] for the extension point \$attStmtFormat

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

289E defined in 6.3.4 Generating an Attestation Object.  
 289F \* Signing procedure: The signing procedure for computing an  
 2900 attestation statement in this format given the public key  
 2901 credential to be attested, the authenticator data structure  
 2902 containing the authenticator data for the attestation, and the hash  
 2903 of the serialized client data.  
 2904 \* Verification procedure: The procedure for verifying an attestation  
 2905 statement, which takes the following verification procedure inputs:  
 2906 + attStmt: The attestation statement structure  
 2907 + authenticatorData: The authenticator data claimed to have been  
 2908 used for the attestation  
 2909 + clientDataHash: The hash of the serialized client data  
 2910 The procedure returns either:  
 2911 + An error indicating that the attestation is invalid, or  
 2912 + The attestation type, and the trust path. This attestation  
 2913 trust path is either empty (in case of self attestation), an  
 2914 identifier of a ECDAAs-Issuer public key (in the case of  
 2915 ECDAAs), or a set of X.509 certificates.

2916  
 2917 The initial list of specified attestation statement formats is in 8  
 2918 Defined Attestation Statement Formats.

### 2919 6.3.3. Attestation Types

2920 WebAuthn supports multiple attestation types:

#### 2921 Basic Attestation

2922 In the case of basic attestation [UAFProtocol], the  
 2923 authenticator's attestation key pair is specific to an  
 2924 authenticator model. Thus, authenticators of the same model  
 2925 often share the same attestation key pair. See 6.3.5.1 Privacy  
 2926 for further information.

#### 2927 Self Attestation

2928 In the case of self attestation, also known as surrogate basic  
 2929 attestation [UAFProtocol], the Authenticator does not have any  
 2930 specific attestation key. Instead it uses the credential private  
 2931 key to create the attestation signature. Authenticators without  
 2932 meaningful protection measures for an attestation private key  
 2933 typically use this attestation type.

#### 2934 Privacy CA

2935 In this case, the Authenticator owns an authenticator-specific  
 2936 (endorsement) key. This key is used to securely communicate with  
 2937 a trusted third party, the Privacy CA. The Authenticator can  
 2938 generate multiple attestation key pairs and asks the Privacy CA  
 2939 to issue an attestation certificate for it. Using this approach,  
 2940 the Authenticator can limit the exposure of the endorsement key  
 2941 (which is a global correlation handle) to Privacy CA(s).  
 2942 Attestation keys can be requested for each public key credential  
 2943 individually.

2944  
 2945 Note: This concept typically leads to multiple attestation  
 2946 certificates. The attestation certificate requested most  
 2947 recently is called "active".

#### 2948 Elliptic Curve based Direct Anonymous Attestation (ECDAAs)

2949 In this case, the Authenticator receives direct anonymous  
 2950 attestation (DAA) credentials from a single DAA-Issuer. These  
 2951 DAA credentials are used along with blinding to sign the  
 2952 attested credential data. The concept of blinding avoids the DAA  
 2953 credentials being misused as global correlation handle. WebAuthn  
 2954 supports DAA using elliptic curve cryptography and bilinear  
 2955 pairings, called ECDAAs (see [FIDOEcdaaAlgorithm]) in this  
 2956 specification. Consequently we denote the DAA-Issuer as  
 2957 ECDAAs-Issuer (see [FIDOEcdaaAlgorithm]).

3214 defined in 6.3.4 Generating an Attestation Object.  
 3215 \* Signing procedure: The signing procedure for computing an  
 3216 attestation statement in this format given the public key  
 3217 credential to be attested, the authenticator data structure  
 3218 containing the authenticator data for the attestation, and the hash  
 3219 of the serialized client data.  
 3220 \* Verification procedure: The procedure for verifying an attestation  
 3221 statement, which takes the following verification procedure inputs:  
 3222 + attStmt: The attestation statement structure  
 3223 + authenticatorData: The authenticator data claimed to have been  
 3224 used for the attestation  
 3225 + clientDataHash: The hash of the serialized client data  
 3226 The procedure returns either:  
 3227 + An error indicating that the attestation is invalid, or  
 3228 + The attestation type, and the trust path. This attestation  
 3229 trust path is either empty (in case of self attestation), an  
 3230 identifier of an ECDAAs-Issuer public key (in the case of  
 3231 ECDAAs), or a set of X.509 certificates.

3232  
 3233 The initial list of specified attestation statement formats is in 8  
 3234 Defined Attestation Statement Formats.

### 3235 6.3.3. Attestation Types

3236 WebAuthn supports multiple attestation types:

#### 3237 Basic Attestation (Basic)

3238 In the case of basic attestation [UAFProtocol], the  
 3239 authenticator's attestation key pair is specific to an  
 3240 authenticator model. Thus, authenticators of the same model  
 3241 often share the same attestation key pair. See 14.1 Attestation  
 3242 Privacy for further information.

#### 3243 Self Attestation (Self)

3244 In the case of self attestation, also known as surrogate basic  
 3245 attestation [UAFProtocol], the Authenticator does not have any  
 3246 specific attestation key. Instead it uses the credential private  
 3247 key to create the attestation signature. Authenticators without  
 3248 meaningful protection measures for an attestation private key  
 3249 typically use this attestation type.

#### 3250 Attestation CA (AttCA)

3251 In this case, an authenticator is based on a Trusted Platform  
 3252 Module (TPM) and holds an authenticator-specific "endorsement  
 3253 key" (EK). This key is used to securely communicate with a  
 3254 trusted third party, the Attestation CA  
 3255 [TCG-CMCPProfile-AIKCertEnroll] (formerly known as a "Privacy  
 3256 CA"). The authenticator can generate multiple attestation  
 3257 identity key pairs (AIK) and requests an Attestation CA to issue  
 3258 an AIK certificate for each. Using this approach, such an  
 3259 authenticator can limit the exposure of the EK (which is a  
 3260 global correlation handle) to Attestation CA(s). AIKs can be  
 3261 requested for each authenticator-generated public key credential  
 3262 individually, and conveyed to Relying Parties as attestation  
 3263 certificates.

3264  
 3265 Note: This concept typically leads to multiple attestation  
 3266 certificates. The attestation certificate requested most  
 3267 recently is called "active".

#### 3268 Elliptic Curve based Direct Anonymous Attestation (ECDAAs)

3269 In this case, the Authenticator receives direct anonymous  
 3270 attestation (DAA) credentials from a single DAA-Issuer. These  
 3271 DAA credentials are used along with blinding to sign the  
 3272 attested credential data. The concept of blinding avoids the DAA  
 3273 credentials being misused as global correlation handle. WebAuthn  
 3274 supports DAA using elliptic curve cryptography and bilinear  
 3275 pairings, called ECDAAs (see [FIDOEcdaaAlgorithm]) in this  
 3276 specification. Consequently we denote the DAA-Issuer as  
 3277 ECDAAs-Issuer (see [FIDOEcdaaAlgorithm]).

2964

### 6.3.4. Generating an Attestation Object

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

**attestationFormat**  
An attestation statement format.

**authData**  
A byte array containing authenticator data.

**hash**  
The hash of the serialized client data.

the authenticator MUST:

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

```
attObj = {
  authData: bytes,
  $$attStmtType
}
```

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

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

### 6.3.5. Security Considerations

#### 6.3.5.1. Privacy

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

- \* A WebAuthn authenticator manufacturer may choose to ship all of their devices with the same (or a fixed number of) attestation key(s) (called Basic Attestation). This will anonymize the user at the risk of not being able to revoke a particular attestation key should its WebAuthn Authenticator be compromised.
- \* A WebAuthn Authenticator may be capable of dynamically generating different attestation keys (and requesting related certificates) per origin (following the Privacy CA approach). For example, a WebAuthn Authenticator can ship with a master attestation key (and certificate), and combined with a cloud operated privacy CA, can dynamically generate per origin attestation keys and attestation certificates.
- \* 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.

#### 6.3.5.2. Attestation Certificate and Attestation Certificate CA Compromise

When an intermediate CA or a root CA used for issuing attestation certificates is compromised, WebAuthn authenticator attestation keys are still safe although their certificates can no longer be trusted. A WebAuthn Authenticator manufacturer that has recorded the public

3284

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

### 6.3.4. Generating an Attestation Object

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

**attestationFormat**  
An attestation statement format.

**authData**  
A byte array containing authenticator data.

**hash**  
The hash of the serialized client data.

the authenticator MUST:

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

```
attObj = {
  authData: bytes,
  $$attStmtType
}
```

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

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

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

3031 attestation keys for their devices can issue new attestation  
3032 certificates for these keys from a new intermediate CA or from a new  
3033 root CA. If the root CA changes, the Relying Parties must update their  
3034 trusted root certificates accordingly.

3035  
3036 A WebAuthn Authenticator attestation certificate must be revoked by the  
3037 issuing CA if its key has been compromised. A WebAuthn Authenticator  
3038 manufacturer may need to ship a firmware update and inject new  
3039 attestation keys and certificates into already manufactured WebAuthn  
3040 Authenticators, if the exposure was due to a firmware flaw. (The  
3041 process by which this happens is out of scope for this specification.)  
3042 If the WebAuthn Authenticator manufacturer does not have this  
3043 capability, then it may not be possible for Relying Parties to trust  
3044 any further attestation statements from the affected WebAuthn  
3045 Authenticators.

3046  
3047 If attestation certificate validation fails due to a revoked  
3048 intermediate attestation CA certificate, and the Relying Party's policy  
3049 requires rejecting the registration/authentication request in these  
3050 situations, then it is recommended that the Relying Party also  
3051 un-registers (or marks with a trust level equivalent to "self  
3052 attestation") public key credentials that were registered after the CA  
3053 compromise date using an attestation certificate chaining up to the  
3054 same intermediate CA. It is thus recommended that Relying Parties  
3055 remember intermediate attestation CA certificates during Authenticator  
3056 registration in order to un-register related public key credentials if  
3057 the registration was performed after revocation of such certificates.

3058  
3059 If an ECDAA attestation key has been compromised, it can be added to  
3060 the RogueList (i.e., the list of revoked authenticators) maintained by  
3061 the related ECDAA-Issuer. The Relying Party should verify whether an  
3062 authenticator belongs to the RogueList when performing ECDAA-Verify  
3063 (see section 3.6 in [FIDOEcdaaAlgorithm]). For example, the FIDO  
3064 Metadata Service [FIDOMetadataService] provides one way to access such  
3065 information.

### 3066 6.3.5.3. Attestation Certificate Hierarchy

3067  
3068 A 3-tier hierarchy for attestation certificates is recommended (i.e.,  
3069 Attestation Root, Attestation Issuing CA, Attestation Certificate). It  
3070 is also recommended that for each WebAuthn Authenticator device line  
3071 (i.e., model), a separate issuing CA is used to help facilitate  
3072 isolating problems with a specific version of a device.  
3073

3074  
3075 If the attestation root certificate is not dedicated to a single  
3076 WebAuthn Authenticator device line (i.e., AAGUID), the AAGUID should be  
3077 specified in the attestation certificate itself, so that it can be  
3078 verified against the authenticator data.

## 3079 7. Relying Party Operations

3080 Upon successful execution of create() or get(), the Relying Party's  
3081 script receives a PublicKeyCredential containing an  
3082  
3083

3324  
3325  
3326  
3327  
3328  
3329  
3330  
3331  
3332  
3333  
3334  
3335  
3336  
3337  
3338  
3339  
3340  
3341  
3342  
3343  
3344  
3345  
3346  
3347  
3348  
3349  
3350  
3351  
3352  
3353  
3354  
3355  
3356

\* For COSEAlgorithmIdentifier -7 (ES256), and other ECDSA-based algorithms, a signature value is encoded as an ASN.1 DER Ecdsa-Sig-Value, as defined in [RFC3279] section 2.2.3.

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

Note: As CTAP1/U2F devices are already producing signature values in this format, CTAP2 devices will also produce signature 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

3084 AuthenticatorAttestationResponse or AuthenticatorAssertionResponse  
3085 structure, respectively, from the client. It must then deliver the  
3086 contents of this structure to the Relying Party server, using methods  
3087 outside the scope of this specification. This section describes the  
3088 operations that the Relying Party must perform upon receipt of these  
3089 structures.

3091 7.1. Registering a new credential

3093 When registering a new credential, represented by a  
3094 AuthenticatorAttestationResponse structure, as part of a registration  
3095 ceremony, a Relying Party MUST proceed as follows:

- 3096 1. Perform JSON deserialization on the clientDataJSON field of the  
3097 AuthenticatorAttestationResponse object to extract the client data  
3098 C claimed as collected during the credential creation.
- 3099 2. Verify that the type in C is the string webauthn.create.
- 3100 3. Verify that the challenge in C matches the challenge that was sent  
3101 to the authenticator in the create() call.
- 3102 4. Verify that the origin in C matches the Relying Party's origin.
- 3103 5. Verify that the tokenBindingId in C matches the Token Binding ID  
3104 for the TLS connection over which the attestation was obtained.
- 3105 6. Verify that the clientExtensions in C is a subset of the extensions  
3106 requested by the RP and that the authenticatorExtensions in C is  
3107 also a subset of the extensions requested by the RP.
- 3108 7. Compute the hash of clientDataJSON using the algorithm identified  
3109 by C.hashAlgorithm.

- 3110 8. Perform CBOR decoding on the attestationObject field of the  
3111 AuthenticatorAttestationResponse structure to obtain the  
3112 attestation statement format fmt, the authenticator data authData,  
3113 and the attestation statement attStmt.
- 3114 9. Verify that the RP ID hash in authData is indeed the SHA-256 hash  
3115 of the RP ID expected by the RP.
- 3116 10. Determine the attestation statement format by performing an USASCII

3117 case-sensitive match on fmt against the set of supported WebAuthn  
3118 Attestation Statement Format Identifier values. The up-to-date list  
3119 of registered WebAuthn Attestation Statement Format Identifier  
3120 values is maintained in the in the IANA registry of the same name  
3121 [WebAuthn-Registries].

- 3122 11. Verify that attStmt is a correct attestation statement, conveying a  
3123 valid attestation signature, by using the attestation statement  
3124 format fmt's verification procedure given attStmt, authData and the  
3125 hash of the serialized client data computed in step 6.

3357 AuthenticatorAttestationResponse or AuthenticatorAssertionResponse  
3358 structure, respectively, from the client. It must then deliver the  
3359 contents of this structure to the Relying Party server, using methods  
3360 outside the scope of this specification. This section describes the  
3361 operations that the Relying Party must perform upon receipt of these  
3362 structures.

3364 7.1. Registering a new credential

3366 When registering a new credential, represented by an  
3367 AuthenticatorAttestationResponse structure response and an  
3368 AuthenticationExtensionsClientOutputs structure clientExtensionResults,  
3369 as part of a registration ceremony, a Relying Party MUST proceed as  
3370 follows:

- 3371 1. Let JSONtext be the result of running UTF-8 decode on the value of  
3372 response.clientDataJSON.  
3373 Note: Using any implementation of UTF-8 decode is acceptable as  
3374 long as it yields the same result as that yielded by the UTF-8  
3375 decode algorithm. In particular, any leading byte order mark (BOM)  
3376 MUST be stripped.
- 3377 2. Let C, the client data claimed as collected during the credential  
3378 creation, be the result of running an implementation-specific JSON  
3379 parser on JSONtext.  
3380 Note: C may be any implementation-specific data structure  
3381 representation, as long as C's components are referenceable, as  
3382 required by this algorithm.
- 3383 3. Verify that the value of C.type is webauthn.create.
- 3384 4. Verify that the value of C.challenge matches the challenge that was  
3385 sent to the authenticator in the create() call.
- 3386 5. Verify that the value of C.origin matches the Relying Party's  
3387 origin.
- 3388 6. Verify that the value of C.tokenBinding.status matches the state of  
3389 Token Binding for the TLS connection over which the assertion was  
3390 obtained. If Token Binding was used on that TLS connection, also  
3391 verify that C.tokenBinding.id matches the base64url encoding of the  
3392 Token Binding ID for the connection.
- 3393 7. Compute the hash of response.clientDataJSON using SHA-256.
- 3394 8. Perform CBOR decoding on the attestationObject field of the  
3395 AuthenticatorAttestationResponse structure to obtain the  
3396 attestation statement format fmt, the authenticator data authData,  
3397 and the attestation statement attStmt.
- 3398 9. Verify that the RP ID hash in authData is indeed the SHA-256 hash  
3399 of the RP ID expected by the RP.
- 3400 10. If user verification is required for this registration, verify that  
3401 the User Verified bit of the flags in authData is set.
- 3402 11. If user verification is not required for this registration, verify  
3403 that the User Present bit of the flags in authData is set.
- 3404 12. Verify that the values of the client extension outputs in  
3405 clientExtensionResults and the authenticator extension outputs in  
3406 the extensions in authData are as expected, considering the client  
3407 extension input values that were given as the extensions option in  
3408 the create() call. In particular, any extension identifier values  
3409 in the clientExtensionResults and the extensions in authData MUST  
3410 be also be present as extension identifier values in the extensions  
3411 member of options, i.e., no extensions are present that were not  
3412 requested. In the general case, the meaning of "are as expected" is  
3413 specific to the Relying Party and which extensions are in use.  
3414 Note: Since all extensions are OPTIONAL for both the client and the  
3415 authenticator, the Relying Party MUST be prepared to handle cases  
3416 where none or not all of the requested extensions were acted upon.
- 3417 13. Determine the attestation statement format by performing a USASCII  
3418 case-sensitive match on fmt against the set of supported WebAuthn  
3419 Attestation Statement Format Identifier values. The up-to-date list  
3420 of registered WebAuthn Attestation Statement Format Identifier  
3421 values is maintained in the in the IANA registry of the same name  
3422 [WebAuthn-Registries].
- 3423 14. Verify that attStmt is a correct attestation statement, conveying a  
3424 valid attestation signature, by using the attestation statement  
3425 format fmt's verification procedure given attStmt, authData and the  
3426 hash of the serialized client data computed in step 7.

3126 Note: Each attestation statement format specifies its own  
 3127 verification procedure. See 8 Defined Attestation Statement  
 3128 Formats for the initially-defined formats, and  
 3129 [WebAuthn-Registries] for the up-to-date list.  
 3130 12. If validation is successful, obtain a list of acceptable trust  
 3131 anchors (attestation root certificates or ECDAAs-Issuer public keys)  
 3132 for that attestation type and attestation statement format fmt,  
 3133 from a trusted source or from policy. For example, the FIDO  
 3134 Metadata Service [FIDOMetadataService] provides one way to obtain  
 3135 such information, using the aaguid in the attestedCredentialData in  
 3136 authData.  
 3137 13. Assess the attestation trustworthiness using the outputs of the  
 3138 verification procedure in step 10, as follows:  
 3139 + If self attestation was used, check if self attestation is  
 3140 acceptable under Relying Party policy.  
 3141 + If ECDAAs was used, verify that the identifier of the  
 3142 ECDAAs-Issuer public key used is included in the set of  
 3143 acceptable trust anchors obtained in step 11.  
 3144 + Otherwise, use the X.509 certificates returned by the  
 3145 verification procedure to verify that the attestation public  
 3146 key correctly chains up to an acceptable root certificate.  
 3147 14. If the attestation statement attStmt verified successfully and is

3148 found to be trustworthy, then register the new credential with the  
 3149 account that was denoted in the options.user passed to create(), by  
 3150 associating it with the credentialId and credentialPublicKey in the  
 3151 attestedCredentialData in authData, as appropriate for the Relying  
 3152 Party's system.

3153 15. If the attestation statement attStmt successfully verified but is  
 3154 not trustworthy per step 12 above, the Relying Party SHOULD fail  
 3155 the registration ceremony.  
 3156 NOTE: However, if permitted by policy, the Relying Party MAY  
 3157 register the credential ID and credential public key but treat the  
 3158 credential as one with self attestation (see 6.3.3 Attestation  
 3159 Types). If doing so, the Relying Party is asserting there is no  
 3160 cryptographic proof that the public key credential has been  
 3161 generated by a particular authenticator model. See [FIDOsecRef] and  
 3162 [UAFProtocol] for a more detailed discussion.  
 3163

3164 Verification of attestation objects requires that the Relying Party has  
 3165 a trusted method of determining acceptable trust anchors in step 11  
 3166 above. Also, if certificates are being used, the Relying Party **must**  
 3167 have access to certificate status information for the intermediate CA  
 3168 certificates. The Relying Party **must** also be able to build the  
 3169 attestation certificate chain if the client did not provide this chain  
 3170 in the attestation information.  
 3171

3172 To avoid ambiguity during authentication, the Relying Party SHOULD  
 3173 check that each credential is registered to no more than one user. If  
 3174 registration is requested for a credential that is already registered  
 3175 to a different user, the Relying Party SHOULD fail this ceremony, or it  
 3176 MAY decide to accept the registration, e.g. while deleting the older  
 3177 registration.  
 3178

3179 7.2. Verifying an authentication assertion  
 3180

3181 When verifying a given PublicKeyCredential structure (credential) as  
 3182 part of an authentication ceremony, the Relying Party **MUST** proceed as  
 3183 follows:  
 3184

- 3185 1. Using credential's id attribute (or the corresponding rawId, if

3427 Note: Each attestation statement format specifies its own  
 3428 verification procedure. See 8 Defined Attestation Statement  
 3429 Formats for the initially-defined formats, and  
 3430 [WebAuthn-Registries] for the up-to-date list.  
 3431 15. If validation is successful, obtain a list of acceptable trust  
 3432 anchors (attestation root certificates or ECDAAs-Issuer public keys)  
 3433 for that attestation type and attestation statement format fmt,  
 3434 from a trusted source or from policy. For example, the FIDO  
 3435 Metadata Service [FIDOMetadataService] provides one way to obtain  
 3436 such information, using the aaguid in the attestedCredentialData in  
 3437 authData.  
 3438 16. Assess the attestation trustworthiness using the outputs of the  
 3439 verification procedure in step 14, as follows:  
 3440 + If self attestation was used, check if self attestation is  
 3441 acceptable under Relying Party policy.  
 3442 + If ECDAAs was used, verify that the identifier of the  
 3443 ECDAAs-Issuer public key used is included in the set of  
 3444 acceptable trust anchors obtained in step 15.  
 3445 + Otherwise, use the X.509 certificates returned by the  
 3446 verification procedure to verify that the attestation public  
 3447 key correctly chains up to an acceptable root certificate.  
 3448 17. Check that the credentialId is not yet registered to any other  
 3449 user. If registration is requested for a credential that is already  
 3450 registered to a different user, the Relying Party SHOULD fail this  
 3451 registration ceremony, or it MAY decide to accept the registration,  
 3452 e.g. while deleting the older registration.  
 3453 18. If the attestation statement attStmt verified successfully and is  
 3454 found to be trustworthy, then register the new credential with the  
 3455 account that was denoted in the options.user passed to create(), by  
 3456 associating it with the credentialId and credentialPublicKey in the  
 3457 attestedCredentialData in authData, as appropriate for the Relying  
 3458 Party's system.  
 3459 19. If the attestation statement attStmt successfully verified but is  
 3460 not trustworthy per step 16 above, the Relying Party SHOULD fail  
 3461 the registration ceremony.  
 3462 NOTE: However, if permitted by policy, the Relying Party MAY  
 3463 register the credential ID and credential public key but treat the  
 3464 credential as one with self attestation (see 6.3.3 Attestation  
 3465 Types). If doing so, the Relying Party is asserting there is no  
 3466 cryptographic proof that the public key credential has been  
 3467 generated by a particular authenticator model. See [FIDOsecRef] and  
 3468 [UAFProtocol] for a more detailed discussion.  
 3469

3470 Verification of attestation objects requires that the Relying Party has  
 3471 a trusted method of determining acceptable trust anchors in step 15  
 3472 above. Also, if certificates are being used, the Relying Party **MUST**  
 3473 have access to certificate status information for the intermediate CA  
 3474 certificates. The Relying Party **MUST** also be able to build the  
 3475 attestation certificate chain if the client did not provide this chain  
 3476 in the attestation information.  
 3477

3478 7.2. Verifying an authentication assertion  
 3479

3480 When verifying a given PublicKeyCredential structure (credential) and  
 3481 an AuthenticationExtensionsClientOutputs structure  
 3482 clientExtensionResults, as part of an authentication ceremony, the  
 3483 Relying Party **MUST** proceed as follows:  
 3484

- 3485 1. If the allowCredentials option was given when this authentication  
 3486 ceremony was initiated, verify that credential.id identifies one of  
 3487 the public key credentials that were listed in allowCredentials.
- 3488 2. If credential.response.userHandle is present, verify that the user  
 3489 identified by this value is the owner of the public key credential  
 identified by credential.id.

base64url encoding is inappropriate for your use case), look up the corresponding credential public key.

2. Let `cData`, `aData` and `sig` denote the value of credential's response's `clientDataJSON`, `authenticatorData`, and signature respectively.
3. Perform JSON deserialization on `cData` to extract the client data `C` used for the signature.
4. Verify that the type in `C` is the string `webauthn.get`.
5. Verify that the challenge member of `C` matches the challenge that was sent to the authenticator in the `PublicKeyCredentialRequestOptions` passed to the `get()` call.
6. Verify that the origin member of `C` matches the Relying Party's origin.
7. Verify that the `tokenBindingId` member of `C` (if present) matches the Token Binding ID for the TLS connection over which the signature was obtained.
8. Verify that the `clientExtensions` member of `C` is a subset of the extensions requested by the Relying Party and that the `authenticatorExtensions` in `C` is also a subset of the extensions requested by the Relying Party.
9. Verify that the `rpldHash` in `aData` is the SHA-256 hash of the RP ID expected by the Relying Party.
10. Let `hash` be the result of computing a hash over the `cData` using the algorithm represented by the `hashAlgorithm` member of `C`.
11. Using the credential public key looked up in step 1, verify that

`sig` is a valid signature over the binary concatenation of `aData` and `hash`.

12. If the signature counter value `adata.signCount` is nonzero or the value stored in conjunction with credential's id attribute is nonzero, then run the following substep:
  - + If the signature counter value `adata.signCount` is greater than the signature counter value stored in conjunction with credential's id attribute. Update the stored signature counter value, associated with credential's id attribute, to be the value of `adata.signCount`.
  - less than or equal to the signature counter value stored in conjunction with credential's id attribute. This is an signal that the authenticator may be cloned, i.e. at least two copies of the credential

3. Using credential's id attribute (or the corresponding `rawId`, if base64url encoding is inappropriate for your use case), look up the corresponding credential public key.
4. Let `cData`, `aData` and `sig` denote the value of credential's response's `clientDataJSON`, `authenticatorData`, and signature respectively.
5. Let `JSONtext` be the result of running UTF-8 decode on the value of `cData`.
  - Note: Using any implementation of UTF-8 decode is acceptable as long as it yields the same result as that yielded by the UTF-8 decode algorithm. In particular, any leading byte order mark (BOM) MUST be stripped.
6. Let `C`, the client data claimed as used for the signature, be the result of running an implementation-specific JSON parser on `JSONtext`.
  - Note: `C` may be any implementation-specific data structure representation, as long as `C`'s components are referenceable, as required by this algorithm.
7. Verify that the value of `C.type` is the string `webauthn.get`.
8. Verify that the value of `C.challenge` matches the challenge that was sent to the authenticator in the `PublicKeyCredentialRequestOptions` passed to the `get()` call.
9. Verify that the value of `C.origin` matches the Relying Party's origin.
10. Verify that the value of `C.tokenBinding.status` matches the state of Token Binding for the TLS connection over which the attestation was obtained. If Token Binding was used on that TLS connection, also verify that `C.tokenBinding.id` matches the base64url encoding of the Token Binding ID for the connection.
11. Verify that the `rpldHash` in `aData` is the SHA-256 hash of the RP ID expected by the Relying Party.
12. If user verification is required for this assertion, verify that the User Verified bit of the flags in `aData` is set.
13. If user verification is not required for this assertion, verify that the User Present bit of the flags in `aData` is set.
14. Verify that the values of the client extension outputs in `clientExtensionResults` and the authenticator extension outputs in the extensions in `authData` are as expected, considering the client extension input values that were given as the extensions option in the `get()` call. In particular, any extension identifier values in the `clientExtensionResults` and the extensions in `authData` MUST be also be present as extension identifier values in the extensions member of options, i.e., no extensions are present that were not requested. In the general case, the meaning of "are as expected" is specific to the Relying Party and which extensions are in use.
  - Note: Since all extensions are OPTIONAL for both the client and the authenticator, the Relying Party MUST be prepared to handle cases where none or not all of the requested extensions were acted upon.
15. Let `hash` be the result of computing a hash over the `cData` using SHA-256.
16. Using the credential public key looked up in step 3, verify that `sig` is a valid signature over the binary concatenation of `aData` and `hash`.
17. If the signature counter value `adata.signCount` is nonzero or the value stored in conjunction with credential's id attribute is nonzero, then run the following sub-step:
  - + If the signature counter value `adata.signCount` is greater than the signature counter value stored in conjunction with credential's id attribute. Update the stored signature counter value, associated with credential's id attribute, to be the value of `adata.signCount`.
  - less than or equal to the signature counter value stored in conjunction with credential's id attribute. This is a signal that the authenticator may be cloned, i.e. at least two copies of the credential

3227 private key may exist and are being used in  
3228 parallel. Relying Parties should incorporate this  
3229 information into their risk scoring. Whether the  
3230 Relying Party updates the stored signature counter  
3231 value in this case, or not, or fails the  
3232 authentication ceremony or not, is Relying  
3233 Party-specific.  
3234  
3235 **13.** If all the above steps are successful, continue with the  
3236 authentication ceremony as appropriate. Otherwise, fail the  
3237 authentication ceremony.  
3238  
3239 **8. Defined Attestation Statement Formats**  
3240  
3241 WebAuthn supports pluggable attestation statement formats. This section  
3242 defines an initial set of such formats.  
3243  
3244 **8.1. Attestation Statement Format Identifiers**  
3245  
3246 Attestation statement formats are identified by a string, called a  
3247 attestation statement format identifier, chosen by the author of the  
3248 attestation statement format.  
3249  
3250 Attestation statement format identifiers SHOULD be registered per  
3251 [WebAuthn-Registries] "Registries for Web Authentication (WebAuthn)".  
3252 All registered attestation statement format identifiers are unique  
3253 amongst themselves as a matter of course.  
3254  
3255 Unregistered attestation statement format identifiers SHOULD use  
3256 lowercase reverse domain-name naming, using a domain name registered by  
3257 the developer, in order to assure uniqueness of the identifier. All  
3258 attestation statement format identifiers MUST be a maximum of 32 octets  
3259 in length and MUST consist only of printable USASCII characters,  
3260 excluding backslash and doublequote, i.e., VCHAR as defined in  
3261 [RFC5234] but without %x22 and %x5c.  
3262  
3263 Note: This means attestation statement format identifiers based on  
3264 domain names MUST incorporate only LDH Labels [RFC5890].  
3265  
3266 Implementations MUST match WebAuthn attestation statement format  
3267 identifiers in a case-sensitive fashion.  
3268  
3269 Attestation statement formats that may exist in multiple versions  
3270 SHOULD include a version in their identifier. In effect, different  
3271 versions are thus treated as different formats, e.g., packed2 as a new  
3272 version of the packed attestation statement format.  
3273  
3274 The following sections present a set of currently-defined and  
3275 registered attestation statement formats and their identifiers. The  
3276 up-to-date list of registered WebAuthn Extensions is maintained in the  
3277 IANA "WebAuthn Attestation Statement Format Identifier" registry  
3278 established by [WebAuthn-Registries].  
3279  
3280 **8.2. Packed Attestation Statement Format**  
3281  
3282 This is a WebAuthn optimized attestation statement format. It uses a  
3283 very compact but still extensible encoding method. It is implementable  
3284 by authenticators with limited resources (e.g., secure elements).  
3285  
3286 Attestation statement format identifier  
3287 packed  
3288  
3289 Attestation types supported  
3290 All  
3291  
3292 Syntax  
3293 The syntax of a Packed Attestation statement is defined by the  
3294 following CDDL:  
3295  
3296 `$$attStmtType // = (`

3558 private key may exist and are being used in  
3559 parallel. Relying Parties should incorporate this  
3560 information into their risk scoring. Whether the  
3561 Relying Party updates the stored signature counter  
3562 value in this case, or not, or fails the  
3563 authentication ceremony or not, is Relying  
3564 Party-specific.  
3565  
3566 **18.** If all the above steps are successful, continue with the  
3567 authentication ceremony as appropriate. Otherwise, fail the  
3568 authentication ceremony.  
3569  
3570 **8. Defined Attestation Statement Formats**  
3571  
3572 WebAuthn supports pluggable attestation statement formats. This section  
3573 defines an initial set of such formats.  
3574  
3575 **8.1. Attestation Statement Format Identifiers**  
3576  
3577 Attestation statement formats are identified by a string, called an  
3578 attestation statement format identifier, chosen by the author of the  
3579 attestation statement format.  
3580  
3581 Attestation statement format identifiers SHOULD be registered per  
3582 [WebAuthn-Registries] "Registries for Web Authentication (WebAuthn)".  
3583 All registered attestation statement format identifiers are unique  
3584 amongst themselves as a matter of course.  
3585  
3586 Unregistered attestation statement format identifiers SHOULD use  
3587 lowercase reverse domain-name naming, using a domain name registered by  
3588 the developer, in order to assure uniqueness of the identifier. All  
3589 attestation statement format identifiers MUST be a maximum of 32 octets  
3590 in length and MUST consist only of printable USASCII characters,  
3591 excluding backslash and doublequote, i.e., VCHAR as defined in  
3592 [RFC5234] but without %x22 and %x5c.  
3593  
3594 Note: This means attestation statement format identifiers based on  
3595 domain names MUST incorporate only LDH Labels [RFC5890].  
3596  
3597 Implementations MUST match WebAuthn attestation statement format  
3598 identifiers in a case-sensitive fashion.  
3599  
3600 Attestation statement formats that may exist in multiple versions  
3601 SHOULD include a version in their identifier. In effect, different  
3602 versions are thus treated as different formats, e.g., packed2 as a new  
3603 version of the packed attestation statement format.  
3604  
3605 The following sections present a set of currently-defined and  
3606 registered attestation statement formats and their identifiers. The  
3607 up-to-date list of registered WebAuthn Extensions is maintained in the  
3608 IANA "WebAuthn Attestation Statement Format Identifier" registry  
3609 established by [WebAuthn-Registries].  
3610  
3611 **8.2. Packed Attestation Statement Format**  
3612  
3613 This is a WebAuthn optimized attestation statement format. It uses a  
3614 very compact but still extensible encoding method. It is implementable  
3615 by authenticators with limited resources (e.g., secure elements).  
3616  
3617 Attestation statement format identifier  
3618 packed  
3619  
3620 Attestation types supported  
3621 All  
3622  
3623 Syntax  
3624 The syntax of a Packed Attestation statement is defined by the  
3625 following CDDL:  
3626  
3627 `$$attStmtType // = (`

```

3297         fmt: "packed",
3298         attStmt: packedStmtFormat
3299     )
3300
3301     packedStmtFormat = {
3302         alg: COSEAlgorithmIdentifier,
3303         sig: bytes,
3304         x5c: [ attestnCert: bytes, * (caCert: bytes) ]
3305     } //
3306
3307     for ED512)
3308         alg: COSEAlgorithmIdentifier, (-260 for ED256 / -261
3309         sig: bytes,
3310         ecdaaKeyId: bytes
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362

```

The semantics of the fields are as follows:

- alg A COSEAlgorithmIdentifier containing the identifier of the algorithm used to generate the attestation signature.
- sig A byte string containing the attestation signature.
- x5c The elements of this array contain the attestation certificate and its certificate chain, each encoded in X.509 format. The attestation certificate **must** be the first element in the array.
- ecdaaKeyId The identifier of the ECDAAs-Issuer public key. This is the BigNumberToB encoding of the component "c" of the ECDAAs-Issuer public key as defined section 3.3, step 3.5 in [FIDOecdaaAlgorithm].

**Signing procedure**

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

1. Let authenticatorData denote the authenticator data for the attestation, and let clientDataHash denote the hash of the serialized client data.
2. If Basic or Privacy CA attestation is in use, the authenticator produces the sig by concatenating authenticatorData and clientDataHash, and signing the result using an attestation private key selected through an authenticator-specific mechanism. It sets x5c to the certificate chain of the attestation public key and alg to the algorithm of the attestation private key.
3. If ECDAAs is in use, the authenticator produces sig by concatenating authenticatorData and clientDataHash, and signing the result using ECDAAs-Sign (see section 3.5 of [FIDOecdaaAlgorithm]) after selecting an ECDAAs-Issuer public key related to the ECDAAs signature private key through an authenticator-specific mechanism (see [FIDOecdaaAlgorithm]). It sets alg to the algorithm of the selected ECDAAs-Issuer public key and ecdaaKeyId to the identifier of the ECDAAs-Issuer public key (see above).
4. If self attestation is in use, the authenticator produces sig by concatenating authenticatorData and clientDataHash, and signing the result using the credential private key. It sets alg to the algorithm of the credential private key, and omits the other fields.

```

3628         fmt: "packed",
3629         attStmt: packedStmtFormat
3630     )
3631
3632     packedStmtFormat = {
3633         alg: COSEAlgorithmIdentifier,
3634         sig: bytes,
3635         x5c: [ attestnCert: bytes, * (caCert: bytes) ]
3636     } //
3637
3638     for ED512)
3639         alg: COSEAlgorithmIdentifier, (-260 for ED256 / -261
3640         sig: bytes,
3641         ecdaaKeyId: bytes
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697

```

The semantics of the fields are as follows:

- alg A COSEAlgorithmIdentifier containing the identifier of the algorithm used to generate the attestation signature.
- sig A byte string containing the attestation signature.
- x5c The elements of this array contain the attestation certificate and its certificate chain, each encoded in X.509 format. The attestation certificate **MUST** be the first element in the array.
- ecdaaKeyId The identifier of the ECDAAs-Issuer public key. This is the BigNumberToB encoding of the component "c" of the ECDAAs-Issuer public key as defined section 3.3, step 3.5 in [FIDOecdaaAlgorithm].

**Signing procedure**

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

1. Let authenticatorData denote the authenticator data for the attestation, and let clientDataHash denote the hash of the serialized client data.
2. If Basic or AttCA attestation is in use, the authenticator produces the sig by concatenating authenticatorData and clientDataHash, and signing the result using an attestation private key selected through an authenticator-specific mechanism. It sets x5c to the certificate chain of the attestation public key and alg to the algorithm of the attestation private key.
3. If ECDAAs is in use, the authenticator produces sig by concatenating authenticatorData and clientDataHash, and signing the result using ECDAAs-Sign (see section 3.5 of [FIDOecdaaAlgorithm]) after selecting an ECDAAs-Issuer public key related to the ECDAAs signature private key through an authenticator-specific mechanism (see [FIDOecdaaAlgorithm]). It sets alg to the algorithm of the selected ECDAAs-Issuer public key and ecdaaKeyId to the identifier of the ECDAAs-Issuer public key (see above).
4. If self attestation is in use, the authenticator produces sig by concatenating authenticatorData and clientDataHash, and signing the result using the credential private key. It sets alg to the algorithm of the credential private key and omits the other fields.

3365 Verification procedure  
 3366 Given the verification procedure inputs attStmt,  
 3365 authenticatorData and clientDataHash, the verification procedure  
 3366 is as follows:  
 3367  
 3368 1. Verify that attStmt is valid CBOR conforming to the syntax  
 3366 defined above, and perform CBOR decoding on it to extract the  
 3365 contained fields.  
 3370 2. If x5c is present, this indicates that the attestation type is  
 3372 not ECDA. In this case:  
 3373 o Verify that sig is a valid signature over the  
 3374 concatenation of authenticatorData and clientDataHash  
 3375 using the attestation public key in x5c with the  
 3376 algorithm specified in alg.  
 3377 o Verify that x5c meets the requirements in 8.2.1 Packed  
 3378 attestation statement certificate requirements.  
 3379 o If x5c contains an extension with OID 1.3.6.1.4.1.45724.1  
 3380 1.4 (id-fido-gen-ce-aaguid) verify that the value of this  
 3381 extension matches the aaguid in authenticatorData.  
 3382  
 3383 o If successful, return attestation type Basic and  
 3384 attestation trust path x5c.  
 3385 3. If ecdaaKeyId is present, then the attestation type is ECDA.  
 3386 In this case:  
 3387 o Verify that sig is a valid signature over the  
 3388 concatenation of authenticatorData and clientDataHash  
 3389 using ECDA-Verify with ECDA-Issuer public key  
 3390 identified by ecdaaKeyId (see [FIDOEcdaaAlgorithm]).  
 3391 o If successful, return attestation type ECDA and  
 3392 attestation trust path ecdaaKeyId.  
 3393 4. If neither x5c nor ecdaaKeyId is present, self attestation is  
 3394 in use.  
 3395 o Validate that alg matches the algorithm of the  
 3396 credentialPublicKey in authenticatorData.  
 3397 o Verify that sig is a valid signature over the  
 3398 concatenation of authenticatorData and clientDataHash  
 3399 using the credential public key with alg.  
 3400 o If successful, return attestation type Self and empty  
 3401 attestation trust path.  
 3402  
 3403 8.2.1. Packed attestation statement certificate requirements  
 3404 The attestation certificate MUST have the following fields/extensions:  
 3405 \* Version must be set to 3.  
 3406  
 3407 \* Subject field MUST be set to:  
 3408  
 3409 Subject-C  
 Country where the Authenticator vendor is incorporated  
 3410  
 3411 Subject-O  
 3412 Legal name of the Authenticator vendor  
 3413  
 3414 Subject-OU  
 3415 Authenticator Attestation  
 3416  
 3417 Subject-CN  
 3418 No stipulation.  
 3419  
 3420 \* If the related attestation root certificate is used for multiple  
 3421 authenticator models, the Extension OID 1.3.6.1.4.1.45724.1.1.4  
 3422 (id-fido-gen-ce-aaguid) MUST be present, containing the AAGUID as  
 3423 value.

3698 Verification procedure  
 3699 Given the verification procedure inputs attStmt,  
 3700 authenticatorData and clientDataHash, the verification procedure  
 3701 is as follows:  
 3702  
 3703 1. Verify that attStmt is valid CBOR conforming to the syntax  
 3704 defined above and perform CBOR decoding on it to extract the  
 3705 contained fields.  
 3706 2. If x5c is present, this indicates that the attestation type is  
 3707 not ECDA. In this case:  
 3708 o Verify that sig is a valid signature over the  
 3709 concatenation of authenticatorData and clientDataHash  
 3710 using the attestation public key in x5c with the  
 3711 algorithm specified in alg.  
 3712 o Verify that x5c meets the requirements in 8.2.1 Packed  
 3713 attestation statement certificate requirements.  
 3714 o If x5c contains an extension with OID  
 3715 1.3.6.1.4.1.45724.1.1.4 (id-fido-gen-ce-aaguid) verify  
 3716 that the value of this extension matches the aaguid in  
 3717 authenticatorData.  
 3718  
 3719 o If successful, return attestation type Basic and  
 3720 attestation trust path x5c.  
 3721 3. If ecdaaKeyId is present, then the attestation type is ECDA.  
 3722 In this case:  
 3723 o Verify that sig is a valid signature over the  
 3724 concatenation of authenticatorData and clientDataHash  
 3725 using ECDA-Verify with ECDA-Issuer public key  
 3726 identified by ecdaaKeyId (see [FIDOEcdaaAlgorithm]).  
 3727 o If successful, return attestation type ECDA and  
 3728 attestation trust path ecdaaKeyId.  
 3729 4. If neither x5c nor ecdaaKeyId is present, self attestation is  
 3730 in use.  
 3731 o Validate that alg matches the algorithm of the  
 3732 credentialPublicKey in authenticatorData.  
 3733 o Verify that sig is a valid signature over the  
 3734 concatenation of authenticatorData and clientDataHash  
 3735 using the credential public key with alg.  
 3736 o If successful, return attestation type Self and empty  
 3737 attestation trust path.  
 3738  
 3739 8.2.1. Packed attestation statement certificate requirements  
 3740 The attestation certificate MUST have the following fields/extensions:  
 3741 \* Version MUST be set to 3 (which is indicated by an ASN.1 INTEGER  
 3742 with value 2).  
 3743 \* Subject field MUST be set to:  
 3744  
 3745 Subject-C  
 3746 ISO 3166 code specifying the country where the  
 3747 Authenticator vendor is incorporated (PrintableString)  
 3748  
 3749 Subject-O  
 3750 Legal name of the Authenticator vendor (UTF8String)  
 3751  
 3752 Subject-OU  
 3753 Literal string "Authenticator Attestation" (UTF8String)  
 3754  
 3755 Subject-CN  
 3756 A UTF8String of the vendor's choosing  
 3757  
 3758 \* If the related attestation root certificate is used for multiple  
 3759 authenticator models, the Extension OID 1.3.6.1.4.1.45724.1.1.4  
 3760 (id-fido-gen-ce-aaguid) MUST be present, containing the AAGUID as a  
 3761 16-byte OCTET STRING. The extension MUST NOT be marked as critical.  
 3762 Note that an X.509 Extension encodes the DER-encoding of the value  
 3763 in an OCTET STRING. Thus, the AAGUID must be wrapped in two OCTET  
 3764 STRINGS to be valid. Here is a sample, encoded Extension structure:  
 3765  
 3766 30 21 -- SEQUENCE  
 3767 06 0b 2b 06 01 04 01 82 e5 1c 01 01 04 -- 1.3.6.1.4.1.45724.1.1.4  
 04 12 -- OCTET STRING

3424 \* The Basic Constraints extension MUST have the CA component set to  
 3425 false  
 3426 \* An Authority Information Access (AIA) extension with entry  
 3427 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are  
 3428 both **optional** as the status of many attestation certificates is  
 3429 available through authenticator metadata services. See, for  
 3430 example, the FIDO Metadata Service [FIDOMetadataService].  
 3431

### 8.3. TPM Attestation Statement Format

3432 This attestation statement format is generally used by authenticators  
 3433 that use a Trusted Platform Module as their cryptographic engine.

3434 Attestation statement format identifier  
 3435 tpm

3436 Attestation types supported  
 3437 **Privacy** CA, ECDAA

3438 Syntax  
 3439 The syntax of a TPM Attestation statement is as follows:

```
3440 $$attStmtType // = (  

    3441     fmt: "tpm",  

    3442     attStmt: tpmStmtFormat  

    3443 )
```

```
3444 tpmStmtFormat = {  

    3445     ver: "2.0",  

    3446     (  

    3447         alg: COSEAlgorithmIdentifier,  

    3448         x5c: [ aikCert: bytes, * (caCert: bytes) ]  

    3449     ) //  

    3450     alg: COSEAlgorithmIdentifier, (-260 for ED256 / -26  

    3451     1 for ED512)  

    3452     ecdaaKeyId: bytes  

    3453 ),  

    3454     sig: bytes,  

    3455     certInfo: bytes,  

    3456     pubArea: bytes  

    3457 }
```

3458 The semantics of the above fields are as follows:

3459 ver  
 3460 The version of the TPM specification to which the  
 3461 signature conforms.

3462 alg  
 3463 A COSEAlgorithmIdentifier containing the identifier of the  
 3464 algorithm used to generate the attestation signature.

3465 x5c  
 3466 The AIK certificate used for the attestation and its  
 3467 certificate chain, in X.509 encoding.

3468 ecdaaKeyId  
 3469 The identifier of the ECDAA-Issuer public key. This is the  
 3470 BigIntegerToB encoding of the component "c" as defined  
 3471 section 3.3, step 3.5 in [FIDOEcdaaAlgorithm].

3472 sig  
 3473 The attestation signature, in the form of a TPMT\_SIGNATURE  
 3474 structure as specified in [TPMv2-Part2] section 11.3.4.

3485

```
3768 04 10 -- OCTET STRING  

    3769 cd 8c 39 5c 26 ed ee de -- AAGUID  

    3770 65 3b 00 79 7d 03 ca 3c
```

3771 \* The Basic Constraints extension MUST have the CA component set to  
 3772 false.  
 3773 \* An Authority Information Access (AIA) extension with entry  
 3774 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are  
 3775 both **OPTIONAL** as the status of many attestation certificates is  
 3776 available through authenticator metadata services. See, for  
 3777 example, the FIDO Metadata Service [FIDOMetadataService].  
 3778

### 8.3. TPM Attestation Statement Format

3779 This attestation statement format is generally used by authenticators  
 3780 that use a Trusted Platform Module as their cryptographic engine.

3781 Attestation statement format identifier  
 3782 tpm

3783 Attestation types supported  
 3784 **AttCA**, ECDAA

3785 Syntax  
 3786 The syntax of a TPM Attestation statement is as follows:

```
3787 $$attStmtType // = (  

    3788     fmt: "tpm",  

    3789     attStmt: tpmStmtFormat  

    3790 )
```

```
3791 tpmStmtFormat = {  

    3792     ver: "2.0",  

    3793     (  

    3794         alg: COSEAlgorithmIdentifier,  

    3795         x5c: [ aikCert: bytes, * (caCert: bytes) ]  

    3796     ) //  

    3797     alg: COSEAlgorithmIdentifier, (-260 for ED256 / -26  

    3798     1 for ED512)  

    3799     ecdaaKeyId: bytes  

    3800 ),  

    3801     sig: bytes,  

    3802     certInfo: bytes,  

    3803     pubArea: bytes  

    3804 }
```

3805 The semantics of the above fields are as follows:

3806 ver  
 3807 The version of the TPM specification to which the  
 3808 signature conforms.

3809 alg  
 3810 A COSEAlgorithmIdentifier containing the identifier of the  
 3811 algorithm used to generate the attestation signature.

3812 x5c  
 3813 The AIK certificate used for the attestation and its  
 3814 certificate chain, in X.509 encoding.

3815 ecdaaKeyId  
 3816 The identifier of the ECDAA-Issuer public key. This is the  
 3817 BigIntegerToB encoding of the component "c" as defined  
 3818 section 3.3, step 3.5 in [FIDOEcdaaAlgorithm].

3819 sig  
 3820 The attestation signature, in the form of a TPMT\_SIGNATURE  
 3821 structure as specified in [TPMv2-Part2] section 11.3.4.

3837

3490 certInfo  
 3491 The TPMS\_ATTEST structure over which the above signature  
 3492 was computed, as specified in [TPMv2-Part2] section  
 3493 10.12.8.  
 3494  
 3495 pubArea  
 3496 The TPMT\_PUBLIC structure (see [TPMv2-Part2] section  
 3497 12.2.4) used by the TPM to represent the credential public  
 3498 key.  
 3499  
 3500 Signing procedure  
 3501 Let authenticatorData denote the authenticator data for the  
 3502 attestation, and let clientDataHash denote the hash of the  
 3503 serialized client data.  
 3504  
 3505 Concatenate authenticatorData and clientDataHash to form  
 3506 attToBeSigned.  
 3507  
 3508 Generate a signature using the procedure specified in  
 3509 [TPMv2-Part3] Section 18.2, using the attestation private key  
 3510 and setting the extraData parameter to the digest of  
 3511 attToBeSigned using the hash algorithm corresponding to the  
 3512 "alg" signature algorithm. (For the "RS256" algorithm, this  
 3513 would be a SHA-256 digest.)  
 3514  
 3515 Set the pubArea field to the public area of the credential  
 3516 public key, the certInfo field to the output parameter of the  
 3517 same name, and the sig field to the signature obtained from the  
 3518 above procedure.  
 3519  
 3520 Verification procedure  
 3521 Given the verification procedure inputs attStmt,  
 3522 authenticatorData and clientDataHash, the verification procedure  
 3523 is as follows:  
 3524  
 3525 Verify that attStmt is valid CBOR conforming to the syntax  
 3526 defined above, and perform CBOR decoding on it to extract the  
 3527 contained fields.  
 3528  
 3529 Verify that the public key specified by the parameters and  
 3530 unique fields of pubArea is identical to the credentialPublicKey  
 3531 in the attestedCredentialData in authenticatorData.  
 3532  
 3533 Concatenate authenticatorData and clientDataHash to form  
 3534 attToBeSigned.  
 3535  
 3536 Validate that certInfo is valid:  
 3537  
 3538 + Verify that magic is set to TPM\_GENERATED\_VALUE.  
 3539 + Verify that type is set to TPM\_ST\_ATTEST\_CERTIFY.  
 3540 + Verify that extraData is set to the hash of attToBeSigned  
 3541 using the hash algorithm employed in "alg".  
 3542 + Verify that attested contains a TPMS\_CERTIFY\_INFO structure,  
 3543 whose name field contains a valid Name for pubArea, as  
 3544 computed using the algorithm in the nameAlg field of pubArea  
 3545 using the procedure specified in [TPMv2-Part1] section 16.  
 3546  
 3547 If x5c is present, this indicates that the attestation type is  
 3548 not ECDSA. In this case:  
 3549  
 3550 + Verify the sig is a valid signature over certInfo using the  
 3551 attestation public key in x5c with the algorithm specified in  
 3552 alg.  
 3553 + Verify that x5c meets the requirements in 8.3.1 TPM  
 3554 attestation statement certificate requirements.

3838 certInfo  
 3839 The TPMS\_ATTEST structure over which the above signature  
 3840 was computed, as specified in [TPMv2-Part2] section  
 3841 10.12.8.  
 3842  
 3843 pubArea  
 3844 The TPMT\_PUBLIC structure (see [TPMv2-Part2] section  
 3845 12.2.4) used by the TPM to represent the credential public  
 3846 key.  
 3847  
 3848 Signing procedure  
 3849 Let authenticatorData denote the authenticator data for the  
 3850 attestation, and let clientDataHash denote the hash of the  
 3851 serialized client data.  
 3852  
 3853 Concatenate authenticatorData and clientDataHash to form  
 3854 attToBeSigned.  
 3855  
 3856 Generate a signature using the procedure specified in  
 3857 [TPMv2-Part3] Section 18.2, using the attestation private key  
 3858 and setting the extraData parameter to the digest of  
 3859 attToBeSigned using the hash algorithm corresponding to the  
 3860 "alg" signature algorithm. (For the "RS256" algorithm, this  
 3861 would be a SHA-256 digest.)  
 3862  
 3863 Set the pubArea field to the public area of the credential  
 3864 public key, the certInfo field to the output parameter of the  
 3865 same name, and the sig field to the signature obtained from the  
 3866 above procedure.  
 3867  
 3868 Verification procedure  
 3869 Given the verification procedure inputs attStmt,  
 3870 authenticatorData and clientDataHash, the verification procedure  
 3871 is as follows:  
 3872  
 3873 Verify that attStmt is valid CBOR conforming to the syntax  
 3874 defined above and perform CBOR decoding on it to extract the  
 3875 contained fields.  
 3876  
 3877 Verify that the public key specified by the parameters and  
 3878 unique fields of pubArea is identical to the credentialPublicKey  
 3879 in the attestedCredentialData in authenticatorData.  
 3880  
 3881 Concatenate authenticatorData and clientDataHash to form  
 3882 attToBeSigned.  
 3883  
 3884 Validate that certInfo is valid:  
 3885  
 3886 + Verify that magic is set to TPM\_GENERATED\_VALUE.  
 3887 + Verify that type is set to TPM\_ST\_ATTEST\_CERTIFY.  
 3888 + Verify that extraData is set to the hash of attToBeSigned  
 3889 using the hash algorithm employed in "alg".  
 3890 + Verify that attested contains a TPMS\_CERTIFY\_INFO structure as  
 3891 specified in [TPMv2-Part2] section 10.12.3, whose name field  
 3892 contains a valid Name for pubArea, as computed using the  
 3893 algorithm in the nameAlg field of pubArea using the procedure  
 3894 specified in [TPMv2-Part1] section 16.  
 3895 + Note that the remaining fields in the "Standard Attestation  
 3896 Structure" [TPMv2-Part1] section 31.2, i.e., qualifiedSigner,  
 3897 clockInfo and firmwareVersion are ignored. These fields MAY be  
 3898 used as an input to risk engines.  
 3899  
 3900 If x5c is present, this indicates that the attestation type is  
 3901 not ECDSA. In this case:  
 3902  
 3903 + Verify the sig is a valid signature over certInfo using the  
 3904 attestation public key in x5c with the algorithm specified in  
 3905 alg.  
 3906 + Verify that x5c meets the requirements in 8.3.1 TPM  
 3907 attestation statement certificate requirements.

3555 + If x5c contains an extension with OID 1 3 6 1 4 1 45724 1 1 4  
 3556 (id-fido-gen-ce-aaguid) verify that the value of this  
 3557 extension matches the aaguid in authenticatorData.  
 3558 + If successful, return attestation type **Privacy CA and**  
 3559 **attestation** trust path x5c.  
 3560  
 3561 If ecdaaKeyId is present, then the attestation type is ECDA.  
 3562  
 3563 + Perform ECDA-Verify on sig to verify that it is a valid  
 3564 signature over certInfo (see [FIDOecdaaAlgorithm]).  
 3565 + If successful, return attestation type ECDA and the  
 3566 identifier of the ECDA-Issuer public key ecdaaKeyId.

### 8.3.1. TPM attestation statement certificate requirements

TPM attestation certificate MUST have the following fields/extensions:

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

### 8.4. Android Key Attestation Statement Format

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

Attestation statement format identifier  
android-key

Attestation types supported  
Basic **Attestation**

#### Syntax

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

```

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

```

```

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

```

#### Signing procedure

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

3908 + If x5c contains an extension with OID 1 3 6 1 4 1 45724 1 1 4  
 3909 (id-fido-gen-ce-aaguid) verify that the value of this  
 3910 extension matches the aaguid in authenticatorData.  
 3911 + If successful, return attestation type **AttCA and attestation**  
 3912 **trust path** x5c.  
 3913  
 3914 If ecdaaKeyId is present, then the attestation type is ECDA.  
 3915  
 3916 + Perform ECDA-Verify on sig to verify that it is a valid  
 3917 signature over certInfo (see [FIDOecdaaAlgorithm]).  
 3918 + If successful, return attestation type ECDA and the  
 3919 identifier of the ECDA-Issuer public key ecdaaKeyId.

### 8.3.1. TPM attestation statement certificate requirements

TPM attestation certificate MUST have the following fields/extensions:

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

### 8.4. Android Key Attestation Statement Format

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

Attestation statement format identifier  
android-key

Attestation types supported  
Basic

#### Syntax

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

```

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

```

```

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

```

#### Signing procedure

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

3625 serialized client data.  
 3626  
 3627  
 3628 Request an Android Key Attestation by calling  
 3629 "keyStore.getCertificateChain(myKeyUUID)" providing  
 3630 clientDataHash as the challenge value (e.g., by using  
 3631 setAttestationChallenge). Set x5c to the returned value.  
 3632  
 3633 The authenticator produces sig by concatenating  
 3634 authenticatorData and clientDataHash, and signing the result  
 3635 using the credential private key. It sets alg to the algorithm  
 3636 of the signature format.  
 3637  
 3638 Verification procedure  
 3639 Given the verification procedure inputs attStmt,  
 3640 authenticatorData and clientDataHash, the verification procedure  
 3641 is as follows:  
 3642  
 3643 + Verify that attStmt is valid CBOR conforming to the syntax  
 3644 defined above, and perform CBOR decoding on it to extract the  
 3645 contained fields.  
 3646 + Verify that the public key in the first certificate in the  
 3647 series of certificates represented by the signature matches  
 3648 the credentialPublicKey in the attestedCredentialData in  
 authenticatorData.  
 3649  
 3650 + Verify that in the attestation certificate extension data:  
 3651 o The value of the attestationChallenge field is identical  
 3652 to the concatenation of authenticatorData and  
 3653 clientDataHash.  
 3654 o The AuthorizationList.allApplications field is not  
 3655 present, since PublicKeyCredentials must be bound to the  
 3656 RP ID.  
 3657 o The value in the AuthorizationList.origin field is equal  
 3658 to KM\_TAG\_GENERATED.  
 3659 o The value in the AuthorizationList.purpose field is equal  
 3660 to KM\_PURPOSE\_SIGN.  
 3661 + If successful, return attestation type Basic with the  
 3662 attestation trust path set to the entire attestation  
 3663 statement.  
 3664  
 3665 8.5. Android SafetyNet Attestation Statement Format  
 3666  
 3667 When the authenticator in question is a platform-provided Authenticator  
 3668 on certain Android platforms, the attestation statement is based on the  
 3669 SafetyNet API. In this case the authenticator data is completely  
 3670 controlled by the caller of the SafetyNet API (typically an application  
 3671 running on the Android platform) and the attestation statement only  
 3672 provides some statements about the health of the platform and the  
 3673 identity of the calling application. This attestation does not provide  
 3674 information regarding provenance of the authenticator and its  
 3675 associated data. Therefore platform-provided authenticators should make  
 3676 use of the Android Key Attestation when available, even if the  
 3677 SafetyNet API is also present.  
 3678  
 3679 Attestation statement format identifier  
 3680 android-safetynet  
 3681  
 3682 Attestation types supported  
 3683 Basic Attestation  
 3684  
 3685 Syntax  
 3686 The syntax of an Android Attestation statement is defined as  
 3687 follows:  
 3688  
 3689 \$\$attStmtType ::= (  
 3690     fmt: "android-safetynet",  
 3691     attStmt: safetynetStmtFormat  
 3692 )

3978 serialized client data.  
 3979  
 3980  
 3981 Request an Android Key Attestation by calling  
 3982 keyStore.getCertificateChain(myKeyUUID) providing clientDataHash  
 3983 as the challenge value (e.g., by using setAttestationChallenge).  
 3984 Set x5c to the returned value.  
 3985  
 3986 The authenticator produces sig by concatenating  
 3987 authenticatorData and clientDataHash, and signing the result  
 3988 using the credential private key. It sets alg to the algorithm  
 3989 of the signature format.  
 3990  
 3991 Verification procedure  
 3992 Given the verification procedure inputs attStmt,  
 3993 authenticatorData and clientDataHash, the verification procedure  
 3994 is as follows:  
 3995  
 3996 + Verify that attStmt is valid CBOR conforming to the syntax  
 3997 defined above and perform CBOR decoding on it to extract the  
 3998 contained fields.  
 3999 + Verify that sig is a valid signature over the concatenation of  
 4000 authenticatorData and clientDataHash using the public key in  
 4001 the first certificate in x5c with the algorithm specified in  
 4002 alg.  
 4003 + Verify that the public key in the first certificate in in x5c  
 4004 matches the credentialPublicKey in the attestedCredentialData  
 4005 in authenticatorData.  
 4006 + Verify that in the attestation certificate extension data:  
 4007 o The value of the attestationChallenge field is identical  
 4008 to clientDataHash.  
 4009 o The AuthorizationList.allApplications field is not  
 4010 present, since PublicKeyCredential must be bound to the  
 4011 RP ID.  
 4012 o The value in the AuthorizationList.origin field is equal  
 4013 to KM\_TAG\_GENERATED.  
 4014 o The value in the AuthorizationList.purpose field is equal  
 4015 to KM\_PURPOSE\_SIGN.  
 4016 + If successful, return attestation type Basic with the  
 4017 attestation trust path set to x5c.  
 4018  
 4019 8.5. Android SafetyNet Attestation Statement Format  
 4020  
 4021 When the authenticator in question is a platform-provided Authenticator  
 4022 on certain Android platforms, the attestation statement is based on the  
 4023 SafetyNet API. In this case the authenticator data is completely  
 4024 controlled by the caller of the SafetyNet API (typically an application  
 4025 running on the Android platform) and the attestation statement only  
 4026 provides some statements about the health of the platform and the  
 4027 identity of the calling application. This attestation does not provide  
 4028 information regarding provenance of the authenticator and its  
 4029 associated data. Therefore platform-provided authenticators should make  
 4030 use of the Android Key Attestation when available, even if the  
 4031 SafetyNet API is also present.  
 4032  
 4033 Attestation statement format identifier  
 4034 android-safetynet  
 4035  
 4036 Attestation types supported  
 4037 Basic  
 4038  
 4039 Syntax  
 4040 The syntax of an Android Attestation statement is defined as  
 4041 follows:  
 4042  
 4043 \$\$attStmtType ::= (  
 4044     fmt: "android-safetynet",  
 4045     attStmt: safetynetStmtFormat  
 4046 )

```

3692 safetynetStmtFormat = {
3693     ver: text,
3694     response: bytes
3695 }
3696
3697 The semantics of the above fields are as follows:
3698
3699 ver
3700 The version number of Google Play Services responsible for
3701 providing the SafetyNet API.
3702
3703 response
3704 The UTF-8 encoded result of the getJwsResult() call of the
3705 SafetyNet API. This value is a JWS [RFC7515] object (see
3706 SafetyNet online documentation) in Compact Serialization.
3707
3708 Signing procedure
3709 Let authenticatorData denote the authenticator data for the
3710 attestation, and let clientDataHash denote the hash of the
3711 serialized client data.
3712
3713 Concatenate authenticatorData and clientDataHash to form
3714 attToBeSigned.
3715
3716 Request a SafetyNet attestation, providing attToBeSigned as the
3717 nonce value. Set response to the result, and ver to the version
3718 of Google Play Services running in the authenticator.
3719
3720 Verification procedure
3721 Given the verification procedure inputs attStmt,
3722 authenticatorData and clientDataHash, the verification procedure
3723 is as follows:
3724
3725 + Verify that attStmt is valid CBOR conforming to the syntax
3726 defined above, and perform CBOR decoding on it to extract the
3727 contained fields.
3728 + Verify that response is a valid SafetyNet response of version
3729 ver.
3730 + Verify that the nonce in the response is identical to the
3731 concatenation of authenticatorData and clientDataHash.
3732 + Verify that the attestation certificate is issued to the
3733 hostname "attest.android.com" (see SafetyNet online
3734 documentation).
3735 + Verify that the ctsProfileMatch attribute in the payload of
3736 response is true.
3737 + If successful, return attestation type Basic with the
3738 attestation trust path set to the above attestation
3739 certificate.
3740
3741 8.6. FIDO U2F Attestation Statement Format
3742
3743 This attestation statement format is used with FIDO U2F authenticators
3744 using the formats defined in [FIDO-U2F-Message-Formats].
3745
3746 Attestation statement format identifier
3747 fido-u2f
3748
3749 Attestation types supported
3750 Basic Attestation, Self Attestation, Privacy CA
3751
3752 Syntax
3753 The syntax of a FIDO U2F attestation statement is defined as
3754 follows:
3755
3756 $$attStmtType ::= (
3757     fmt: "fido-u2f",
3758     attStmt: u2fStmtFormat
3759 )
3760
3761

```

```

4046 safetynetStmtFormat = {
4047     ver: text,
4048     response: bytes
4049 }
4050
4051 The semantics of the above fields are as follows:
4052
4053 ver
4054 The version number of Google Play Services responsible for
4055 providing the SafetyNet API.
4056
4057 response
4058 The UTF-8 encoded result of the getJwsResult() call of the
4059 SafetyNet API. This value is a JWS [RFC7515] object (see
4060 SafetyNet online documentation) in Compact Serialization.
4061
4062 Signing procedure
4063 Let authenticatorData denote the authenticator data for the
4064 attestation, and let clientDataHash denote the hash of the
4065 serialized client data.
4066
4067 Concatenate authenticatorData and clientDataHash to form
4068 attToBeSigned.
4069
4070 Request a SafetyNet attestation, providing attToBeSigned as the
4071 nonce value. Set response to the result, and ver to the version
4072 of Google Play Services running in the authenticator.
4073
4074 Verification procedure
4075 Given the verification procedure inputs attStmt,
4076 authenticatorData and clientDataHash, the verification procedure
4077 is as follows:
4078
4079 + Verify that attStmt is valid CBOR conforming to the syntax
4080 defined above and perform CBOR decoding on it to extract the
4081 contained fields.
4082 + Verify that response is a valid SafetyNet response of version
4083 ver.
4084 + Verify that the nonce in the response is identical to the
4085 concatenation of authenticatorData and clientDataHash.
4086 + Verify that the attestation certificate is issued to the
4087 hostname "attest.android.com" (see SafetyNet online
4088 documentation).
4089 + Verify that the ctsProfileMatch attribute in the payload of
4090 response is true.
4091 + If successful, return attestation type Basic with the
4092 attestation trust path set to the above attestation
4093 certificate.
4094
4095 8.6. FIDO U2F Attestation Statement Format
4096
4097 This attestation statement format is used with FIDO U2F authenticators
4098 using the formats defined in [FIDO-U2F-Message-Formats].
4099
4100 Attestation statement format identifier
4101 fido-u2f
4102
4103 Attestation types supported
4104 Basic, AttCA
4105
4106 Syntax
4107 The syntax of a FIDO U2F attestation statement is defined as
4108 follows:
4109
4110 $$attStmtType ::= (
4111     fmt: "fido-u2f",
4112     attStmt: u2fStmtFormat
4113 )
4114
4115

```

```

3762 u2fStmntFormat = {
3763     x5c: [ attestnCert: bytes, * (caCert: bytes) ],
3764     sig: bytes
3765 }
3766
3767 The semantics of the above fields are as follows:
3768
3769 x5c
3770 The elements of this array contain the attestation
3771 certificate and its certificate chain, each encoded in
3772 X.509 format. The attestation certificate must be the
3773 first element in the array.
3774
3775 sig
3776 The attestation signature. The signature was calculated
3777 over the (raw) U2F registration response message
3778 [FIDO-U2F-Message-Formats] received by the platform from
3779 the authenticator.
3780
3781 Signing procedure
3782 If the credential public key of the given credential is not of
3783 algorithm -7 ("ES256"), stop and return an error. Otherwise, let
3784 authenticatorData denote the authenticator data for the
3785 attestation, and let clientDataHash denote the hash of the
3786 serialized client data.
3787
3788 If clientDataHash is 256 bits long, set tbsHash to this value.
3789 Otherwise set tbsHash to the SHA-256 hash of clientDataHash.
3790
3791 Generate a Registration Response Message as specified in
3792 [FIDO-U2F-Message-Formats] section 4.3, with the application
3793 parameter set to the SHA-256 hash of the RP ID associated with
3794 the given credential, the challenge parameter set to tbsHash,
3795 and the key handle parameter set to the credential ID of the
3796 given credential. Set the raw signature part of this
3797 Registration Response Message (i.e., without the user public
3798 key, key handle, and attestation certificates) as sig and set
3799 the attestation certificates of the attestation public key as
3800 x5c.
3801
3802 Verification procedure
3803 Given the verification procedure inputs attStmnt,
3804 authenticatorData and clientDataHash, the verification procedure
3805 is as follows:
3806
3807 1. Verify that attStmnt is valid CBOR conforming to the syntax
3808 defined above, and perform CBOR decoding on it to extract the
3809 contained fields.
3810
3811 2. Let attCert be value of the first element of x5c. Let
3812 certificate public key be the public key conveyed by attCert.
3813 If certificate public key is not an Elliptic Curve (EC) public
3814 key over the P-256 curve, terminate this algorithm and return
3815 an appropriate error.
3816
3817 3. Extract the claimed rpIdHash from authenticatorData, and the
3818 claimed credentialId and credentialPublicKey from
3819 authenticatorData.attestedCredentialData.
3820
3821 4. If clientDataHash is 256 bits long, set tbsHash to this value.
3822 Otherwise set tbsHash to the SHA-256 hash of clientDataHash.
3823
3824 5. Convert the COSE_KEY formatted credentialPublicKey (see
3825 Section 7 of [RFC8152]) to CTAP1/U2F public Key format
3826 [FIDO-CTAP].
3827     o Let publicKeyU2F represent the result of the conversion
3828 operation and set its first byte to 0x04. Note: This
3829 signifies uncompressed ECC key format.
3830     o Extract the value corresponding to the "-2" key
3831 (representing x coordinate) from credentialPublicKey,
3832 confirm its size to be of 32 bytes and concatenate it
3833 with publicKeyU2F. If size differs or "-2" key is not
3834 found, terminate this algorithm and return an appropriate
3835 error.

```

```

4116 u2fStmntFormat = {
4117     x5c: [ attestnCert: bytes, * (caCert: bytes) ],
4118     sig: bytes
4119 }
4120
4121 The semantics of the above fields are as follows:
4122
4123 x5c
4124 The elements of this array contain the attestation
4125 certificate and its certificate chain, each encoded in
4126 X.509 format. The attestation certificate MUST be the
4127 first element in the array.
4128
4129 sig
4130 The attestation signature. The signature was calculated
4131 over the (raw) U2F registration response message
4132 [FIDO-U2F-Message-Formats] received by the platform from
4133 the authenticator.
4134
4135 Signing procedure
4136 If the credential public key of the given credential is not of
4137 algorithm -7 ("ES256"), stop and return an error. Otherwise, let
4138 authenticatorData denote the authenticator data for the
4139 attestation, and let clientDataHash denote the hash of the
4140 serialized client data. (Since SHA-256 is used to hash the
4141 serialized client data, clientDataHash will be 32 bytes long.)
4142
4143 Generate a Registration Response Message as specified in
4144 [FIDO-U2F-Message-Formats] section 4.3, with the application
4145 parameter set to the SHA-256 hash of the RP ID associated with
4146 the given credential, the challenge parameter set to
4147 clientDataHash, and the key handle parameter set to the
4148 credential ID of the given credential. Set the raw signature
4149 part of this Registration Response Message (i.e., without the
4150 user public key, key handle, and attestation certificates) as
4151 sig and set the attestation certificates of the attestation
4152 public key as x5c.
4153
4154 Verification procedure
4155 Given the verification procedure inputs attStmnt,
4156 authenticatorData and clientDataHash, the verification procedure
4157 is as follows:
4158
4159 1. Verify that attStmnt is valid CBOR conforming to the syntax
4160 defined above and perform CBOR decoding on it to extract the
4161 contained fields.
4162
4163 2. Let attCert be the value of the first element of x5c. Let
4164 certificate public key be the public key conveyed by attCert.
4165 If certificate public key is not an Elliptic Curve (EC) public
4166 key over the P-256 curve, terminate this algorithm and return
4167 an appropriate error.
4168
4169 3. Extract the claimed rpIdHash from authenticatorData, and the
4170 claimed credentialId and credentialPublicKey from
4171 authenticatorData.attestedCredentialData.
4172
4173 4. Convert the COSE_KEY formatted credentialPublicKey (see
4174 Section 7 of [RFC8152]) to CTAP1/U2F public Key format
4175 [FIDO-CTAP].
4176     o Let publicKeyU2F represent the result of the conversion
4177 operation and set its first byte to 0x04. Note: This
4178 signifies uncompressed ECC key format.
4179     o Extract the value corresponding to the "-2" key
4180 (representing x coordinate) from credentialPublicKey,
4181 confirm its size to be of 32 bytes and concatenate it
4182 with publicKeyU2F. If size differs or "-2" key is not
4183 found, terminate this algorithm and return an appropriate
4184 error.

```

3832 o Extract the value corresponding to the "-3" key  
 3833 (representing y coordinate) from credentialPublicKey,  
 3834 confirm its size to be of 32 bytes and concatenate it  
 3835 with publicKeyU2F. If size differs or "-3" key is not  
 3836 found, terminate this algorithm and return an appropriate  
 3837 error.  
 3838 6. Let verificationData be the concatenation of (0x00 || rpIdHash  
 3839 || tbsHash || credentialId || publicKeyU2F) (see [Section 4.3](#)  
 3840 of [FIDO-U2F-Message-Formats]).  
 3841 7. Verify the sig using verificationData and certificate public  
 3842 key per [SEC1].  
 3843 8. If successful, return attestation type Basic with the  
 3844 attestation trust path set to x5c.  
 3845

## 3846 9. WebAuthn Extensions

3847 The mechanism for generating public key credentials, as well as  
 3848 requesting and generating Authentication assertions, as defined in 5  
 3849 Web Authentication API, can be extended to suit particular use cases.  
 3850 Each case is addressed by defining a registration extension and/or an  
 3851 authentication extension.  
 3852

3853 Every extension is a client extension, meaning that the extension  
 3854 involves communication with and processing by the client. Client  
 3855 extensions define the following steps and data:  
 3856 \* navigator.credentials.create() extension request parameters and  
 3857 response values for registration extensions.  
 3858 \* navigator.credentials.get() extension request parameters and  
 3859 response values for authentication extensions.  
 3860 \* Client extension processing for registration extensions and  
 3861 authentication extensions.  
 3862  
 3863

3864 When creating a public key credential or requesting an authentication  
 3865 assertion, a Relying Party can request the use of a set of extensions.  
 3866 These extensions will be invoked during the requested operation if they  
 3867 are supported by the client and/or the authenticator. The Relying Party  
 3868 sends the client extension input for each extension in the get() call  
 3869 (for authentication extensions) or create() call (for registration  
 3870 extensions) to the client platform. The client platform performs client

4182 o Extract the value corresponding to the "-3" key  
 4183 (representing y coordinate) from credentialPublicKey,  
 4184 confirm its size to be of 32 bytes and concatenate it  
 4185 with publicKeyU2F. If size differs or "-3" key is not  
 4186 found, terminate this algorithm and return an appropriate  
 4187 error.  
 4188 5. Let verificationData be the concatenation of (0x00 || rpIdHash  
 4189 || clientDataHash || credentialId || publicKeyU2F) (see  
 4190 [Section 4.3](#) of [FIDO-U2F-Message-Formats]).  
 4191 6. Verify the sig using verificationData and certificate public  
 4192 key per [SEC1].  
 4193 7. If successful, return attestation type Basic with the  
 4194 attestation trust path set to x5c.  
 4195

## 4196 8.7. None Attestation Statement Format

4197 The none attestation statement format is used to replace any  
 4198 authenticator-provided attestation statement when a Relying Party  
 4199 indicates it does not wish to receive attestation information, see  
 4200 5.4.6 Attestation Conveyance Preference enumeration (enum  
 4201 AttestationConveyancePreference).  
 4202

4203 Attestation statement format identifier  
 4204 none

4205 Attestation types supported  
 4206 None

4207 Syntax  
 4208 The syntax of a none attestation statement is defined as  
 4209 follows:

```
4210 $$attStmtType ::= (  

  4211     fmt: "none",  

  4212     attStmt: emptyMap  

  4213 )
```

4214 emptyMap = {}

4215 Signing procedure  
 4216 Return the fixed attestation statement defined above.

4217 Verification procedure  
 4218 Return attestation type None with an empty trust path.

## 4219 9. WebAuthn Extensions

4220 The mechanism for generating public key credentials, as well as  
 4221 requesting and generating Authentication assertions, as defined in 5  
 4222 Web Authentication API, can be extended to suit particular use cases.  
 4223 Each case is addressed by defining a registration extension and/or an  
 4224 authentication extension.  
 4225

4226 Every extension is a client extension, meaning that the extension  
 4227 involves communication with and processing by the client. Client  
 4228 extensions define the following steps and data:  
 4229 \* navigator.credentials.create() extension request parameters and  
 4230 response values for registration extensions.  
 4231 \* navigator.credentials.get() extension request parameters and  
 4232 response values for authentication extensions.  
 4233 \* Client extension processing for registration extensions and  
 4234 authentication extensions.  
 4235

4236 When creating a public key credential or requesting an authentication  
 4237 assertion, a Relying Party can request the use of a set of extensions.  
 4238 These extensions will be invoked during the requested operation if they  
 4239 are supported by the client and/or the authenticator. The Relying Party  
 4240 sends the client extension input for each extension in the get() call  
 4241 (for authentication extensions) or create() call (for registration  
 4242 extensions) to the client platform. The client platform performs client

3871 extension processing for each extension that it supports, and augments  
3872 the client data as specified by each extension, by including the  
3873 extension identifier and client extension output values.  
3874

3875 An extension can also be an authenticator extension, meaning that the  
3876 extension involves communication with and processing by the  
3877 authenticator. Authenticator extensions define the following steps and  
3878 data:  
3879 \* authenticatorMakeCredential extension request parameters and  
3880 response values for registration extensions.  
3881 \* authenticatorGetAssertion extension request parameters and response  
3882 values for authentication extensions.  
3883 \* Authenticator extension processing for registration extensions and  
3884 authentication extensions.  
3885

3886 For authenticator extensions, as part of the client extension  
3887 processing, the client also creates the CBOR authenticator extension  
3888 input value for each extension (often based on the corresponding client  
3889 extension input value), and passes them to the authenticator in the  
3890 create() call (for registration extensions) or the get() call (for  
3891 authentication extensions). These authenticator extension input values  
3892 are represented in CBOR and passed as name-value pairs, with the  
3893 extension identifier as the name, and the corresponding authenticator  
3894 extension input as the value. The authenticator, in turn, performs  
3895 additional processing for the extensions that it supports, and returns  
3896 the CBOR authenticator extension output for each as specified by the  
3897 extension. Part of the client extension processing for authenticator  
3898 extensions is to use the authenticator extension output as an input to  
3899 creating the client extension output.  
3900

3901 All WebAuthn extensions are **optional** for both clients and  
3902 authenticators. Thus, any extensions requested by a Relying Party **may**  
3903 be ignored by the client browser or OS and not passed to the  
3904 authenticator at all, or they **may** be ignored by the authenticator.  
3905 Ignoring an extension is never considered a failure in WebAuthn API  
3906 processing, so when Relying Parties include extensions with any API  
3907 calls, they **must** be prepared to handle cases where some or all of those  
3908 extensions are ignored.  
3909

3910 Clients wishing to support the widest possible range of extensions **may**  
3911 choose to pass through any extensions that they do not recognize to  
3912 authenticators, generating the authenticator extension input by simply  
3913 encoding the client extension input in CBOR. All WebAuthn extensions  
3914 **MUST** be defined in such a way that this implementation choice does not  
3915 endanger the user's security or privacy. For instance, if an extension  
3916 requires client processing, it could be defined in a manner that  
3917 ensures such a naive pass-through will produce a semantically invalid  
3918 authenticator extension input value, resulting in the extension being  
3919 ignored by the authenticator. Since all extensions are **optional**, this  
3920 will not cause a functional failure in the API operation. Likewise,  
3921 clients can choose to produce a client extension output value for an  
3922 extension that it does not understand by encoding the authenticator  
3923 extension output value into JSON, provided that the CBOR output uses  
3924 only types present in JSON.  
3925

4252 extension processing for each extension that it supports, and augments  
4253 the client data as specified by each extension, by including the  
4254 extension identifier and client extension output values.  
4255

4256 An extension can also be an authenticator extension, meaning that the  
4257 extension involves communication with and processing by the  
4258 authenticator. Authenticator extensions define the following steps and  
4259 data:  
4260 \* authenticatorMakeCredential extension request parameters and  
4261 response values for registration extensions.  
4262 \* authenticatorGetAssertion extension request parameters and response  
4263 values for authentication extensions.  
4264 \* Authenticator extension processing for registration extensions and  
4265 authentication extensions.  
4266

4267 For authenticator extensions, as part of the client extension  
4268 processing, the client also creates the CBOR authenticator extension  
4269 input value for each extension (often based on the corresponding client  
4270 extension input value), and passes them to the authenticator in the  
4271 create() call (for registration extensions) or the get() call (for  
4272 authentication extensions). These authenticator extension input values  
4273 are represented in CBOR and passed as name-value pairs, with the  
4274 extension identifier as the name, and the corresponding authenticator  
4275 extension input as the value. The authenticator, in turn, performs  
4276 additional processing for the extensions that it supports, and returns  
4277 the CBOR authenticator extension output for each as specified by the  
4278 extension. Part of the client extension processing for authenticator  
4279 extensions is to use the authenticator extension output as an input to  
4280 creating the client extension output.  
4281

4282 All WebAuthn extensions are **OPTIONAL** for both clients and  
4283 authenticators. Thus, any extensions requested by a Relying Party **MAY**  
4284 be ignored by the client browser or OS and not passed to the  
4285 authenticator at all, or they **MAY** be ignored by the authenticator.  
4286 Ignoring an extension is never considered a failure in WebAuthn API  
4287 processing, so when Relying Parties include extensions with any API  
4288 calls, they **MUST** be prepared to handle cases where some or all of those  
4289 extensions are ignored.  
4290

4291 Clients wishing to support the widest possible range of extensions **MAY**  
4292 choose to pass through any extensions that they do not recognize to  
4293 authenticators, generating the authenticator extension input by simply  
4294 encoding the client extension input in CBOR. All WebAuthn extensions  
4295 **MUST** be defined in such a way that this implementation choice does not  
4296 endanger the user's security or privacy. For instance, if an extension  
4297 requires client processing, it could be defined in a manner that  
4298 ensures such a naive pass-through will produce a semantically invalid  
4299 authenticator extension input value, resulting in the extension being  
4300 ignored by the authenticator. Since all extensions are **OPTIONAL**, this  
4301 will not cause a functional failure in the API operation. Likewise,  
4302 clients can choose to produce a client extension output value for an  
4303 extension that it does not understand by encoding the authenticator  
4304 extension output value into JSON, provided that the CBOR output uses  
4305 only types present in JSON.  
4306

4307 When clients choose to pass through extensions they do not recognize,  
4308 the JavaScript values in the client extension inputs are converted to  
4309 CBOR values in the authenticator extension inputs. When the JavaScript  
4310 value is an %ArrayBuffer%, it is converted to a CBOR byte array. When  
4311 the JavaScript value is a non-integer number, it is converted to a  
4312 64-bit CBOR floating point number. Otherwise, when the JavaScript type  
4313 corresponds to a JSON type, the conversion is done using the rules  
4314 defined in Section 4.2 of [RFC7049] (Converting from JSON to CBOR), but  
4315 operating on inputs of JavaScript type values rather than inputs of  
4316 JSON type values. Once these conversions are done, canonicalization of  
4317 the resulting CBOR **MUST** be performed using the CTAP2 canonical CBOR  
4318 encoding form.  
4319

4320 Likewise, when clients receive outputs from extensions they have passed  
4321 through that they do not recognize, the CBOR values in the

3926 The IANA "WebAuthn Extension Identifier" registry established by  
 3927 [WebAuthn-Registries] **should** be consulted for an up-to-date list of  
 3928 registered WebAuthn Extensions.  
 3929

3930 9.1. Extension Identifiers

3931  
 3932 Extensions are identified by a string, called an extension identifier,  
 3933 chosen by the extension author.  
 3934

3935 Extension identifiers SHOULD be registered per [WebAuthn-Registries]  
 3936 "Registries for Web Authentication (WebAuthn)". All registered  
 3937 extension identifiers are unique amongst themselves as a matter of  
 3938 course.  
 3939

3940 Unregistered extension identifiers **should** aim to be globally unique,  
 3941 e.g., by including the defining entity such as myCompany\_extension.  
 3942

3943 All extension identifiers MUST be a maximum of 32 octets in length and  
 3944 MUST consist only of printable USASCII characters, excluding backslash  
 3945 and doublequote, i.e., VCHAR as defined in [RFC5234] but without %x22  
 3946 and %x5c. Implementations MUST match WebAuthn extension identifiers in  
 3947 a case-sensitive fashion.  
 3948

3949 Extensions that may exist in multiple versions should take care to  
 3950 include a version in their identifier. In effect, different versions  
 3951 are thus treated as different extensions, e.g., myCompany\_extension\_01  
 3952

3953 10 Defined Extensions defines an initial set of extensions and their  
 3954 identifiers. See the IANA "WebAuthn Extension Identifier" registry  
 3955 established by [WebAuthn-Registries] for an up-to-date list of  
 3956 registered WebAuthn Extension Identifiers.  
 3957

3958 9.2. Defining extensions

3959

3960 A definition of an extension **must** specify an extension identifier, a  
 3961 client extension input argument to be sent via the get() or create()  
 3962 call, the client extension processing rules, and a client extension  
 3963 output value. If the extension communicates with the authenticator  
 3964 (meaning it is an authenticator extension), it **must** also specify the  
 3965 CBOR authenticator extension input argument sent via the  
 3966 authenticatorGetAssertion or authenticatorMakeCredential call, the  
 3967 authenticator extension processing rules, and the CBOR authenticator  
 3968 extension output value.  
 3969

3970 Any client extension that is processed by the client MUST return a  
 3971 client extension output value so that the Relying Party knows that the  
 3972 extension was honored by the client. Similarly, any extension that  
 3973 requires authenticator processing MUST return an authenticator  
 3974 extension output to let the Relying Party know that the extension was  
 3975 honored by the authenticator. If an extension does not otherwise  
 3976 require any result values, it SHOULD be defined as returning a JSON  
 3977 Boolean client extension output result, set to true to signify that the  
 3978 extension was understood and processed. Likewise, any authenticator  
 3979 extension that does not otherwise require any result values MUST return  
 3980 a value and SHOULD return a CBOR Boolean authenticator extension output  
 3981 result, set to true to signify that the extension was understood and

4322 authenticator extension outputs are converted to JavaScript values in  
 4323 the client extension outputs. When the CBOR value is a byte string, it  
 4324 is converted to a JavaScript %ArrayBuffer% (rather than a  
 4325 base64url-encoded string). Otherwise, when the CBOR type corresponds to  
 4326 a JSON type, the conversion is done using the rules defined in Section  
 4327 4.1 of [RFC7049] (Converting from CBOR to JSON), but producing outputs  
 4328 of JavaScript type values rather than outputs of JSON type values.  
 4329

4330 Note that some clients may choose to implement this pass-through  
 4331 capability under a feature flag. Supporting this capability can  
 4332 facilitate innovation, allowing authenticators to experiment with new  
 4333 extensions and Relying Parties to use them before there is explicit  
 4334 support for them in clients.  
 4335

4336 The IANA "WebAuthn Extension Identifier" registry established by  
 4337 [WebAuthn-Registries] **can** be consulted for an up-to-date list of  
 4338 registered WebAuthn Extensions.  
 4339

4340 9.1. Extension Identifiers

4341  
 4342 Extensions are identified by a string, called an extension identifier,  
 4343 chosen by the extension author.  
 4344

4345 Extension identifiers SHOULD be registered per [WebAuthn-Registries]  
 4346 "Registries for Web Authentication (WebAuthn)". All registered  
 4347 extension identifiers are unique amongst themselves as a matter of  
 4348 course.  
 4349

4350 Unregistered extension identifiers **SHOULD** aim to be globally unique,  
 4351 e.g., by including the defining entity such as myCompany\_extension.  
 4352

4353 All extension identifiers MUST be a maximum of 32 octets in length and  
 4354 MUST consist only of printable USASCII characters, excluding backslash  
 4355 and doublequote, i.e., VCHAR as defined in [RFC5234] but without %x22  
 4356 and %x5c. Implementations MUST match WebAuthn extension identifiers in  
 4357 a case-sensitive fashion.  
 4358

4359 Extensions that may exist in multiple versions should take care to  
 4360 include a version in their identifier. In effect, different versions  
 4361 are thus treated as different extensions, e.g., myCompany\_extension\_01  
 4362

4363 10 Defined Extensions defines an initial set of extensions and their  
 4364 identifiers. See the IANA "WebAuthn Extension Identifier" registry  
 4365 established by [WebAuthn-Registries] for an up-to-date list of  
 4366 registered WebAuthn Extension Identifiers.  
 4367

4368 9.2. Defining extensions

4369

4370 A definition of an extension **MUST** specify an extension identifier, a  
 4371 client extension input argument to be sent via the get() or create()  
 4372 call, the client extension processing rules, and a client extension  
 4373 output value. If the extension communicates with the authenticator  
 4374 (meaning it is an authenticator extension), it **MUST** also specify the  
 4375 CBOR authenticator extension input argument sent via the  
 4376 authenticatorGetAssertion or authenticatorMakeCredential call, the  
 4377 authenticator extension processing rules, and the CBOR authenticator  
 4378 extension output value.  
 4379

4380 Any client extension that is processed by the client MUST return a  
 4381 client extension output value so that the Relying Party knows that the  
 4382 extension was honored by the client. Similarly, any extension that  
 4383 requires authenticator processing MUST return an authenticator  
 4384 extension output to let the Relying Party know that the extension was  
 4385 honored by the authenticator. If an extension does not otherwise  
 4386 require any result values, it SHOULD be defined as returning a JSON  
 4387 Boolean client extension output result, set to true to signify that the  
 4388 extension was understood and processed. Likewise, any authenticator  
 4389 extension that does not otherwise require any result values MUST return  
 4390 a value and SHOULD return a CBOR Boolean authenticator extension output  
 4391 result, set to true to signify that the extension was understood and

processed.

### 9.3. Extending request parameters

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

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

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

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

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

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

### 9.4. Client extension processing

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

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

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

processed.

### 9.3. Extending request parameters

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

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

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

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

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

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

### 9.4. Client extension processing

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

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

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

4052 9.5. Authenticator extension processing  
 4053  
 4054 The CBOR authenticator extension input value of each processed  
 4055 authenticator extension is included in the extensions **data part** of the  
 4056 authenticator request. This part is a CBOR map, with CBOR extension  
 4057 identifier values as keys, and the CBOR authenticator extension input  
 4058 value of each extension as the value.  
 4059  
 4060 Likewise, the extension output is represented in the authenticator data  
 4061 as a CBOR map with CBOR extension identifiers as keys, and the CBOR  
 4062 authenticator extension output value of each extension as the value.  
 4063  
 4064 The authenticator extension processing rules are used create the  
 4065 authenticator extension output from the authenticator extension input,  
 4066 and possibly also other inputs, for each extension.  
 4067  
 4068 9.6. Example Extension  
 4069  
 4070 This section is not normative.  
 4071  
 4072 To illustrate the requirements above, consider a hypothetical  
 4073 registration extension and authentication extension "Geo". This  
 4074 extension, if supported, enables a geolocation location to be returned  
 4075 from the authenticator or client to the Relying Party.  
 4076  
 4077 The extension identifier is chosen as `webauthnExample_geo`. The client  
 4078 extension input is the constant value `true`, since the extension does  
 4079 not require the Relying Party to pass any particular information to the  
 4080 client, other than that it requests the use of the extension. The  
 4081 Relying Party sets this value in its request for an assertion:  
 4082 

```
var assertionPromise =  

  4083     navigator.credentials.get({  

  4084         publicKey: {  

  4085             // The challenge must be produced by the server, see the Security Co  

  4086             nsiderations  

  4087             challenge: new Uint8Array([11,103,35 /* 29 more random bytes generat  

  4088             ed by the server */]),  

  4089             allowCredentials: [], /* Empty filter */  

  4090             extensions: { 'webauthnExample_geo': true }  

  4091         }  

  4092     });
```

  
 4093  
 4094 The extension also requires the client to set the authenticator  
 4095 parameter to the fixed value `true`.  
 4096  
 4097 The extension requires the authenticator to specify its geolocation in  
 4098 the authenticator extension output, if known. The extension e.g.  
 4099 specifies that the location shall be encoded as a two-element array of  
 4100 floating point numbers, encoded with CBOR. An authenticator does this  
 4101 by including it in the authenticator data. As an example, authenticator  
 4102 data may be as follows (notation taken from [RFC7049]):  
 4103 

```
81 (hex) -- Flags, ED and UP both set.  

  4104 20 05 58 1F -- Signature counter  

  4105 A1 -- CBOR map of one element  

  4106 73 -- Key 1: CBOR text string of 19 byt  

  4107 es  

  4108 77 65 62 61 75 74 68 6E 45 78 61  

  4109 6D 70 6C 65 5F 67 65 6F -- "webauthnExample_geo" [=UTF-8 enc  

  4110 oded=] string  

  4111 82 -- Value 1: CBOR array of two elemen  

  4112 ts  

  4113 FA 42 82 1E B3 -- Element 1: Latitude as CBOR encod  

  4114 ed float  

  4115 FA C1 5F E3 7F -- Element 2: Longitude as CBOR enco  

  4116 ded float  

  4117  

  4118 The extension defines the client extension output to be the geolocation  

  4119 information, if known, as a GeoJSON [GeoJSON] point. The client  

  4120 constructs the following client data:  

  4121 {
```

4461 9.5. Authenticator extension processing  
 4462  
 4463 The CBOR authenticator extension input value of each processed  
 4464 authenticator extension is included in the extensions **parameter** of the  
 4465 authenticator `MakeCredential` and `authenticatorGetAssertion` operations.  
 4466 The extensions parameter is a CBOR map where each key is an extension  
 4467 identifier and the corresponding value is the authenticator extension  
 4468 input for that extension.  
 4469  
 4470 Likewise, the extension output is represented in the extensions part of  
 4471 the authenticator data. The extensions part of the authenticator data  
 4472 is a CBOR map where each key is an extension identifier and the  
 4473 corresponding value is the authenticator extension output for that  
 4474 extension.  
 4475  
 4476 For each supported extension, the authenticator extension processing  
 4477 rule for that extension is used create the authenticator extension  
 4478 output from the authenticator extension input and possibly also other  
 4479 inputs.

```

4122     'extensions': {
4123       'webauthnExample_geo': {
4124         'type': 'Point',
4125         'coordinates': [65.059962, -13.993041]
4126       }
4127     }
4128   }
4129 }

```

## 10. Defined Extensions

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

### 10.1. FIDO AppID Extension (appid)

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

Extension identifier  
appid

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

#### Client extension processing

**If rpId is present, return a DOMException whose name is "NotAllowedError", and terminate this algorithm (5.1.4.1 PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method).**

**Otherwise, replace the calculation of rpId in Step 6 of 5.1.4.1 PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method with the following procedure: The client uses the value of appid to perform the Appid validation procedure (as defined by [FIDO-APPID]). If valid, the value of rpId for all client processing should be replaced by the value of appid.**

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

```

4480
4481
4482
4483
4484
4485
4486
4487
4488
4489
4490
4491
4492
4493
4494
4495
4496
4497
4498
4499
4500
4501
4502
4503
4504
4505
4506
4507
4508
4509
4510
4511
4512

```

## 10. Defined Extensions

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

### 10.1. FIDO AppID Extension (appid)

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

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

Extension identifier  
appid

Client extension input  
A single **USVString** specifying a FIDO **AppID**.

```

4508 partial dictionary AuthenticationExtensionsClientInputs {
4509   USVString appid;
4510 };

```

#### Client extension processing

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

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

4170 Authenticator extension input  
4171 None.  
4172  
4173 Authenticator extension processing  
4174 None.  
4175  
4176 Authenticator extension output  
4177 None.  
4178  
4179  
4180 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
4181  
4182 This registration extension and authentication extension allows for a  
4183 simple form of transaction authorization. A Relying Party can specify a  
4184 prompt string, intended for display on a trusted device on the  
4185 authenticator.  
4186  
4187 Extension identifier  
4188 txAuthSimple  
4189  
4190 Client extension input  
4191 A single **JSON** string prompt.

4192  
4193 Client extension processing  
4194 None, except creating the authenticator extension input from the  
4195 client extension input.  
4196  
4197 Client extension output  
4198 Returns the authenticator extension output string UTF-8 decoded  
4199 into a **JSON** string

4200  
4201 Authenticator extension input  
4202 The client extension input encoded as a CBOR text string (major  
4203 type 3).  
4204

4205  
4206 Authenticator extension processing  
4207 The authenticator **MUST** display the prompt to the user before  
4208 performing either user verification or test of user presence.  
4209 The authenticator **may** insert line breaks if needed.  
4210  
4211 Authenticator extension output  
4212 A single CBOR string, representing the prompt as displayed  
4213 (including any eventual line breaks).

4214  
4215 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
4216  
4217 This registration extension and authentication extension allows images  
4218 to be used as transaction authorization prompts as well. This allows  
4219 authenticators without a font rendering engine to be used and also  
4220 supports a richer visual appearance.  
4221  
4222 Extension identifier  
txAuthGeneric

4538 partial dictionary AuthenticationExtensionsClientOutputs {  
4539 boolean appid;  
4540 };  
4541  
4542 Authenticator extension input  
4543 None.  
4544  
4545 Authenticator extension processing  
4546 None.  
4547  
4548 Authenticator extension output  
4549 None.  
4550  
4551 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
4552  
4553 This registration extension and authentication extension allows for a  
4554 simple form of transaction authorization. A Relying Party can specify a  
4555 prompt string, intended for display on a trusted device on the  
4556 authenticator.  
4557  
4558 Extension identifier  
4559 txAuthSimple  
4560  
4561 Client extension input  
4562 A single **USVString** prompt.  
4563  
4564 partial dictionary AuthenticationExtensionsClientInputs {  
4565 USVString txAuthSimple;  
4566 };  
4567  
4568 Client extension processing  
4569 None, except creating the authenticator extension input from the  
4570 client extension input.  
4571  
4572 Client extension output  
4573 Returns the authenticator extension output string UTF-8 decoded  
4574 into a **USVString**.  
4575  
4576 partial dictionary AuthenticationExtensionsClientOutputs {  
4577 USVString txAuthSimple;  
4578 };  
4579  
4580 Authenticator extension input  
4581 The client extension input encoded as a CBOR text string (major  
4582 type 3).  
4583  
4584 CDDL:  
4585 txAuthSimpleInput = (tstr)  
4586  
4587 Authenticator extension processing  
4588 The authenticator **MUST** display the prompt to the user before  
4589 performing either user verification or test of user presence.  
4590 The authenticator **MAY** insert line breaks if needed.  
4591  
4592 Authenticator extension output  
4593 A single CBOR string, representing the prompt as displayed  
4594 (including any eventual line breaks).  
4595  
4596 CDDL:  
4597 txAuthSimpleOutput = (tstr)  
4598  
4599 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
4600  
4601 This registration extension and authentication extension allows images  
4602 to be used as transaction authorization prompts as well. This allows  
4603 authenticators without a font rendering engine to be used and also  
4604 supports a richer visual appearance.  
4605  
4606 Extension identifier  
4607 txAuthGeneric

4223 Client extension input  
 4224 A **CBOR map** defined as follows:  
 4225  
 4226  
 4227 txAuthGenericArg = {  
 4228 content: **text**, // MIME-Type of the content, e.g.  
 4229 "image/png"  
 4230 content: **bytes**  
 4231 }

4232 Client extension processing  
 4233 None, except creating the authenticator extension input from the  
 4234 client extension input.  
 4235  
 4236 Client extension output  
 4237 Returns the **base64url encoding of the authenticator extension**  
 4238 **output value as a JSON string**  
 4239

4240 Authenticator extension input  
 4241 The client extension input encoded as a CBOR map.  
 4242  
 4243 Authenticator extension processing  
 4244 The authenticator **MUST** display the content to the user before  
 4245 performing either user verification or test of user presence.  
 4246 The authenticator **may** add other information below the content.  
 4247 No changes are allowed to the content itself, i.e., inside  
 4248 content boundary box.  
 4249  
 4250 Authenticator extension output  
 4251 The hash value of the content which was displayed. The  
 4252 authenticator **MUST** use the same hash algorithm as it uses for  
 4253 the signature itself.  
 4254  
 4255 10.4. Authenticator Selection Extension (authnSel)  
 4256  
 4257 This registration extension allows a Relying Party to guide the  
 4258 selection of the authenticator that will be leveraged when creating the  
 4259 credential. It is intended primarily for Relying Parties that wish to  
 4260 tightly control the experience around credential creation.  
 4261  
 4262 Extension identifier  
 4263 authnSel  
 4264  
 4265 Client extension input  
 4266 A sequence of AAGUIDs:  
 4267  
 4268 typedef sequence<AAGUID> AuthenticatorSelectionList;  
 4269  
 4270  
 4271 Each AAGUID corresponds to an authenticator model that is  
 4272 acceptable to the Relying Party for this credential creation.  
 4273 The list is ordered by decreasing preference.  
 4274  
 4275 An AAGUID is defined as an array containing the globally unique  
 4276 identifier of the authenticator model being sought.  
 4277  
 4278 typedef BufferSource AAGUID;  
 4279  
 4280 Client extension processing

4608 Client extension input  
 4609 A **JavaScript object** defined as follows:  
 4610  
 4611  
 4612 dictionary txAuthGenericArg {  
 4613 required **USVString** contentType; // MIME-Type of the content, e.g., "image  
 4614 /png"  
 4615 required **ArrayBuffer** content;  
 4616 }  
 4617  
 4618 partial dictionary AuthenticationExtensionsClientInputs {  
 4619 txAuthGenericArg txAuthGeneric;  
 4620 }  
 4621  
 4622 Client extension processing  
 4623 None, except creating the authenticator extension input from the  
 4624 client extension input.  
 4625  
 4626 Client extension output  
 4627 Returns the **authenticator extension output value as an**  
 4628 **ArrayBuffer**.  
 4629  
 4630 partial dictionary AuthenticationExtensionsClientOutputs {  
 4631 **ArrayBuffer** txAuthGeneric;  
 4632 }  
 4633  
 4634 Authenticator extension input  
 4635 The client extension input encoded as a CBOR map.  
 4636  
 4637 Authenticator extension processing  
 4638 The authenticator **MUST** display the content to the user before  
 4639 performing either user verification or test of user presence.  
 4640 The authenticator **MAY** add other information below the content.  
 4641 No changes are allowed to the content itself, i.e., inside  
 4642 content boundary box.  
 4643  
 4644 Authenticator extension output  
 4645 The hash value of the content which was displayed. The  
 4646 authenticator **MUST** use the same hash algorithm as it uses for  
 4647 the signature itself.  
 4648  
 4649 10.4. Authenticator Selection Extension (authnSel)  
 4650  
 4651 This registration extension allows a Relying Party to guide the  
 4652 selection of the authenticator that will be leveraged when creating the  
 4653 credential. It is intended primarily for Relying Parties that wish to  
 4654 tightly control the experience around credential creation.  
 4655  
 4656 Extension identifier  
 4657 authnSel  
 4658  
 4659 Client extension input  
 4660 A sequence of AAGUIDs:  
 4661  
 4662 typedef sequence<AAGUID> AuthenticatorSelectionList;  
 4663  
 4664  
 4665 partial dictionary AuthenticationExtensionsClientInputs {  
 4666 **AuthenticatorSelectionList** authnSel;  
 4667 }  
 4668  
 4669 Each AAGUID corresponds to an authenticator model that is  
 4670 acceptable to the Relying Party for this credential creation.  
 4671 The list is ordered by decreasing preference.  
 4672  
 4673 An AAGUID is defined as an array containing the globally unique  
 4674 identifier of the authenticator model being sought.  
 4675  
 4676 typedef BufferSource AAGUID;  
 4677  
 4678 Client extension processing

4281 This extension can only be used during create(). If the client  
 4282 supports the Authenticator Selection Extension, it MUST use the  
 4283 first available authenticator whose AAGUID is present in the  
 4284 AuthenticatorSelectionList. If none of the available  
 4285 authenticators match a provided AAGUID, the client MUST select  
 4286 an authenticator from among the available authenticators to  
 4287 generate the credential.  
 4288

4288 Client extension output

4289 Returns the **JSON** value true to indicate to the RP that the  
 4291 extension was acted upon

4292 Authenticator extension input  
 4293 None.

4294 Authenticator extension processing  
 4295 None.

4296 Authenticator extension output  
 4297 None.

4300 10.5. Supported Extensions Extension (exts)

4301 This registration extension enables the Relying Party to determine  
 4302 which extensions the authenticator supports.  
 4303

4304 Extension identifier  
 4305 exts

4306 Client extension input

4307 The Boolean value true to indicate that this extension is  
 4308 requested by the Relying Party.  
 4309

4310 Client extension processing

4311 None, except creating the authenticator extension input from the  
 4312 client extension input.  
 4313

4314 Client extension output

4315 Returns the list of supported extensions as a **JSON** array of  
 4316 extension identifier strings  
 4317

4321 Authenticator extension input

4322 The Boolean value true, encoded in CBOR (major type 7, value  
 4323 21).  
 4324

4325 Authenticator extension processing

4326 The authenticator sets the authenticator extension output to be  
 4327 a list of extensions that the authenticator supports, as defined  
 4328 below. This extension can be added to attestation objects.  
 4329

4330 Authenticator extension output

4331 The SupportedExtensions extension is a list (CBOR array) of  
 4332 extension identifier (UTF-8 encoded strings).  
 4333

4334 10.6. User Verification Index Extension (uvi)

4335

4336

4678 This extension can only be used during create(). If the client  
 4679 supports the Authenticator Selection Extension, it MUST use the  
 4680 first available authenticator whose AAGUID is present in the  
 4681 AuthenticatorSelectionList. If none of the available  
 4682 authenticators match a provided AAGUID, the client MUST select  
 4683 an authenticator from among the available authenticators to  
 4684 generate the credential.  
 4685

4685 Client extension output

4686 Returns the value true to indicate to the RP that the **extension**  
 4687 was acted upon.  
 4688

4689 partial dictionary AuthenticationExtensionsClientOutputs {  
 4690 boolean authnSel;  
 4691 };  
 4692

4693 Authenticator extension input  
 4694 None.

4695 Authenticator extension processing  
 4696 None.

4697 Authenticator extension output  
 4698 None.

4700 10.5. Supported Extensions Extension (exts)

4701 This registration extension enables the Relying Party to determine  
 4702 which extensions the authenticator supports.  
 4703

4704 Extension identifier  
 4705 exts

4706 Client extension input

4707 The Boolean value true to indicate that this extension is  
 4708 requested by the Relying Party.  
 4709

4710 partial dictionary AuthenticationExtensionsClientInputs {  
 4711 boolean exts;  
 4712 };  
 4713

4714 Client extension processing

4715 None, except creating the authenticator extension input from the  
 4716 client extension input.  
 4717

4718 Client extension output

4719 Returns the list of supported extensions as an array of  
 4720 extension identifier strings.  
 4721

4722 typedef sequence<USVString> AuthenticationExtensionsSupported;

4723 partial dictionary AuthenticationExtensionsClientOutputs {  
 4724 AuthenticationExtensionsSupported exts;  
 4725 };  
 4726

4727 Authenticator extension input

4728 The Boolean value true, encoded in CBOR (major type 7, value  
 4729 21).  
 4730

4731 Authenticator extension processing

4732 The authenticator sets the authenticator extension output to be  
 4733 a list of extensions that the authenticator supports, as defined  
 4734 below. This extension can be added to attestation objects.  
 4735

4736 Authenticator extension output

4737 The SupportedExtensions extension is a list (CBOR array) of  
 4738 extension identifier (UTF-8 encoded) strings.  
 4739

4740 10.6. User Verification Index Extension (uvi)

4741

4742

4337 This registration extension and authentication extension enables use of  
 4338 a user verification index.  
 4339  
 4340 Extension identifier  
 4341 uvi  
 4342  
 4343 Client extension input  
 4344 The Boolean value true to indicate that this extension is  
 4345 requested by the Relying Party.  
 4346

4347 Client extension processing  
 4348 None, except creating the authenticator extension input from the  
 4349 client extension input.  
 4350  
 4351 Client extension output  
 4352 Returns a JSON string containing the base64url encoding of the  
 4353 authenticator extension output

4354 Authenticator extension input  
 4355 The Boolean value true, encoded in CBOR (major type 7, value  
 4356 21).  
 4357

4358 Authenticator extension processing  
 4359 The authenticator sets the authenticator extension output to be  
 4360 a user verification index indicating the method used by the user  
 4361 to authorize the operation, as defined below. This extension can  
 4362 be added to attestation objects and assertions.  
 4363  
 4364 Authenticator extension output  
 4365 The user verification index (UVI) is a value uniquely  
 4366 identifying a user verification data record. The UVI is encoded  
 4367 as CBOR byte string (type 0x58). Each UVI value MUST be specific  
 4368 to the related key (in order to provide unlinkability). It also  
 4369 **must** contain sufficient entropy that makes guessing impractical.  
 4370 UVI values MUST NOT be reused by the Authenticator (for other  
 4371 biometric data or users).  
 4372  
 4373 The UVI data can be used by servers to understand whether an  
 4374 authentication was authorized by the exact same biometric data  
 4375 as the initial key generation. This allows the detection and  
 4376 prevention of "friendly fraud".  
 4377  
 4378 As an example, the UVI could be computed as SHA256(KeyID ||  
 4379 SHA256(rawUVI)), where || represents concatenation, and the  
 4380 rawUVI reflects (a) the biometric reference data, (b) the  
 4381 related OS level user ID and (c) an identifier which changes  
 4382 whenever a factory reset is performed for the device, e.g.  
 4383 rawUVI = biometricReferenceData || OSLevelUserID ||  
 4384 FactoryResetCounter.  
 4385  
 4386 Servers supporting UVI extensions MUST support a length of up to  
 4387 32 bytes for the UVI value.  
 4388  
 4389 Example for authenticator data containing one UVI extension  
 4390  
 4391  
 4392 ... -- [=RP ID=] hash (32 bytes)  
 4393 81 -- UP and ED set  
 4394 00 00 00 01 -- (initial) signature counter  
 4395 ... -- all public key alg etc.  
 4396 A1 -- extension: CBOR map of one elemen  
 4397 t  
 4398 63 -- Key 1: CBOR text string of 3 byte  
 4399 s

4748 This registration extension and authentication extension enables use of  
 4749 a user verification index.  
 4750  
 4751 Extension identifier  
 4752 uvi  
 4753  
 4754 Client extension input  
 4755 The Boolean value true to indicate that this extension is  
 4756 requested by the Relying Party.  
 4757

```
4758 partial dictionary AuthenticationExtensionsClientInputs {
4759   boolean uvi;
4760 };
```

4761 Client extension processing  
 4762 None, except creating the authenticator extension input from the  
 4763 client extension input.  
 4764  
 4765 Client extension output  
 4766 Returns the authenticator extension output as an ArrayBuffer.  
 4767

```
4768 partial dictionary AuthenticationExtensionsClientOutputs {
4769   ArrayBuffer uvi;
4770 };
```

```
4771 partial dictionary AuthenticationExtensionsClientOutputs {
4772   ArrayBuffer uvi;
4773 };
```

4774 Authenticator extension input  
 4775 The Boolean value true, encoded in CBOR (major type 7, value  
 4776 21).  
 4777

4778 Authenticator extension processing  
 4779 The authenticator sets the authenticator extension output to be  
 4780 a user verification index indicating the method used by the user  
 4781 to authorize the operation, as defined below. This extension can  
 4782 be added to attestation objects and assertions.  
 4783  
 4784 Authenticator extension output  
 4785 The user verification index (UVI) is a value uniquely  
 4786 identifying a user verification data record. The UVI is encoded  
 4787 as CBOR byte string (type 0x58). Each UVI value MUST be specific  
 4788 to the related key (in order to provide unlinkability). It also  
 4789 **MUST** contain sufficient entropy that makes guessing impractical.  
 4790 UVI values MUST NOT be reused by the Authenticator (for other  
 4791 biometric data or users).  
 4792  
 4793 The UVI data can be used by servers to understand whether an  
 4794 authentication was authorized by the exact same biometric data  
 4795 as the initial key generation. This allows the detection and  
 4796 prevention of "friendly fraud".  
 4797  
 4798 As an example, the UVI could be computed as SHA256(KeyID ||  
 4799 SHA256(rawUVI)), where || represents concatenation, and the  
 4800 rawUVI reflects (a) the biometric reference data, (b) the  
 4801 related OS level user ID and (c) an identifier which changes  
 4802 whenever a factory reset is performed for the device, e.g.  
 4803 rawUVI = biometricReferenceData || OSLevelUserID ||  
 4804 FactoryResetCounter.  
 4805  
 4806 Servers supporting UVI extensions MUST support a length of up to  
 4807 32 bytes for the UVI value.  
 4808  
 4809 Example for authenticator data containing one UVI extension  
 4810  
 4811  
 4812 ... -- [=RP ID=] hash (32 bytes)  
 4813 81 -- UP and ED set  
 4814 00 00 00 01 -- (initial) signature counter  
 4815 ... -- all public key alg etc.  
 4816 A1 -- extension: CBOR map of one elemen  
 4817 t  
 4818 63 -- Key 1: CBOR text string of 3 byte  
 4819 s

```

4400 75 76 69 -- "uvi" [=UTF-8 encoded=] string
4401 58 20 -- Value 1: CBOR byte string with 0x
4402 20 bytes
4403 00 43 B8 E3 BE 27 95 8C -- the UVI value itself
4404 28 D5 74 BF 46 8A 85 CF
4405 46 9A 14 F0 E5 16 69 31
4406 DA 4B CF FF C1 BB 11 32
4407 82
4408
4409 10.7. Location Extension (loc)
4410
4411 The location registration extension and authentication extension
4412 provides the client device's current location to the WebAuthn Relying
4413 Party.
4414
4415 Extension identifier
4416 loc
4417
4418 Client extension input
4419 The Boolean value true to indicate that this extension is
4420 requested by the Relying Party.
4421
4422
4423 Client extension processing
4424 None, except creating the authenticator extension input from the
4425 client extension input.
4426
4427 Client extension output
4428 Returns a JSON object that encodes the location information in
4429 the authenticator extension output as a Coordinates value, as
4430 defined by The W3C Geolocation API Specification.
4431
4432 Authenticator extension input
4433 The Boolean value true, encoded in CBOR (major type 7, value
4434 21).
4435
4436 Authenticator extension processing
4437 If the authenticator does not support the extension, then the
4438 authenticator MUST ignore the extension request. If the
4439 authenticator accepts the extension, then the authenticator
4440 SHOULD only add this extension data to a packed attestation or
4441 assertion.
4442
4443 Authenticator extension output
4444 If the authenticator accepts the extension request, then
4445 authenticator extension output SHOULD provide location data in
4446 the form of a CBOR-encoded map, with the first value being the
4447 extension identifier and the second being an array of returned
4448 values. The array elements SHOULD be derived from (key,value)
4449 pairings for each location attribute that the authenticator
4450 supports. The following is an example of authenticator data
4451 where the returned array is comprised of a {longitude, latitude,
4452 altitude} triplet, following the coordinate representation
4453 defined in The W3C Geolocation API Specification.
4454
4455 ... -- [=RP ID=] hash (32 bytes)
4456 81 -- UP and ED set
4457 00 00 00 01 -- (initial) signature counter
4458 ... -- all public key alg etc.
4459 A1 -- extension: CBOR map of one elemen
4460 t
4461 63 -- Value 1: CBOR text string of 3 by
tes

```

```

4818 75 76 69 -- "uvi" [=UTF-8 encoded=] string
4819 58 20 -- Value 1: CBOR byte string with 0x
4820 20 bytes
4821 43 B8 E3 BE 27 95 8C 28 -- the UVI value itself
4822 D5 74 BF 46 8A 85 CF 46
4823 9A 14 F0 E5 16 69 31 DA
4824 4B CF FF C1 BB 11 32 82
4825
4826 10.7. Location Extension (loc)
4827
4828 The location registration extension and authentication extension
4829 provides the client device's current location to the WebAuthn Relying
4830 Party.
4831
4832 Extension identifier
4833 loc
4834
4835 Client extension input
4836 The Boolean value true to indicate that this extension is
4837 requested by the Relying Party.
4838
4839 partial dictionary AuthenticationExtensionsClientInputs {
4840 boolean loc;
4841 };
4842
4843 Client extension processing
4844 None, except creating the authenticator extension input from the
4845 client extension input.
4846
4847 Client extension output
4848 Returns a JavaScript object that encodes the location
4849 information in the authenticator extension output as a
4850 Coordinates value, as defined by [Geolocation-API].
4851
4852 partial dictionary AuthenticationExtensionsClientOutputs {
4853 Coordinates loc;
4854 };
4855
4856 Authenticator extension input
4857 The Boolean value true, encoded in CBOR (major type 7, value
4858 21).
4859
4860 Authenticator extension processing
4861 Determine the Geolocation value.
4862
4863 Authenticator extension output
4864 A [Geolocation-API] Coordinates record encoded as a CBOR map.
4865 Values represented by the "double" type in JavaScript are
4866 represented as 64-bit CBOR floating point numbers. Per the
4867 Geolocation specification, the "latitude", "longitude", and
4868 "accuracy" values are required and other values such as
4869 "altitude" are optional.

```

```

4462 6C 6F 63 -- "loc" [=UTF-8 encoded=] string
4463 86 -- Value 2: array of 6 elements
4464 68 -- Element 1: CBOR text string of 8 bytes
4465 6C 61 74 69 74 75 64 65 -- "latitude" [=UTF-8 encoded=] stri
4466 ng
4467 FB ... -- Element 2: Latitude as CBOR encoded double-p
4468 recision float
4469 69 -- Element 3: CBOR text string of 9 bytes
4470 6C 6F 6E 67 69 74 75 64 65 -- "longitude" [=UTF-8 encoded=] str
4471 ing
4472 FB ... -- Element 4: Longitude as CBOR encoded double-
4473 precision float
4474 68 -- Element 5: CBOR text string of 8 bytes
4475 61 6C 74 69 74 75 64 65 -- "altitude" [=UTF-8 encoded=] stri
4476 ng
4477 FB ... -- Element 6: Altitude as CBOR encoded double-p
4478 recision float

```

### 10.8. User Verification Method Extension (uvm)

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

Extension identifier  
uvm

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

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

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

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

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

Authenticator extension output  
Authenticators can report up to 3 different user verification methods (factors) used in a single authentication instance, using the CBOR syntax defined below:

```

uvmFormat = [ 1*3 uvmEntry ]
uvmEntry = [
    userVerificationMethod: uint .size 4,
    keyProtectionType: uint .size 2,
    matcherProtectionType: uint .size 2

```

```

4870
4871 10.8. User Verification Method Extension (uvm)
4872
4873 This registration extension and authentication extension enables use of
4874 a user verification method.
4875
4876 Extension identifier
4877 uvm
4878
4879 Client extension input
4880 The Boolean value true to indicate that this extension is
4881 requested by the Relying Party.
4882

```

```

4883 partial dictionary AuthenticationExtensionsClientInputs {
4884     boolean uvm;
4885 };

```

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

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

```

4895 typedef sequence<unsigned long> UvmEntry;
4896 typedef sequence<UvmEntry> UvmEntries;
4897

```

```

4898 partial dictionary AuthenticationExtensionsClientOutputs {
4899     UvmEntries uvm;
4900 };

```

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

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

Authenticator extension output  
Authenticators can report up to 3 different user verification methods (factors) used in a single authentication instance, using the CBOR syntax defined below:

```

4917 uvmFormat = [ 1*3 uvmEntry ]
4918 uvmEntry = [
4919     userVerificationMethod: uint .size 4,
4920     keyProtectionType: uint .size 2,
4921     matcherProtectionType: uint .size 2
4922

```

4521 ]  
 4522  
 4523 The semantics of the fields in each uvmEntry are as follows:  
 4524  
 4525 userVerificationMethod  
 4526 The authentication method/factor used by the authenticator  
 4527 to verify the user. Available values are defined in  
 4528 [FIDOReg], "User Verification Methods" section.  
 4529  
 4530 keyProtectionType  
 4531 The method used by the authenticator to protect the FIDO  
 4532 registration private key material. Available values are  
 4533 defined in [FIDOReg], "Key Protection Types" section.  
 4534  
 4535 matcherProtectionType  
 4536 The method used by the authenticator to protect the  
 4537 matcher that performs user verification. Available values  
 4538 are defined in [FIDOReg], "Matcher Protection Types"  
 4539 section.  
 4540  
 4541 If >3 factors can be used in an authentication instance the  
 4542 authenticator vendor **must** select the 3 factors it believes will  
 4543 be most relevant to the Server to include in the UVM.  
 4544  
 4545 Example for authenticator data containing one UVM extension for  
 4546 a multi-factor authentication instance where 2 factors were  
 4547 used:  
 4548  
 4549 ... -- [=RP ID=] hash (32 bytes)  
 4550 81 -- UP and ED set  
 4551 00 00 00 01 -- (initial) signature counter  
 4552 ... -- all public key alg etc.  
 4553 A1 -- extension: CBOR map of one element  
 4554 63 -- Key 1: CBOR text string of 3 bytes  
 4555 75 76 6d -- "uvm" [=UTF-8 encoded=] string  
 4556 82 -- Value 1: CBOR array of length 2 indicating two factor  
 4557 usage  
 4558 83 -- Item 1: CBOR array of length 3  
 4559 02 -- Subitem 1: CBOR integer for User Verification Method  
 4560 Fingerprint  
 4561 04 -- Subitem 2: CBOR short for Key Protection Type TEE  
 4562 02 -- Subitem 3: CBOR short for Matcher Protection Type TE  
 4563 E  
 4564 83 -- Item 2: CBOR array of length 3  
 4565 04 -- Subitem 1: CBOR integer for User Verification Method  
 4566 Passcode  
 4567 01 -- Subitem 2: CBOR short for Key Protection Type Softwa  
 4568 re  
 4569 01 -- Subitem 3: CBOR short for Matcher Protection Type So  
 4570 ftware  
 4571

4923 ]  
 4924  
 4925 The semantics of the fields in each uvmEntry are as follows:  
 4926  
 4927 userVerificationMethod  
 4928 The authentication method/factor used by the authenticator  
 4929 to verify the user. Available values are defined in  
 4930 [FIDOReg], "User Verification Methods" section.  
 4931  
 4932 keyProtectionType  
 4933 The method used by the authenticator to protect the FIDO  
 4934 registration private key material. Available values are  
 4935 defined in [FIDOReg], "Key Protection Types" section.  
 4936  
 4937 matcherProtectionType  
 4938 The method used by the authenticator to protect the  
 4939 matcher that performs user verification. Available values  
 4940 are defined in [FIDOReg], "Matcher Protection Types"  
 4941 section.  
 4942  
 4943 If >3 factors can be used in an authentication instance the  
 4944 authenticator vendor **MUST** select the 3 factors it believes will  
 4945 be most relevant to the Server to include in the UVM.  
 4946  
 4947 Example for authenticator data containing one UVM extension for  
 4948 a multi-factor authentication instance where 2 factors were  
 4949 used:  
 4950  
 4951 ... -- [=RP ID=] hash (32 bytes)  
 4952 81 -- UP and ED set  
 4953 00 00 00 01 -- (initial) signature counter  
 4954 ... -- all public key alg etc.  
 4955 A1 -- extension: CBOR map of one element  
 4956 63 -- Key 1: CBOR text string of 3 bytes  
 4957 75 76 6d -- "uvm" [=UTF-8 encoded=] string  
 4958 82 -- Value 1: CBOR array of length 2 indicating two factor  
 4959 usage  
 4960 83 -- Item 1: CBOR array of length 3  
 4961 02 -- Subitem 1: CBOR integer for User Verification Method  
 4962 Fingerprint  
 4963 04 -- Subitem 2: CBOR short for Key Protection Type TEE  
 4964 02 -- Subitem 3: CBOR short for Matcher Protection Type TE  
 4965 E  
 4966 83 -- Item 2: CBOR array of length 3  
 4967 04 -- Subitem 1: CBOR integer for User Verification Method  
 4968 Passcode  
 4969 01 -- Subitem 2: CBOR short for Key Protection Type Softwa  
 4970 re  
 4971 01 -- Subitem 3: CBOR short for Matcher Protection Type So  
 4972 ftware

### 10.9. Biometric Authenticator Performance Bounds Extension (biometricPerfBounds)

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

Extension identifier  
biometricPerfBounds

Client extension input  
Biometric performance bounds:

```
dictionary authenticatorBiometricPerfBounds{
  float FAR;
  float FRR;
};
```

The FAR is the maximum false acceptance rate for a biometric

4572 11. IANA Considerations  
4573  
4574 11.1. WebAuthn Attestation Statement Format Identifier Registrations  
4575  
4576 This section registers the attestation statement formats defined in  
4577 Section 8 Defined Attestation Statement Formats in the IANA "WebAuthn  
4578 Attestation Statement Format Identifier" registry established by  
4579 [WebAuthn-Registries].  
4580 \* WebAuthn Attestation Statement Format Identifier: packed  
4581 \* Description: The "packed" attestation statement format is a  
4582 WebAuthn-optimized format for attestation. It uses a very compact  
4583 but still extensible encoding method. This format is implementable  
4584 by authenticators with limited resources (e.g., secure elements).  
4585 \* Specification Document: Section 8.2 Packed Attestation Statement  
4586 Format of this specification  
4587 \* WebAuthn Attestation Statement Format Identifier: tpm  
4588 \* Description: The TPM attestation statement format returns an  
4589 attestation statement in the same format as the packed attestation  
4590 statement format, although the rawData and signature fields are  
4591 computed differently.  
4592 \* Specification Document: Section 8.3 TPM Attestation Statement  
4593 Format of this specification  
4594 \* WebAuthn Attestation Statement Format Identifier: android-key  
4595 \* Description: Platform-provided authenticators based on versions  
4596 "N", and later, may provide this proprietary "hardware attestation"  
4597 statement.  
4598 \* Specification Document: Section 8.4 Android Key Attestation  
4599 Statement Format of this specification  
4600 \* WebAuthn Attestation Statement Format Identifier: android-safetynet  
4601 \* Description: Android-based, platform-provided authenticators may  
4602 produce an attestation statement based on the Android SafetyNet  
4603 API.  
4604 \* Specification Document: Section 8.5 Android SafetyNet Attestation  
4605 Statement Format of this specification  
4606 \* WebAuthn Attestation Statement Format Identifier: fido-u2f  
4607 \* Description: Used with FIDO U2F authenticators  
4608 \* Specification Document: Section 8.6 FIDO U2F Attestation Statement  
4609 Format of this specification  
4610  
4611 11.2. WebAuthn Extension Identifier Registrations  
4612  
4613 This section registers the extension identifier values defined in  
4614 Section 9 WebAuthn Extensions in the IANA "WebAuthn Extension

4993 authenticator allowed by the Relying Party.  
4994  
4995 The FAR is the maximum false rejection rate for a biometric  
4996 authenticator allowed by the Relying Party.  
4997  
4998 Client extension processing  
4999 This extension can only be used during create(). If the client  
5000 supports this extension, it MUST NOT use a biometric  
5001 authenticator whose FAR or FRR does not match the bounds as  
5002 provided. The client can obtain information about the biometric  
5003 authenticator's performance from authoritative sources such as  
5004 the FIDO Metadata Service [FIDOMetadataService] (see Sec. 3.2 of  
5005 [FIDOUAFAuthenticatorMetadataStatements]).  
5006  
5007 Client extension output  
5008 Returns the JSON value true to indicate to the RP that the  
5009 extension was acted upon  
5010  
5011 Authenticator extension input  
5012 None.  
5013  
5014 Authenticator extension processing  
5015 None.  
5016  
5017 Authenticator extension output  
5018 None.  
5019  
5020 11. IANA Considerations  
5021  
5022 11.1. WebAuthn Attestation Statement Format Identifier Registrations  
5023  
5024 This section registers the attestation statement formats defined in  
5025 Section 8 Defined Attestation Statement Formats in the IANA "WebAuthn  
5026 Attestation Statement Format Identifier" registry established by  
5027 [WebAuthn-Registries].  
5028 \* WebAuthn Attestation Statement Format Identifier: packed  
5029 \* Description: The "packed" attestation statement format is a  
5030 WebAuthn-optimized format for attestation. It uses a very compact  
5031 but still extensible encoding method. This format is implementable  
5032 by authenticators with limited resources (e.g., secure elements).  
5033 \* Specification Document: Section 8.2 Packed Attestation Statement  
5034 Format of this specification  
5035 \* WebAuthn Attestation Statement Format Identifier: tpm  
5036 \* Description: The TPM attestation statement format returns an  
5037 attestation statement in the same format as the packed attestation  
5038 statement format, although the rawData and signature fields are  
5039 computed differently.  
5040 \* Specification Document: Section 8.3 TPM Attestation Statement  
5041 Format of this specification  
5042 \* WebAuthn Attestation Statement Format Identifier: android-key  
5043 \* Description: Platform-provided authenticators based on versions  
5044 "N", and later, may provide this proprietary "hardware attestation"  
5045 statement.  
5046 \* Specification Document: Section 8.4 Android Key Attestation  
5047 Statement Format of this specification  
5048 \* WebAuthn Attestation Statement Format Identifier: android-safetynet  
5049 \* Description: Android-based, platform-provided authenticators MAY  
5050 produce an attestation statement based on the Android SafetyNet  
5051 API.  
5052 \* Specification Document: Section 8.5 Android SafetyNet Attestation  
5053 Statement Format of this specification  
5054 \* WebAuthn Attestation Statement Format Identifier: fido-u2f  
5055 \* Description: Used with FIDO U2F authenticators  
5056 \* Specification Document: Section 8.6 FIDO U2F Attestation Statement  
5057 Format of this specification  
5058  
5059 11.2. WebAuthn Extension Identifier Registrations  
5060  
5061 This section registers the extension identifier values defined in  
5062 Section 9 WebAuthn Extensions in the IANA "WebAuthn Extension

4615 Identifier" registry established by [WebAuthn-Registries].  
 4616 \* WebAuthn Extension Identifier: appid  
 4617 \* Description: This authentication extension allows Relying Parties  
 4618 that have previously registered a credential using the legacy FIDO  
 4619 JavaScript APIs to request an assertion.  
 4620 \* Specification Document: Section 10.1 FIDO AppID Extension (appid)  
 4621 of this specification  
 4622 \* WebAuthn Extension Identifier: txAuthSimple  
 4623 \* Description: This registration extension and authentication  
 4624 extension allows for a simple form of transaction authorization. A  
 4625 WebAuthn Relying Party can specify a prompt string, intended for  
 4626 display on a trusted device on the authenticator  
 4627 \* Specification Document: Section 10.2 Simple Transaction  
 4628 Authorization Extension (txAuthSimple) of this specification  
 4629 \* WebAuthn Extension Identifier: txAuthGeneric  
 4630 \* Description: This registration extension and authentication  
 4631 extension allows images to be used as transaction authorization  
 4632 prompts as well. This allows authenticators without a font  
 4633 rendering engine to be used and also supports a richer visual  
 4634 appearance than accomplished with the webauthn.txauth.simple  
 4635 extension.  
 4636 \* Specification Document: Section 10.3 Generic Transaction  
 4637 Authorization Extension (txAuthGeneric) of this specification  
 4638 \* WebAuthn Extension Identifier: authnSel  
 4639 \* Description: This registration extension allows a WebAuthn Relying  
 4640 Party to guide the selection of the authenticator that will be  
 4641 leveraged when creating the credential. It is intended primarily  
 4642 for WebAuthn Relying Parties that wish to tightly control the  
 4643 experience around credential creation.  
 4644 \* Specification Document: Section 10.4 Authenticator Selection  
 4645 Extension (authnSel) of this specification  
 4646 \* WebAuthn Extension Identifier: exts  
 4647 \* Description: This registration extension enables the Relying Party  
 4648 to determine which extensions the authenticator supports. The  
 4649 extension data is a list (CBOR array) of extension identifiers  
 4650 encoded as UTF-8 Strings. This extension is added automatically by  
 4651 the authenticator. This extension can be added to attestation  
 4652 statements.  
 4653 \* Specification Document: Section 10.5 Supported Extensions  
 4654 Extension (exts) of this specification  
 4655 \* WebAuthn Extension Identifier: uvi  
 4656 \* Description: This registration extension and authentication  
 4657 extension enables use of a user verification index. The user  
 4658 verification index is a value uniquely identifying a user  
 4659 verification data record. The UVI data can be used by servers to  
 4660 understand whether an authentication was authorized by the exact  
 4661 same biometric data as the initial key generation. This allows the  
 4662 detection and prevention of "friendly fraud".  
 4663 \* Specification Document: Section 10.6 User Verification Index  
 4664 Extension (uvi) of this specification  
 4665 \* WebAuthn Extension Identifier: loc  
 4666 \* Description: The location registration extension and authentication  
 4667 extension provides the client device's current location to the  
 4668 WebAuthn relying party, if supported by the client device and  
 4669 subject to user consent.  
 4670 \* Specification Document: Section 10.7 Location Extension (loc) of  
 4671 this specification  
 4672 \* WebAuthn Extension Identifier: uvm  
 4673 \* Description: This registration extension and authentication  
 4674 extension enables use of a user verification method. The user  
 4675 verification method extension returns to the Webauthn relying party  
 4676 which user verification methods (factors) were used for the  
 4677 WebAuthn operation.  
 4678 \* Specification Document: Section 10.8 User Verification Method  
 4679 Extension (uvm) of this specification

### 11.3. COSE Algorithm Registrations

This section registers identifiers for RSASSA-PKCS1-v1\_5 [RFC8017] algorithms using SHA-2 and SHA-1 hash functions in the IANA COSE

5063 Identifier" registry established by [WebAuthn-Registries].  
 5064 \* WebAuthn Extension Identifier: appid  
 5065 \* Description: This authentication extension allows Relying Parties  
 5066 that have previously registered a credential using the legacy FIDO  
 5067 JavaScript APIs to request an assertion.  
 5068 \* Specification Document: Section 10.1 FIDO AppID Extension (appid)  
 5069 of this specification  
 5070 \* WebAuthn Extension Identifier: txAuthSimple  
 5071 \* Description: This registration extension and authentication  
 5072 extension allows for a simple form of transaction authorization. A  
 5073 WebAuthn Relying Party can specify a prompt string, intended for  
 5074 display on a trusted device on the authenticator  
 5075 \* Specification Document: Section 10.2 Simple Transaction  
 5076 Authorization Extension (txAuthSimple) of this specification  
 5077 \* WebAuthn Extension Identifier: txAuthGeneric  
 5078 \* Description: This registration extension and authentication  
 5079 extension allows images to be used as transaction authorization  
 5080 prompts as well. This allows authenticators without a font  
 5081 rendering engine to be used and also supports a richer visual  
 5082 appearance than accomplished with the webauthn.txauth.simple  
 5083 extension.  
 5084 \* Specification Document: Section 10.3 Generic Transaction  
 5085 Authorization Extension (txAuthGeneric) of this specification  
 5086 \* WebAuthn Extension Identifier: authnSel  
 5087 \* Description: This registration extension allows a WebAuthn Relying  
 5088 Party to guide the selection of the authenticator that will be  
 5089 leveraged when creating the credential. It is intended primarily  
 5090 for WebAuthn Relying Parties that wish to tightly control the  
 5091 experience around credential creation.  
 5092 \* Specification Document: Section 10.4 Authenticator Selection  
 5093 Extension (authnSel) of this specification  
 5094 \* WebAuthn Extension Identifier: exts  
 5095 \* Description: This registration extension enables the Relying Party  
 5096 to determine which extensions the authenticator supports. The  
 5097 extension data is a list (CBOR array) of extension identifiers  
 5098 encoded as UTF-8 Strings. This extension is added automatically by  
 5099 the authenticator. This extension can be added to attestation  
 5100 statements.  
 5101 \* Specification Document: Section 10.5 Supported Extensions  
 5102 Extension (exts) of this specification  
 5103 \* WebAuthn Extension Identifier: uvi  
 5104 \* Description: This registration extension and authentication  
 5105 extension enables use of a user verification index. The user  
 5106 verification index is a value uniquely identifying a user  
 5107 verification data record. The UVI data can be used by servers to  
 5108 understand whether an authentication was authorized by the exact  
 5109 same biometric data as the initial key generation. This allows the  
 5110 detection and prevention of "friendly fraud".  
 5111 \* Specification Document: Section 10.6 User Verification Index  
 5112 Extension (uvi) of this specification  
 5113 \* WebAuthn Extension Identifier: loc  
 5114 \* Description: The location registration extension and authentication  
 5115 extension provides the client device's current location to the  
 5116 WebAuthn relying party, if supported by the client device and  
 5117 subject to user consent.  
 5118 \* Specification Document: Section 10.7 Location Extension (loc) of  
 5119 this specification  
 5120 \* WebAuthn Extension Identifier: uvm  
 5121 \* Description: This registration extension and authentication  
 5122 extension enables use of a user verification method. The user  
 5123 verification method extension returns to the Webauthn relying party  
 5124 which user verification methods (factors) were used for the  
 5125 WebAuthn operation.  
 5126 \* Specification Document: Section 10.8 User Verification Method  
 5127 Extension (uvm) of this specification

### 11.3. COSE Algorithm Registrations

This section registers identifiers for RSASSA-PKCS1-v1\_5 [RFC8017] algorithms using SHA-2 and SHA-1 hash functions in the IANA COSE

4685 Algorithms registry [IANA-COSE-ALGS-REG]. It also registers identifiers  
 4686 for ECDAAs algorithms.  
 4687 \* Name: RS256  
 4688 \* Value: -257  
 4689 \* Description: RSASSA-PKCS1-v1\_5 w/ SHA-256  
 4690 \* Reference: Section 8.2 of [RFC8017]  
 4691 \* Recommended: No  
 4692 \* Name: RS384  
 4693 \* Value: -258  
 4694 \* Description: RSASSA-PKCS1-v1\_5 w/ SHA-384  
 4695 \* Reference: Section 8.2 of [RFC8017]  
 4696 \* Recommended: No  
 4697 \* Name: RS512  
 4698 \* Value: -259  
 4699 \* Description: RSASSA-PKCS1-v1\_5 w/ SHA-512  
 4700 \* Reference: Section 8.2 of [RFC8017]  
 4701 \* Recommended: No  
 4702 \* Name: ED256  
 4703 \* Value: -260  
 4704 \* Description: TPM\_ECC\_BN\_P256 curve w/ SHA-256  
 4705 \* Reference: Section 4.2 of [FIDOEcdaaAlgorithm]  
 4706 \* Recommended: Yes  
 4707 \* Name: ED512  
 4708 \* Value: -261  
 4709 \* Description: ECC\_BN\_ISOP512 curve w/ SHA-512  
 4710 \* Reference: Section 4.2 of [FIDOEcdaaAlgorithm]  
 4711 \* Recommended: Yes  
 4712 \* Name: RS1  
 4713 \* Value: -262  
 4714 \* Description: RSASSA-PKCS1-v1\_5 w/ SHA-1  
 4715 \* Reference: Section 8.2 of [RFC8017]  
 4716 \* Recommended: No

## 12. Sample scenarios

This section is not normative.

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

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

### 12.1. Registration

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

1. The user visits example.com, which serves up a script. At this point, the user may already be logged in using a legacy username and password, or additional authenticator, or other means acceptable to the Relying Party. Or the user may be in the process of creating a new account.
2. The Relying Party script runs the code snippet below.
3. The client platform searches for and locates the authenticator.
4. The client platform connects to the authenticator, performing any pairing actions if necessary.

5133 Algorithms registry [IANA-COSE-ALGS-REG]. It also registers identifiers  
 5134 for ECDAAs algorithms.  
 5135 \* Name: RS256  
 5136 \* Value: TBD (requested assignment -257)  
 5137 \* Description: RSASSA-PKCS1-v1\_5 w/ SHA-256  
 5138 \* Reference: Section 8.2 of [RFC8017]  
 5139 \* Recommended: No  
 5140 \* Name: RS384  
 5141 \* Value: TBD (requested assignment -258)  
 5142 \* Description: RSASSA-PKCS1-v1\_5 w/ SHA-384  
 5143 \* Reference: Section 8.2 of [RFC8017]  
 5144 \* Recommended: No  
 5145 \* Name: RS512  
 5146 \* Value: TBD (requested assignment -259)  
 5147 \* Description: RSASSA-PKCS1-v1\_5 w/ SHA-512  
 5148 \* Reference: Section 8.2 of [RFC8017]  
 5149 \* Recommended: No  
 5150 \* Name: ED256  
 5151 \* Value: TBD (requested assignment -260)  
 5152 \* Description: TPM\_ECC\_BN\_P256 curve w/ SHA-256  
 5153 \* Reference: Section 4.2 of [FIDOEcdaaAlgorithm]  
 5154 \* Recommended: Yes  
 5155 \* Name: ED512  
 5156 \* Value: TBD (requested assignment -261)  
 5157 \* Description: ECC\_BN\_ISOP512 curve w/ SHA-512  
 5158 \* Reference: Section 4.2 of [FIDOEcdaaAlgorithm]  
 5159 \* Recommended: Yes  
 5160 \* Name: RS1  
 5161 \* Value: TBD (requested assignment -262)  
 5162 \* Description: RSASSA-PKCS1-v1\_5 w/ SHA-1  
 5163 \* Reference: Section 8.2 of [RFC8017]  
 5164 \* Recommended: No

## 12. Sample scenarios

This section is not normative.

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

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

### 12.1. Registration

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

1. The user visits example.com, which serves up a script. At this point, the user may already be logged in using a legacy username and password, or additional authenticator, or other means acceptable to the Relying Party. Or the user may be in the process of creating a new account.
2. The Relying Party script runs the code snippet below.
3. The client platform searches for and locates the authenticator.
4. The client platform connects to the authenticator, performing any pairing actions if necessary.

```

4755 5. The authenticator shows appropriate UI for the user to select the
4756 authenticator on which the new credential will be created, and
4757 obtains a biometric or other authorization gesture from the user.
4758 6. The authenticator returns a response to the client platform, which
4759 in turn returns a response to the Relying Party script. If the user
4760 declined to select an authenticator or provide authorization, an
4761 appropriate error is returned.
4762 7. If a new credential was created,
4763 + The Relying Party script sends the newly generated credential
4764 public key to the server, along with additional information
4765 such as attestation regarding the provenance and
4766 characteristics of the authenticator.
4767 + The server stores the credential public key in its database
4768 and associates it with the user as well as with the
4769 characteristics of authentication indicated by attestation,
4770 also storing a friendly name for later use.
4771 + The script may store data such as the credential ID in local
4772 storage, to improve future UX by narrowing the choice of
4773 credential for the user.
4774
4775 The sample code for generating and registering a new key follows:
4776 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */}
4777
4778 var publicKey = {
4779 // The challenge must be produced by the server, see the Security Consideratio
4780 ns
4781 challenge: new Uint8Array([21,31,105 /* 29 more random bytes generated by the
4782 server */]),
4783
4784 // Relying Party:
4785 rp: {
4786 name: "Acme"
4787 },
4788
4789 // User:
4790 user: {
4791 id: Uint8Array.from(window.atob("MIIBkzCCATigAwIBAJCCAAMwggE4oAMCAQIwggGTMII
4792 ="), c=>c.charCodeAt(0)),
4793 name: "john.p.smith@example.com",
4794 displayName: "John P. Smith",
4795 icon: "https://pics.acme.com/00/p/aBjjjpqPb.png"
4796 },
4797
4798 // This Relying Party will accept either an ES256 or RS256 credential, but
4799 // prefers an ES256 credential.
4800 pubKeyCredParams: [
4801 {
4802 type: "public-key",
4803 alg: -7 // "ES256" as registered in the IANA COSE Algorithms registry
4804 },
4805 {
4806 type: "public-key",
4807 alg: -257 // Value registered by this specification for "RS256"
4808 }
4809 ],
4810
4811 timeout: 60000, // 1 minute
4812 excludeCredentials: [], // No exclude list of PKCredDescriptors
4813 extensions: {"loc": true} // Include location information
4814 // in attestation
4815 };
4816
4817 // Note: The following call will cause the authenticator to display UI.
4818 navigator.credentials.create({ publicKey })
4819 .then(function (newCredentialInfo) {
4820 // Send new credential info to server for verification and registration.
4821 }).catch(function (err) {
4822 // No acceptable authenticator or user refused consent. Handle appropriately
4823
4824 });

```

```

5203 5. The authenticator shows appropriate UI for the user to select the
5204 authenticator on which the new credential will be created, and
5205 obtains a biometric or other authorization gesture from the user.
5206 6. The authenticator returns a response to the client platform, which
5207 in turn returns a response to the Relying Party script. If the user
5208 declined to select an authenticator or provide authorization, an
5209 appropriate error is returned.
5210 7. If a new credential was created,
5211 + The Relying Party script sends the newly generated credential
5212 public key to the server, along with additional information
5213 such as attestation regarding the provenance and
5214 characteristics of the authenticator.
5215 + The server stores the credential public key in its database
5216 and associates it with the user as well as with the
5217 characteristics of authentication indicated by attestation,
5218 also storing a friendly name for later use.
5219 + The script may store data such as the credential ID in local
5220 storage, to improve future UX by narrowing the choice of
5221 credential for the user.
5222
5223 The sample code for generating and registering a new key follows:
5224 if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */}
5225
5226 var publicKey = {
5227 // The challenge must be produced by the server, see the Security Consideratio
5228 ns
5229 challenge: new Uint8Array([21,31,105 /* 29 more random bytes generated by the
5230 server */]),
5231
5232 // Relying Party:
5233 rp: {
5234 name: "ACME Corporation"
5235 },
5236
5237 // User:
5238 user: {
5239 id: Uint8Array.from(window.atob("MIIBkzCCATigAwIBAJCCAAMwggE4oAMCAQIwggGTMII
5240 ="), c=>c.charCodeAtAt(0)),
5241 name: "alex.p.mueller@example.com",
5242 displayName: "Alex P. Miller",
5243 icon: "https://pics.example.com/00/p/aBjjjpqPb.png"
5244 },
5245
5246 // This Relying Party will accept either an ES256 or RS256 credential, but
5247 // prefers an ES256 credential.
5248 pubKeyCredParams: [
5249 {
5250 type: "public-key",
5251 alg: -7 // "ES256" as registered in the IANA COSE Algorithms registry
5252 },
5253 {
5254 type: "public-key",
5255 alg: -257 // Value registered by this specification for "RS256"
5256 }
5257 ],
5258
5259 timeout: 60000, // 1 minute
5260 excludeCredentials: [], // No exclude list of PKCredDescriptors
5261 extensions: {"loc": true} // Include location information
5262 // in attestation
5263 };
5264
5265 // Note: The following call will cause the authenticator to display UI.
5266 navigator.credentials.create({ publicKey })
5267 .then(function (newCredentialInfo) {
5268 // Send new credential info to server for verification and registration.
5269 }).catch(function (err) {
5270 // No acceptable authenticator or user refused consent. Handle appropriately
5271
5272 });

```

```

4825
4826 12.2. Registration Specifically with User Verifying Platform Authenticator
4827
4828 This is flow for when the Relying Party is specifically interested in
4829 creating a public key credential with a user-verifying platform
4830 authenticator.
4831 1. The user visits example.com and clicks on the login button, which
4832 redirects the user to login.example.com.
4833 2. The user enters a username and password to log in. After successful
4834 login, the user is redirected back to example.com.
4835 3. The Relying Party script runs the code snippet below.
4836 4. The user agent asks the user whether they are willing to register
4837 with the Relying Party using an available platform authenticator.
4838 5. If the user is not willing, terminate this flow.
4839 6. The user is shown appropriate UI and guided in creating a
4840 credential using one of the available platform authenticators. Upon
4841 successful credential creation, the RP script conveys the new
4842 credential to the server.
4843 if (!PublicKeyCredential) { /* Platform not capable of the API. Handle error. */
4844 }
4845
4846 PublicKeyCredential.isUserVerifyingPlatformAuthenticatorAvailable()
4847 .then(function (userIntent) {
4848
4849 // If the user has affirmed willingness to register with RP using an available platform authenticator
4850 if (userIntent) {
4851 var publicKeyOptions = { /* Public key credential creation options.
4852 */};
4853
4854 // Create and register credentials.
4855 return navigator.credentials.create({ "publicKey": publicKeyOptions
4856 });
4857 } else {
4858
4859 // Record that the user does not intend to use a platform authenticator
4860 // and default the user to a password-based flow in the future.
4861 }
4862
4863 }).then(function (newCredentialInfo) {
4864 // Send new credential info to server for verification and registration.
4865 }).catch( function(err) {
4866 // Something went wrong. Handle appropriately.
4867 });
4868
4869 12.3. Authentication
4870
4871 This is the flow when a user with an already registered credential
4872 visits a website and wants to authenticate using the credential.
4873 1. The user visits example.com, which serves up a script.
4874 2. The script asks the client platform for an Authentication
4875 Assertion, providing as much information as possible to narrow the
4876 choice of acceptable credentials for the user. This may be obtained
4877 from the data that was stored locally after registration, or by
4878 other means such as prompting the user for a username.
4879 3. The Relying Party script runs one of the code snippets below.
4880 4. The client platform searches for and locates the authenticator.
4881 5. The client platform connects to the authenticator, performing any
4882 pairing actions if necessary.
4883 6. The authenticator presents the user with a notification that their
4884 attention is required. On opening the notification, the user is
4885 shown a friendly selection menu of acceptable credentials using the
4886 account information provided when creating the credentials, along
4887 with some information on the origin that is requesting these keys.
4888 7. The authenticator obtains a biometric or other authorization
4889 gesture from the user.
4890 8. The authenticator returns a response to the client platform, which
4891 in turn returns a response to the Relying Party script. If the user
4892
4893

```

```

5273
5274 12.2. Registration Specifically with User Verifying Platform Authenticator
5275
5276 This is flow for when the Relying Party is specifically interested in
5277 creating a public key credential with a user-verifying platform
5278 authenticator.
5279 1. The user visits example.com and clicks on the login button, which
5280 redirects the user to login.example.com.
5281 2. The user enters a username and password to log in. After successful
5282 login, the user is redirected back to example.com.
5283 3. The Relying Party script runs the code snippet below.
5284 4. The user agent asks the user whether they are willing to register
5285 with the Relying Party using an available platform authenticator.
5286 5. If the user is not willing, terminate this flow.
5287 6. The user is shown appropriate UI and guided in creating a
5288 credential using one of the available platform authenticators. Upon
5289 successful credential creation, the RP script conveys the new
5290 credential to the server.
5291 if (!window.PublicKeyCredential) { /* Platform not capable of the API. Handle error. */
5292 }
5293
5294 PublicKeyCredential.isUserVerifyingPlatformAuthenticatorAvailable()
5295 .then(function (userIntent) {
5296
5297 // If the user has affirmed willingness to register with RP using an available platform authenticator
5298 if (userIntent) {
5299 var publicKeyOptions = { /* Public key credential creation options.
5300 */};
5301
5302 // Create and register credentials.
5303 return navigator.credentials.create({ "publicKey": publicKeyOptions
5304 });
5305 } else {
5306
5307 // Record that the user does not intend to use a platform authenticator
5308 // and default the user to a password-based flow in the future.
5309 }
5310
5311 }).then(function (newCredentialInfo) {
5312 // Send new credential info to server for verification and registration.
5313 }).catch( function(err) {
5314 // Something went wrong. Handle appropriately.
5315 });
5316
5317 12.3. Authentication
5318
5319 This is the flow when a user with an already registered credential
5320 visits a website and wants to authenticate using the credential.
5321 1. The user visits example.com, which serves up a script.
5322 2. The script asks the client platform for an Authentication
5323 Assertion, providing as much information as possible to narrow the
5324 choice of acceptable credentials for the user. This can be obtained
5325 from the data that was stored locally after registration, or by
5326 other means such as prompting the user for a username.
5327 3. The Relying Party script runs one of the code snippets below.
5328 4. The client platform searches for and locates the authenticator.
5329 5. The client platform connects to the authenticator, performing any
5330 pairing actions if necessary.
5331 6. The authenticator presents the user with a notification that their
5332 attention is needed. On opening the notification, the user is
5333 shown a friendly selection menu of acceptable credentials using the
5334 account information provided when creating the credentials, along
5335 with some information on the origin that is requesting these keys.
5336 7. The authenticator obtains a biometric or other authorization
5337 gesture from the user.
5338 8. The authenticator returns a response to the client platform, which
5339 in turn returns a response to the Relying Party script. If the user
5340
5341
5342

```

```

4894 declined to select a credential or provide an authorization, an
4895 appropriate error is returned.
4896 9. If an assertion was successfully generated and returned,
4897 + The script sends the assertion to the server.
4898 + The server examines the assertion, extracts the credential ID,
4899 looks up the registered credential public key it is database,
4900 and verifies the assertion's authentication signature. If
4901 valid, it looks up the identity associated with the
4902 assertion's credential ID; that identity is now authenticated.
4903 If the credential ID is not recognized by the server (e.g., it
4904 has been deregistered due to inactivity) then the
4905 authentication has failed; each Relying Party will handle this
4906 in its own way.
4907 + The server now does whatever it would otherwise do upon
4908 successful authentication -- return a success page, set
4909 authentication cookies, etc.
4910
4911 If the Relying Party script does not have any hints available (e.g.,
4912 from locally stored data) to help it narrow the list of credentials,
4913 then the sample code for performing such an authentication might look
4914 like this:
4915 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */}
4916
4917 var options = {
4918     // The challenge must be produced by the server, see the Security
4919     // Considerations
4920     challenge: new Uint8Array([4,101,15 /* 29 more random bytes generated
4921     by the server */]),
4922     timeout: 60000, // 1 minute
4923     allowCredentials: [{ type: "public-key" }]
4924 };
4925
4926 navigator.credentials.get({ "publicKey": options })
4927 .then(function (assertion) {
4928     // Send assertion to server for verification
4929 }).catch(function (err) {
4930     // No acceptable credential or user refused consent. Handle appropriately.
4931 });
4932
4933 On the other hand, if the Relying Party script has some hints to help
4934 it narrow the list of credentials, then the sample code for performing
4935 such an authentication might look like the following. Note that this
4936 sample also demonstrates how to use the extension for transaction
4937 authorization.
4938 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */}
4939
4940 var encoder = new TextEncoder();
4941 var acceptableCredential1 = {
4942     type: "public-key",
4943     id: encoder.encode("!!!!!!!hi there!!!!!!!\n")
4944 };
4945 var acceptableCredential2 = {
4946     type: "public-key",
4947     id: encoder.encode("roses are red, violets are blue\n")
4948 };
4949
4950 var options = {
4951     // The challenge must be produced by the server, see the Security
4952     // Considerations
4953     challenge: new Uint8Array([8,18,33 /* 29 more random bytes generated
4954     by the server */]),
4955     timeout: 60000, // 1 minute
4956     allowCredentials: [acceptableCredential1, acceptableCredential2]
4957 ,
4958     extensions: { 'txAuthSimple':
4959         "Wave your hands in the air like you just don't care" }
4960 };
4961
4962 navigator.credentials.get({ "publicKey": options })
4963 .then(function (assertion) {

```

```

5343 declined to select a credential or provide an authorization, an
5344 appropriate error is returned.
5345 9. If an assertion was successfully generated and returned,
5346 + The script sends the assertion to the server.
5347 + The server examines the assertion, extracts the credential ID,
5348 looks up the registered credential public key it is database,
5349 and verifies the assertion's authentication signature. If
5350 valid, it looks up the identity associated with the
5351 assertion's credential ID; that identity is now authenticated.
5352 If the credential ID is not recognized by the server (e.g., it
5353 has been deregistered due to inactivity) then the
5354 authentication has failed; each Relying Party will handle this
5355 in its own way.
5356 + The server now does whatever it would otherwise do upon
5357 successful authentication -- return a success page, set
5358 authentication cookies, etc.
5359
5360 If the Relying Party script does not have any hints available (e.g.,
5361 from locally stored data) to help it narrow the list of credentials,
5362 then the sample code for performing such an authentication might look
5363 like this:
5364 if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */}
5365
5366 var options = {
5367     // The challenge must be produced by the server, see the Security
5368     // Considerations
5369     challenge: new Uint8Array([4,101,15 /* 29 more random bytes generated
5370     by the server */]),
5371     timeout: 60000, // 1 minute
5372     allowCredentials: [{ type: "public-key" }]
5373 };
5374
5375 navigator.credentials.get({ "publicKey": options })
5376 .then(function (assertion) {
5377     // Send assertion to server for verification
5378 }).catch(function (err) {
5379     // No acceptable credential or user refused consent. Handle appropriately.
5380 });
5381
5382 On the other hand, if the Relying Party script has some hints to help
5383 it narrow the list of credentials, then the sample code for performing
5384 such an authentication might look like the following. Note that this
5385 sample also demonstrates how to use the extension for transaction
5386 authorization.
5387 if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */}
5388
5389 var encoder = new TextEncoder();
5390 var acceptableCredential1 = {
5391     type: "public-key",
5392     id: encoder.encode("!!!!!!!hi there!!!!!!!\n")
5393 };
5394 var acceptableCredential2 = {
5395     type: "public-key",
5396     id: encoder.encode("roses are red, violets are blue\n")
5397 };
5398
5399 var options = {
5400     // The challenge must be produced by the server, see the Security
5401     // Considerations
5402     challenge: new Uint8Array([8,18,33 /* 29 more random bytes generated
5403     by the server */]),
5404     timeout: 60000, // 1 minute
5405     allowCredentials: [acceptableCredential1, acceptableCredential2]
5406 ,
5407     extensions: { 'txAuthSimple':
5408         "Wave your hands in the air like you just don't care" }
5409 };
5410
5411 navigator.credentials.get({ "publicKey": options })
5412 .then(function (assertion) {

```

```

4964 // Send assertion to server for verification
4965 }).catch(function (err) {
4966 // No acceptable credential or user refused consent. Handle appropriately.
4967 });
4968
4969 12.4. Aborting Authentication Operations
4970
4971 The below example shows how a developer may use the AbortSignal
4972 parameter to abort a credential registration operation. A similar
4973 procedure applies to an authentication operation.
4974 const authAbortController = new AbortController();
4975 const authAbortSignal = authAbortController.signal;
4976
4977 authAbortSignal.onabort = function () {
4978 // Once the page knows the abort started, inform user it is attempting to ab
4979 ort.
4980 }
4981
4982 var options = {
4983 // A list of options.
4984 }
4985
4986 navigator.credentials.create({
4987 publicKey: options,
4988 signal: authAbortSignal})
4989 .then(function (attestation) {
4990 // Register the user.
4991 }).catch(function (error) {
4992 if (error == "AbortError") {
4993 // Inform user the credential hasn't been created.
4994 // Let the server know a key hasn't been created.
4995 }
4996 });
4997
4998 // Assume widget shows up whenever auth occurs.
4999 if (widget == "disappear") {
5000 authAbortSignal.abort();
5001 }
5002
5003
5004 12.5. Decommissioning
5005
5006 The following are possible situations in which decommissioning a
5007 credential might be desired. Note that all of these are handled on the
5008 server side and do not need support from the API specified here.
5009 * Possibility #1 -- user reports the credential as lost.
5010 + User goes to server.example.net, authenticates and follows a
5011 link to report a lost/stolen device.
5012 + Server returns a page showing the list of registered
5013 credentials with friendly names as configured during
5014 registration.
5015 + User selects a credential and the server deletes it from its
5016 database.
5017 + In future, the Relying Party script does not specify this
5018 credential in any list of acceptable credentials, and
5019 assertions signed by this credential are rejected.
5020 * Possibility #2 -- server deregisters the credential due to
5021 inactivity.
5022 + Server deletes credential from its database during maintenance
5023 activity.
5024 + In the future, the Relying Party script does not specify this
5025 credential in any list of acceptable credentials, and
5026 assertions signed by this credential are rejected.
5027 * Possibility #3 -- user deletes the credential from the device.
5028 + User employs a device-specific method (e.g., device settings
5029 UI) to delete a credential from their device.
5030 + From this point on, this credential will not appear in any
5031 selection prompts, and no assertions can be generated with it.
5032 + Sometime later, the server deregisters this credential due to
5033 inactivity.

```

```

5413 // Send assertion to server for verification
5414 }).catch(function (err) {
5415 // No acceptable credential or user refused consent. Handle appropriately.
5416 });
5417
5418 12.4. Aborting Authentication Operations
5419
5420 The below example shows how a developer may use the AbortSignal
5421 parameter to abort a credential registration operation. A similar
5422 procedure applies to an authentication operation.
5423 const authAbortController = new AbortController();
5424 const authAbortSignal = authAbortController.signal;
5425
5426 authAbortSignal.onabort = function () {
5427 // Once the page knows the abort started, inform user it is attempting to ab
5428 ort.
5429 }
5430
5431 var options = {
5432 // A list of options.
5433 }
5434
5435 navigator.credentials.create({
5436 publicKey: options,
5437 signal: authAbortSignal})
5438 .then(function (attestation) {
5439 // Register the user.
5440 }).catch(function (error) {
5441 if (error == "AbortError") {
5442 // Inform user the credential hasn't been created.
5443 // Let the server know a key hasn't been created.
5444 }
5445 });
5446
5447 // Assume widget shows up whenever authentication occurs.
5448 if (widget == "disappear") {
5449 authAbortSignal.abort();
5450 }
5451
5452
5453 12.5. Decommissioning
5454
5455 The following are possible situations in which decommissioning a
5456 credential might be desired. Note that all of these are handled on the
5457 server side and do not need support from the API specified here.
5458 * Possibility #1 -- user reports the credential as lost.
5459 + User goes to server.example.net, authenticates and follows a
5460 link to report a lost/stolen device.
5461 + Server returns a page showing the list of registered
5462 credentials with friendly names as configured during
5463 registration.
5464 + User selects a credential and the server deletes it from its
5465 database.
5466 + In future, the Relying Party script does not specify this
5467 credential in any list of acceptable credentials, and
5468 assertions signed by this credential are rejected.
5469 * Possibility #2 -- server deregisters the credential due to
5470 inactivity.
5471 + Server deletes credential from its database during maintenance
5472 activity.
5473 + In the future, the Relying Party script does not specify this
5474 credential in any list of acceptable credentials, and
5475 assertions signed by this credential are rejected.
5476 * Possibility #3 -- user deletes the credential from the device.
5477 + User employs a device-specific method (e.g., device settings
5478 UI) to delete a credential from their device.
5479 + From this point on, this credential will not appear in any
5480 selection prompts, and no assertions can be generated with it.
5481 + Sometime later, the server deregisters this credential due to
5482 inactivity.

```

5034  
5035  
5036

### 13. Security Considerations

5037

#### 13.1. Cryptographic Challenges

5038  
5039  
5040  
5041  
5042  
5043  
5044  
5045  
5046  
5047  
5048

As a cryptographic protocol, Web Authentication is dependent upon randomized challenges to avoid replay attacks. Therefore, both `{MakePublicKeyCredentialOptions/challenge}`'s and challenge's value, MUST be randomly generated by the Relying Party in an environment they trust (e.g., on the server-side), and the challenge 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.

5049  
5050  
5051

#### 14. Acknowledgements

5052  
5053  
5054  
5055  
5056  
5057

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

5482  
5483  
5484  
5485  
5486  
5487  
5488  
5489  
5490  
5491  
5492  
5493  
5494  
5495  
5496  
5497  
5498  
5499  
5500  
5501  
5502  
5503  
5504  
5505  
5506  
5507  
5508  
5509  
5510  
5511  
5512  
5513  
5514  
5515  
5516  
5517  
5518  
5519  
5520  
5521  
5522  
5523  
5524  
5525  
5526  
5527  
5528  
5529  
5530  
5531  
5532  
5533  
5534  
5535  
5536  
5537  
5538  
5539  
5540  
5541  
5542  
5543  
5544  
5545  
5546  
5547  
5548  
5549  
5550

### 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 [FIDOAuthnSecReqs] 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.

#### 13.2. Attestation Security Considerations

##### 13.2.1. Attestation Certificate Hierarchy

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

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

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

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

A WebAuthn Authenticator attestation certificate MUST be revoked by the issuing CA if its key has been compromised. A WebAuthn Authenticator manufacturer may need to ship a firmware update and inject new attestation keys and certificates into already manufactured WebAuthn Authenticators, if the exposure was due to a firmware flaw. (The process by which this happens is out of scope for this specification.) If the WebAuthn Authenticator manufacturer does not have this

5551  
5552  
5553  
5554  
5555  
5556  
5557  
5558  
5559  
5560  
5561  
5562  
5563  
5564  
5565  
5566  
5567  
5568  
5569  
5570  
5571  
5572  
5573  
5574  
5575  
5576  
5577  
5578  
5579  
5580  
5581  
5582  
5583  
5584  
5585  
5586  
5587  
5588  
5589  
5590  
5591  
5592  
5593  
5594  
5595  
5596  
5597  
5598  
5599  
5600  
5601  
5602  
5603  
5604  
5605  
5606  
5607  
5608  
5609  
5610  
5611  
5612  
5613  
5614  
5615  
5616  
5617  
5618  
5619  
5620

capability, then it may not be possible for Relying Parties to trust any further attestation statements from the affected WebAuthn Authenticators.

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

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

### 13.3. credentialId Unsigned

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

### 13.4. Browser Permissions Framework and Extensions

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

## 14. Privacy Considerations

The privacy principles in [FIDO-Privacy-Principles] also apply to this specification.

### 14.1. Attestation Privacy

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

- \* A WebAuthn authenticator manufacturer may choose to ship all of their devices with the same (or a fixed number of) attestation key(s) (called Basic Attestation). This will anonymize the user at the risk of not being able to revoke a particular attestation key if its private key is compromised. [UAFProtocol] requires that at least 100,000 devices share the same attestation certificate in order to produce sufficiently large groups. This may serve as guidance about suitable batch sizes.
- \* A WebAuthn authenticator may be capable of dynamically generating different attestation keys (and requesting related certificates) per-origin (similar to the Attestation CA approach). For example, an authenticator can ship with a master attestation key (and certificate), and combined with a cloud-operated Anonymization CA, can dynamically generate per-origin attestation keys and attestation certificates.

Note: In various places outside this specification, the term "Privacy CA" is used to refer to what is termed here as an Anonymization CA. Because the Trusted Computing Group (TCG) also

5621  
5622  
5623  
5624  
5625  
5626  
5627  
5628  
5629  
5630  
5631  
5632  
5633  
5634  
5635  
5636  
5637  
5638  
5639  
5640  
5641  
5642  
5643  
5644  
5645  
5646  
5647  
5648  
5649  
5650  
5651  
5652  
5653  
5654  
5655  
5656  
5657  
5658  
5659  
5660  
5661  
5662  
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

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

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

### 15. Acknowledgements

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

5058 Index  
5059

5060 Terms defined by this specification

- 5061 \* [aaguid](#), in 6.3.1
- 5062 \* AAGUID, in 10.4
- 5063
- 5064
- 5065 \* alg, in 5.3
- 5066 \* allowCredentials, in 5.5
- 5067
- 5068
- 5069
- 5070 \* Assertion, in 4
- 5071 \* assertion signature, in 6
- 5072 \* attachment modality, in 5.4.5
- 5073
- 5074 \* Attestation, in 4
- 5075 \* attestation, in 5.4
- 5076
- 5077 \* Attestation Certificate, in 4
- 5078 \* Attestation Conveyance, in 5.4.6
- 5079 \* AttestationConveyancePreference, in 5.4.6
- 5080 \* attestationConveyancePreferenceOption, in 5.1.3
- 5081 \* attestation key pair, in 4
- 5082 \* attestationObject, in 5.2.1
- 5083 \* attestation object, in 6.3
- 5084 \* attestationObjectResult, in 5.1.3
- 5085 \* attestation private key, in 4
- 5086 \* attestation public key, in 4
- 5087 \* attestation signature, in 6
- 5088 \* attestation statement, in 6.3
- 5089 \* attestation statement format, in 6.3
- 5090 \* attestation statement format identifier, in 8.1
- 5091 \* attestation trust path, in 6.3.2
- 5092 \* attestation type, in 6.3
- 5093 \* Attested credential data, in 6.3.1
- 5094 \* attestedCredentialData, in 6.1
- 5095 \* authDataExtensions, in 6.1
- 5096 \* Authentication, in 4
- 5097 \* Authentication Assertion, in 4
- 5098 \* authentication extension, in 9
- 5099 \* AuthenticationExtensions
- 5100
- 5101 + definition of, in 5.7
- 5102 + ([typedef](#)), in 5.7
- 5103
- 5104 \* Authenticator, in 4
- 5105 \* AuthenticatorAssertionResponse, in 5.2.2
- 5106 \* AuthenticatorAttachment, in 5.4.5
- 5107 \* authenticatorAttachment, in 5.4.4
- 5108 \* AuthenticatorAttestationResponse, in 5.2.1
- 5109 \* authenticatorCancel, in 6.2.3
- 5110
- 5111 \* authenticator data, in 6.1
- 5112 \* authenticatorData, in 5.2.2

5691 Johan Verrept, and Boris Zbarsky.

5692 We thank Anthony Nadalin, John Fontana, and Richard Barnes for their  
5693 contributions as co-chairs of the Web Authentication Working Group.

5694 We also thank Wendy Seltzer, Samuel Weiler, and Harry Halpin for their  
5695 contributions as our W3C Team Contacts.

5696 Index

5697 Terms defined by this specification

- 5700
- 5701 \* AAGUID, in 10.4
- 5702 \* [aaguid](#), in 6.3.1
- 5703 \* alg, in 5.3
- 5704 \* allowCredentials, in 5.5
- 5705 \* Anonymization CA, in 14.1
- 5706 \* appid
- 5707 + dict-member for AuthenticationExtensionsClientInputs, in 10.1
- 5708 + dict-member for AuthenticationExtensionsClientOutputs, in 10.1
- 5709
- 5710 \* Assertion, in 4
- 5711 \* assertion signature, in 6
- 5712 \* attachment modality, in 5.4.5
- 5713 \* [AttCA](#), in 6.3.3
- 5714 \* Attestation, in 4
- 5715 \* attestation, in 5.4
- 5716 \* [Attestation CA](#), in 6.3.3
- 5717 \* Attestation Certificate, in 4
- 5718 \* Attestation Conveyance, in 5.4.6
- 5719 \* AttestationConveyancePreference, in 5.4.6
- 5720 \* attestationConveyancePreferenceOption, in 5.1.3
- 5721 \* attestation key pair, in 4
- 5722 \* attestationObject, in 5.2.1
- 5723 \* attestation object, in 6.3
- 5724 \* attestationObjectResult, in 5.1.3
- 5725 \* attestation private key, in 4
- 5726 \* attestation public key, in 4
- 5727 \* attestation signature, in 6
- 5728 \* attestation statement, in 6.3
- 5729 \* attestation statement format, in 6.3
- 5730 \* attestation statement format identifier, in 8.1
- 5731 \* attestation trust path, in 6.3.2
- 5732 \* attestation type, in 6.3
- 5733 \* Attested credential data, in 6.3.1
- 5734 \* attestedCredentialData, in 6.1
- 5735 \* authDataExtensions, in 6.1
- 5736 \* Authentication, in 4
- 5737 \* Authentication Assertion, in 4
- 5738 \* authentication extension, in 9
- 5739 \* AuthenticationExtensions
- 5740
- 5741 \* [AuthenticationExtensionsAuthenticatorInputs](#)
- 5742 + definition of, in 5.9
- 5743 + ([typedef](#)), in 5.9
- 5744 \* [AuthenticationExtensionsClientInputs](#)
- 5745 + definition of, in 5.7
- 5746 + ([dictionary](#)), in 5.7
- 5747 \* [AuthenticationExtensionsClientOutputs](#)
- 5748 + definition of, in 5.8
- 5749 + ([dictionary](#)), in 5.8
- 5750 \* [AuthenticationExtensionsSupported](#), in 10.5
- 5751 \* Authenticator, in 4
- 5752 \* AuthenticatorAssertionResponse, in 5.2.2
- 5753 \* AuthenticatorAttachment, in 5.4.5
- 5754 \* authenticatorAttachment, in 5.4.4
- 5755 \* AuthenticatorAttestationResponse, in 5.2.1
- 5756 \* authenticatorBiometricPerfBounds, in 10.9
- 5757 \* [authenticatorCancel](#), in 6.2.4
- 5758 \* authenticator data, in 6.1
- 5759 \* authenticatorData, in 5.2.2

5105 \* authenticator data claimed to have been used for the attestation,  
5106 in 6.3.2  
5107 \* authenticator data for the attestation, in 6.3.2  
5108 \* authenticatorDataResult, in 5.1.4.1  
5109 \* authenticator extension, in 9  
5110 \* authenticator extension input, in 9.3  
5111 \* authenticator extension output, in 9.5  
5112 \* Authenticator extension processing, in 9.5  
5113 \* authenticatorExtensions, in 5.8.1  
5114 \* authenticatorGetAssertion, in 6.2.2  
5115 \* authenticatorMakeCredential, in 6.2.1

5116 \* AuthenticatorResponse, in 5.2  
5117 \* authenticatorSelection, in 5.4  
5118 \* AuthenticatorSelectionCriteria, in 5.4.4  
5119 \* AuthenticatorSelectionList, in 10.4  
5120 \* authenticator session, in 6.2  
5121 \* AuthenticatorTransport, in 5.8.4

5122 \* Authorization Gesture, in 4  
5123 \* Base64url Encoding, in 3

5124 \* Basic Attestation, in 6.3.3

5125 \* Biometric Recognition, in 4  
5126 \* ble, in 5.8.4  
5127 \* CBOR, in 3  
5128 \* Ceremony, in 4  
5129 \* challenge  
5130 + dict-member for MakePublicKeyCredentialOptions, in 5.4  
5131 + dict-member for PublicKeyCredentialRequestOptions, in 5.5  
5132 + dict-member for CollectedClientData, in 5.8.1  
5133 \* Client, in 4  
5134 \* client data, in 5.8.1  
5135 \* clientDataJSON, in 5.2  
5136 \* clientDataJSONResult  
5137 + dfn for credentialCreationData, in 5.1.3  
5138 + dfn for assertionCreationData, in 5.1.4.1  
5139 \* client extension, in 9  
5140 \* client extension input, in 9.3  
5141 \* client extension output, in 9.4  
5142 \* Client extension processing, in 9.4  
5143 \* clientExtensionResults  
5144 + dfn for credentialCreationData, in 5.1.3  
5145 + dfn for assertionCreationData, in 5.1.4.1  
5146 \* clientExtensions, in 5.8.1  
5147 \* [[clientExtensionsResults]], in 5.1  
5148 \* Client-Side, in 4  
5149 \* client-side credential private key storage, in 4  
5150 \* Client-side-resident Credential Private Key, in 4  
5151 \* CollectedClientData, in 5.8.1  
5152 \* [[CollectFromCredentialStore]](origin, options,  
5153 sameOriginWithAncestors), in 5.1.4  
5154 \* Conforming User Agent, in 4

5155 \* COSEAlgorithmIdentifier  
5156 + definition of, in 5.8.5  
5157 + (typedef), in 5.8.5  
5158 \* [[Create]](origin, options, sameOriginWithAncestors), in 5.1.3  
5159 \* Credential ID, in 4  
5160 \* credentialId, in 6.3.1  
5161 \* credentialIdLength, in 6.3.1  
5162 \* credentialIdResult, in 5.1.4.1  
5163 \* credential key pair, in 4  
5164 \* credential private key, in 4  
5165 \* Credential Public Key, in 4

5760 \* authenticator data claimed to have been used for the attestation,  
5761 in 6.3.2  
5762 \* authenticator data for the attestation, in 6.3.2  
5763 \* authenticatorDataResult, in 5.1.4.1  
5764 \* authenticator extension, in 9  
5765 \* authenticator extension input, in 9.3  
5766 \* authenticator extension output, in 9.5  
5767 \* Authenticator extension processing, in 9.5  
5768 \* authenticatorGetAssertion, in 6.2.3  
5769 \* authenticatorMakeCredential, in 6.2.2  
5770 \* Authenticator Model, in 6  
5771 \* Authenticator operations, in 6.2  
5772 \* AuthenticatorResponse, in 5.2  
5773 \* authenticatorSelection, in 5.4  
5774 \* AuthenticatorSelectionCriteria, in 5.4.4  
5775 \* AuthenticatorSelectionList, in 10.4  
5776 \* authenticator session, in 6.2  
5777 \* AuthenticatorTransport, in 5.10.4  
5778 \* authnSel  
5779 + dict-member for AuthenticationExtensionsClientInputs, in 10.4  
5780 + dict-member for AuthenticationExtensionsClientOutputs, in  
5781 10.4  
5782 \* Authorization Gesture, in 4  
5783 \* Base64url Encoding, in 3  
5784 \* Basic, in 6.3.3  
5785 \* Basic Attestation, in 6.3.3  
5786 \* Biometric Authenticator, in 4  
5787 \* Biometric Recognition, in 4  
5788 \* ble, in 5.10.4  
5789 \* CBOR, in 3  
5790 \* Ceremony, in 4  
5791 \* challenge  
5792 + dict-member for PublicKeyCredentialCreationOptions, in 5.4  
5793 + dict-member for PublicKeyCredentialRequestOptions, in 5.5  
5794 + dict-member for CollectedClientData, in 5.10.1  
5795 \* Client, in 4  
5796 \* client data, in 5.10.1  
5797 \* clientDataJSON, in 5.2  
5798 \* clientDataJSONResult  
5799 + dfn for credentialCreationData, in 5.1.3  
5800 + dfn for assertionCreationData, in 5.1.4.1  
5801 \* client extension, in 9  
5802 \* client extension input, in 9.3  
5803 \* client extension output, in 9.4  
5804 \* Client extension processing, in 9.4  
5805 \* clientExtensionResults  
5806 + dfn for credentialCreationData, in 5.1.3  
5807 + dfn for assertionCreationData, in 5.1.4.1

5808 \* [[clientExtensionsResults]], in 5.1  
5809 \* Client-Side, in 4  
5810 \* client-side credential private key storage, in 4  
5811 \* Client-side-resident Credential Private Key, in 4  
5812 \* CollectedClientData, in 5.10.1  
5813 \* [[CollectFromCredentialStore]](origin, options,  
5814 sameOriginWithAncestors), in 5.1.4  
5815 \* Conforming User Agent, in 4  
5816 \* content, in 10.3  
5817 \* contentType, in 10.3  
5818 \* COSEAlgorithmIdentifier  
5819 + definition of, in 5.10.5  
5820 + (typedef), in 5.10.5  
5821 \* [[Create]](origin, options, sameOriginWithAncestors), in 5.1.3  
5822 \* Credential ID, in 4  
5823 \* credentialId, in 6.3.1  
5824 \* credentialIdLength, in 6.3.1  
5825 \* credentialIdResult, in 5.1.4.1  
5826 \* credential key pair, in 4  
5827 \* credential private key, in 4  
5828 \* Credential Public Key, in 4

5166 \* credentialPublicKey, in 6.3.1

5167 \* "cross-platform", in 5.4.5

5168 \* cross-platform, in 5.4.5

5169 \* cross-platform attached, in 5.4.5

5170 \* cross-platform attachment, in 5.4.5

5171 \* DAA, in 6.3.3

5172 \* direct, in 5.4.6

5173 \* "discouraged", in 5.8.6

5174 \* discouraged, in 5.8.6

5175 \* [[DiscoverFromExternalSource]](origin, options,

5176 sameOriginWithAncestors), in 5.1.4.1

5177 \* [[discovery]], in 5.1

5178 \* displayName, in 5.4.3

5179 \* ECDAAs, in 6.3.3

5180 \* ECDAAs-Issuer public key, in 8.2

5181 \* effective user verification requirement for assertion, in 5.1.4.1

5182 \* effective user verification requirement for credential creation, in

5183 5.1.3

5184 \* Elliptic Curve based Direct Anonymous Attestation, in 6.3.3

5185 \* excludeCredentials, in 5.4

5186 \* extension identifier, in 9.1

5187 \* extensions

5188 + dict-member for MakePublicKeyCredentialOptions, in 5.4

5189 + dict-member for PublicKeyCredentialRequestOptions, in 5.5

5190

5191 \* flags, in 6.1

5192 \* getClientExtensionResults(), in 5.1

5193 \* hashAlgorithm, in 5.8.1

5194 \* Hash of the serialized client data, in 5.8.1

5195 \* icon, in 5.4.1

5196 \* id

5197 + dict-member for PublicKeyCredentialRpEntity, in 5.4.2

5198 + dict-member for PublicKeyCredentialUserEntity, in 5.4.3

5199 + dict-member for PublicKeyCredentialDescriptor, in 5.8.3

5200

5201 \* [[identifier]], in 5.1

5202 \* identifier of the ECDAAs-Issuer public key, in 8.2

5203 \* indirect, in 5.4.6

5204 \* isUserVerifyingPlatformAuthenticatorAvailable(), in 5.1.6

5205 \* JSON-serialized client data, in 5.8.1

5206 \* MakePublicKeyCredentialOptions, in 5.4

5207

5208 \* managing authenticator, in 4

5209 \* name, in 5.4.1

5210 \* nfc, in 5.8.4

5211

5212 \* none, in 5.4.6

5213 \* origin, in 5.8.1

5214

5215 \* platform, in 5.4.5

5216 \* "platform", in 5.4.5

5217 \* platform attachment, in 5.4.5

5218 \* platform authenticators, in 5.4.5

5219 \* "preferred", in 5.8.6

5220 \* preferred, in 5.8.6

5221 \* Privacy CA, in 6.3.3

5829 \* credentialPublicKey, in 6.3.1

5830 \* credentials map, in 6

5831 \* "cross-platform", in 5.4.5

5832 \* cross-platform, in 5.4.5

5833 \* cross-platform attached, in 5.4.5

5834 \* cross-platform attachment, in 5.4.5

5835 \* DAA, in 6.3.3

5836 \* direct, in 5.4.6

5837 \* "discouraged", in 5.10.6

5838 \* discouraged, in 5.10.6

5839 \* [[DiscoverFromExternalSource]](origin, options,

5840 sameOriginWithAncestors), in 5.1.4.1

5841 \* [[discovery]], in 5.1

5842 \* displayName, in 5.4.3

5843 \* ECDAAs, in 6.3.3

5844 \* ECDAAs-Issuer public key, in 8.2

5845 \* effective user verification requirement for assertion, in 5.1.4.1

5846 \* effective user verification requirement for credential creation, in

5847 5.1.3

5848 \* Elliptic Curve based Direct Anonymous Attestation, in 6.3.3

5849 \* excludeCredentials, in 5.4

5850 \* extension identifier, in 9.1

5851 \* extensions

5852 + dict-member for PublicKeyCredentialCreationOptions, in 5.4

5853 + dict-member for PublicKeyCredentialRequestOptions, in 5.5

5854 \* exts

5855 + dict-member for AuthenticationExtensionsClientInputs, in 10.5

5856 + dict-member for AuthenticationExtensionsClientOutputs, in

5857 10.5

5858 \* FAR, in 10.9

5859 \* flags, in 6.1

5860 \* FRR, in 10.9

5861 \* getClientExtensionResults(), in 5.1

5862 \* Hash of the serialized client data, in 5.10.1

5863 \* Human Palatability, in 4

5864 \* icon, in 5.4.1

5865 \* id

5866 + dfn for public key credential source, in 4

5867 + dict-member for PublicKeyCredentialRpEntity, in 5.4.2

5868 + dict-member for PublicKeyCredentialUserEntity, in 5.4.3

5869 + dict-member for TokenBinding, in 5.10.1

5870 + dict-member for PublicKeyCredentialDescriptor, in 5.10.3

5871 \* [[identifier]], in 5.1

5872 \* identifier of the ECDAAs-Issuer public key, in 8.2

5873 \* indirect, in 5.4.6

5874 \* isUserVerifyingPlatformAuthenticatorAvailable(), in 5.1.7

5875 \* JSON-serialized client data, in 5.10.1

5876 \* loc

5877 + dict-member for AuthenticationExtensionsClientInputs, in 10.7

5878 + dict-member for AuthenticationExtensionsClientOutputs, in

5879 10.7

5880 \* looking up, in 6.2.1

5881 \* managing authenticator, in 4

5882 \* name, in 5.4.1

5883 \* nfc, in 5.10.4

5884 \* No attestation statement, in 6.3.3

5885 \* None, in 6.3.3

5886 \* none, in 5.4.6

5887 \* none attestation statement format, in 8.7

5888 \* "not-supported", in 5.10.1

5889 \* not-supported, in 5.10.1

5890 \* origin, in 5.10.1

5891 \* otherUI, in 4

5892 \* platform, in 5.4.5

5893 \* "platform", in 5.4.5

5894 \* platform attachment, in 5.4.5

5895 \* platform authenticators, in 5.4.5

5896 \* platform credential, in 5.4.5

5897 \* "preferred", in 5.10.6

5898 \* preferred, in 5.10.6

5217 \* pubKeyCredParams, in 5.4  
 5218 \* publicKey  
 5219 + dict-member for CredentialCreationOptions, in 5.1.1  
 5220 + dict-member for CredentialRequestOptions, in 5.1.2  
 5221 \* public-key, in 5.8.2  
 5222 \* Public Key Credential, in 4  
 5223 \* PublicKeyCredential, in 5.1  
 5224 \* PublicKeyCredentialDescriptor, in 5.8.3

5225 \* PublicKeyCredentialEntity, in 5.4.1  
 5226 \* PublicKeyCredentialParameters, in 5.3  
 5227 \* PublicKeyCredentialRequestOptions, in 5.5  
 5228 \* PublicKeyCredentialRpEntity, in 5.4.2  
 5229 \* Public Key Credential Source, in 4  
 5230 \* PublicKeyCredentialType, in 5.8.2  
 5231 \* PublicKeyCredentialUserEntity, in 5.4.3  
 5232 \* Rate Limiting, in 4  
 5233 \* rawId, in 5.1  
 5234 \* Registration, in 4  
 5235 \* registration extension, in 9  
 5236 \* Relying Party, in 4  
 5237 \* Relying Party Identifier, in 4  
 5238 \* "required", in 5.8.6  
 5239 \* required, in 5.8.6  
 5240 \* requireResidentKey, in 5.4.4  
 5241 \* response, in 5.1  
 5242 \* roaming authenticators, in 5.4.5

5243 \* rp, in 5.4  
 5244 \* rpId, in 5.5

5245 \* RP ID, in 4  
 5246 \* rpIdHash, in 6.1

5247 \* Self Attestation, in 6.3.3  
 5248 \* signature, in 5.2.2  
 5249 \* Signature Counter, in 6.1.1  
 5250 \* signatureResult, in 5.1.4.1  
 5251 \* signCount, in 6.1  
 5252 \* Signing procedure, in 6.3.2

5253 \* [[Store]](credential, sameOriginWithAncestors), in 5.1.5

5254 \* Test of User Presence, in 4  
 5255 \* timeout  
 5256 + dict-member for MakePublicKeyCredentialOptions, in 5.4  
 5257 + dict-member for PublicKeyCredentialRequestOptions, in 5.5  
 5258 \* tokenBindingId, in 5.8.1  
 5259 \* transports, in 5.8.3

5260 \* [[type]], in 5.1  
 5261 \* type

5899 \* "present", in 5.10.1  
 5900 \* present, in 5.10.1  
 5901 \* [[preventSilentAccess]](credential, sameOriginWithAncestors), in  
 5902 5.1.6  
 5903 \* privateKey, in 4  
 5904 \* pubKeyCredParams, in 5.4  
 5905 \* publicKey  
 5906 + dict-member for CredentialCreationOptions, in 5.1.1  
 5907 + dict-member for CredentialRequestOptions, in 5.1.2  
 5908 \* public-key, in 5.10.2  
 5909 \* Public Key Credential, in 4  
 5910 \* PublicKeyCredential, in 5.1  
 5911 \* PublicKeyCredentialCreationOptions, in 5.4  
 5912 \* PublicKeyCredentialDescriptor, in 5.10.3  
 5913 \* PublicKeyCredentialEntity, in 5.4.1  
 5914 \* PublicKeyCredentialParameters, in 5.3  
 5915 \* PublicKeyCredentialRequestOptions, in 5.5  
 5916 \* PublicKeyCredentialRpEntity, in 5.4.2  
 5917 \* Public Key Credential Source, in 4  
 5918 \* PublicKeyCredentialType, in 5.10.2  
 5919 \* PublicKeyCredentialUserEntity, in 5.4.3  
 5920 \* Rate Limiting, in 4  
 5921 \* rawId, in 5.1  
 5922 \* Registration, in 4  
 5923 \* registration extension, in 9  
 5924 \* Relying Party, in 4  
 5925 \* Relying Party Identifier, in 4  
 5926 \* "required", in 5.10.6  
 5927 \* required, in 5.10.6  
 5928 \* requireResidentKey, in 5.4.4  
 5929 \* response, in 5.1  
 5930 \* roaming authenticators, in 5.4.5  
 5931 \* roaming credential, in 5.4.5  
 5932 \* rp, in 5.4  
 5933 \* rpId  
 5934 + dfn for public key credential source, in 4  
 5935 + dict-member for PublicKeyCredentialRequestOptions, in 5.5  
 5936 \* RP ID, in 4  
 5937 \* rpIdHash, in 6.1  
 5938 \* Self, in 6.3.3  
 5939 \* Self Attestation, in 6.3.3  
 5940 \* signature, in 5.2.2  
 5941 \* Signature Counter, in 6.1.1  
 5942 \* signatureResult, in 5.1.4.1  
 5943 \* signCount, in 6.1  
 5944 \* Signing procedure, in 6.3.2  
 5945 \* status, in 5.10.1  
 5946 \* [[Store]](credential, sameOriginWithAncestors), in 5.1.5  
 5947 \* supported, in 5.10.1  
 5948 \* "supported", in 5.10.1  
 5949 \* Test of User Presence, in 4  
 5950 \* timeout  
 5951 + dict-member for PublicKeyCredentialCreationOptions, in 5.4  
 5952 + dict-member for PublicKeyCredentialRequestOptions, in 5.5  
 5953 \* tokenBinding, in 5.10.1  
 5954 \* TokenBinding, in 5.10.1  
 5955 \* TokenBindingStatus, in 5.10.1  
 5956 \* transports, in 5.10.3  
 5957 \* txAuthGeneric  
 5958 + dict-member for AuthenticationExtensionsClientInputs, in 10.3  
 5959 + dict-member for AuthenticationExtensionsClientOutputs, in  
 5960 10.3  
 5961 \* txAuthGenericArg, in 10.3  
 5962 \* txAuthSimple  
 5963 + dict-member for AuthenticationExtensionsClientInputs, in 10.2  
 5964 + dict-member for AuthenticationExtensionsClientOutputs, in  
 5965 10.2  
 5966 \* [[type]], in 5.1  
 5967 \* type  
 5968 + dfn for public key credential source, in 4

- 5262 + dict-member for PublicKeyCredentialParameters, in 5.3
  - 5263 + dict-member for CollectedClientData, in 5.8.1
  - 5264 + dict-member for PublicKeyCredentialDescriptor, in 5.8.3
  - 5265 \* UP, in 4
  - 5266 \* usb, in 5.8.4
  - 5267 \* user, in 5.4
  - 5268 \* User Consent, in 4
  - 5269 \* userHandle, in 5.2.2
- 
- 5270 \* User Handle, in 4
  - 5271 \* userHandleResult, in 5.1.4.1
  - 5272 \* User Present, in 4
  - 5273 \* userVerification
  - 5274 + dict-member for AuthenticatorSelectionCriteria, in 5.4.4
  - 5275 + dict-member for PublicKeyCredentialRequestOptions, in 5.5
  - 5276 \* User Verification, in 4
  - 5277 \* UserVerificationRequirement, in 5.8.6
  - 5278 \* User Verified, in 4
  - 5279 \* UV, in 4

- 5280 \* Verification procedure, in 6.3.2
- 5281 \* verification procedure inputs, in 6.3.2
- 5282 \* Web Authentication API, in 5
- 5283 \* WebAuthn Client, in 4

Terms defined by reference

- 5285 \* [CREDENTIAL-MANAGEMENT-1] defines the following terms:
- 5286 + Credential
- 5287 + CredentialCreationOptions
- 5288 + CredentialRequestOptions
- 5289 + CredentialsContainer
- 5290 + Request a Credential
- 5291 + [[CollectFromCredentialStore]](origin, options, sameOriginWithAncestors)
- 5292 + [[Create]](origin, options, sameOriginWithAncestors)
- 5293 + [[Store]](credential, sameOriginWithAncestors)
- 5294 + [[discovery]]
- 5295 + [[type]]
- 5296 + create()
- 5297 + credential
- 5298 + credential source
- 5299 + get()
- 5300 + id
- 5301 + remote
- 5302 + same-origin with its ancestors
- 5303 + signal (for CredentialCreationOptions)
- 5304 + signal (for CredentialRequestOptions)
- 5305 + store()
- 5306 + type
- 5307 + user mediation
- 5308 \* [DOM4] defines the following terms:
- 5309 + AbortController
- 5310 + aborted flag
- 5311 + document
- 5312 \* [ECMAScript] defines the following terms:
- 5313 + %arraybuffer%
- 5314 + internal method
- 5315 + internal slot
- 5316 + stringify

- 5969 + dict-member for PublicKeyCredentialParameters, in 5.3
- 5970 + dict-member for CollectedClientData, in 5.10.1
- 5971 + dict-member for PublicKeyCredentialDescriptor, in 5.10.3
- 5972 \* UP, in 4
- 5973 \* usb, in 5.10.4
- 5974 \* user, in 5.4
- 5975 \* User Consent, in 4
- 5976 \* userHandle
- 5977 + dfn for public key credential source, in 4
- 5978 + attribute for AuthenticatorAssertionResponse, in 5.2.2
- 5979 \* User Handle, in 4
- 5980 \* userHandleResult, in 5.1.4.1
- 5981 \* User Present, in 4
- 5982 \* userVerification
- 5983 + dict-member for AuthenticatorSelectionCriteria, in 5.4.4
- 5984 + dict-member for PublicKeyCredentialRequestOptions, in 5.5
- 5985 \* User Verification, in 4
- 5986 \* UserVerificationRequirement, in 5.10.6
- 5987 \* User Verified, in 4
- 5988 \* UV, in 4
- 5989 \* uvi
- 5990 + dict-member for AuthenticationExtensionsClientInputs, in 10.6
- 5991 + dict-member for AuthenticationExtensionsClientOutputs, in 10.6
- 5992 \* uvm
- 5993 + dict-member for AuthenticationExtensionsClientInputs, in 10.8
- 5994 + dict-member for AuthenticationExtensionsClientOutputs, in 10.8
- 5995 \* UvmEntries, in 10.8
- 5996 \* UvmEntry, in 10.8
- 5997 \* Verification procedure, in 6.3.2
- 5998 \* verification procedure inputs, in 6.3.2
- 5999 \* Web Authentication API, in 5
- 6000 \* WebAuthn Client, in 4

Terms defined by reference

- 6001 \* [CREDENTIAL-MANAGEMENT-1] defines the following terms:
- 6002 + Credential
- 6003 + CredentialCreationOptions
- 6004 + CredentialRequestOptions
- 6005 + CredentialsContainer
- 6006 + Request a Credential
- 6007 + [[CollectFromCredentialStore]](origin, options, sameOriginWithAncestors)
- 6008 + [[Create]](origin, options, sameOriginWithAncestors)
- 6009 + [[Store]](credential, sameOriginWithAncestors)
- 6010 + [[discovery]]
- 6011 + [[type]]
- 6012 + create()
- 6013 + credential
- 6014 + credential source
- 6015 + get()
- 6016 + id
- 6017 + remote
- 6018 + same-origin with its ancestors
- 6019 + signal (for CredentialCreationOptions)
- 6020 + signal (for CredentialRequestOptions)
- 6021 + store()
- 6022 + type
- 6023 + user mediation
- 6024 \* [DOM4] defines the following terms:
- 6025 + AbortController
- 6026 + aborted flag
- 6027 + document
- 6028 \* [ECMAScript] defines the following terms:
- 6029 + %arraybuffer%
- 6030 + internal method
- 6031 + internal slot
- 6032 + stringify

5320 \* [ENCODING] defines the following terms:  
5321 + utf-8 encode  
5322 \* [FETCH] defines the following terms:  
5323 + window

5324 \* [HTML] defines the following terms:  
5325 + ascii serialization of an origin  
5326 + effective domain  
5327 + environment settings object  
5328 + global object  
5329 + is a registrable domain suffix of or is equal to  
5330 + is not a registrable domain suffix of and is not equal to  
5331 + origin  
5332 + relevant settings object  
5333 \* [HTML52] defines the following terms:  
5334 + document.domain  
5335 + opaque origin  
5336 + origin  
5337 \* [INFRA] defines the following terms:  
5338 + append (for list)  
5339 + append (for set)  
5340 + byte sequence  
5341 + continue  
5342 + empty  
5343 + for each (for list)  
5344 + for each (for map)  
5345 + is empty  
5346 + is not empty  
5347 + item (for list)  
5348 + item (for struct)  
5349 + list  
5350 + map  
5351 + ordered set  
5352 + remove  
5353 + set  
5354 + size  
5355 + struct  
5356 + while  
5357 + willful violation  
5358 \* [mixed-content] defines the following terms:  
5359 + a priori authenticated url  
5360 \* [page-visibility] defines the following terms:  
5361 + visibility states  
5362 \* [secure-contexts] defines the following terms:  
5363 + secure contexts  
5364 \* [TokenBinding] defines the following terms:  
5365 + token binding  
5366 + token binding id  
5367 \* [URL] defines the following terms:  
5368 + domain  
5369 + empty host  
5370 + host  
5371 + ipv4 address  
5372 + ipv6 address  
5373 + opaque host  
5374 + url serializer  
5375 + valid domain  
5376 + valid domain string  
5377 \* [WebCryptoAPI] defines the following terms:  
5378 + recognized algorithm name  
5379 \* [WebIDL] defines the following terms:  
5380 + AbortError  
5381 + ArrayBuffer

6039 \* [ENCODING] defines the following terms:  
6040 + utf-8 decode  
6041 + utf-8 encode  
6042 \* [FETCH] defines the following terms:  
6043 + window  
6044 \* [FIDO-APPID] defines the following terms:  
6045 + determining if a caller's facetid is authorized for an appid  
6046 + determining the facetid of a calling application  
6047 \* [FIDO-CTAP] defines the following terms:  
6048 + ctap2 canonical cbor encoding form  
6049 \* [Geolocation-API] defines the following terms:  
6050 + Coordinates  
6051 \* [HTML] defines the following terms:  
6052 + ascii serialization of an origin  
6053 + effective domain  
6054 + environment settings object  
6055 + global object  
6056 + is a registrable domain suffix of or is equal to  
6057 + is not a registrable domain suffix of and is not equal to  
6058 + origin  
6059 + relevant settings object  
6060 \* [HTML52] defines the following terms:  
6061 + document.domain  
6062 + opaque origin  
6063 + origin  
6064 \* [INFRA] defines the following terms:  
6065 + append (for list)  
6066 + append (for set)  
6067 + byte sequence  
6068 + continue  
6069 + for each (for list)  
6070 + for each (for map)  
6071 + is empty  
6072 + is not empty  
6073 + item (for list)  
6074 + item (for struct)  
6075 + list  
6076 + map  
6077 + ordered set  
6078 + remove  
6079 + set  
6080 + set (for map)  
6081 + struct  
6082 + while  
6083 + willful violation  
6084 \* [mixed-content] defines the following terms:  
6085 + a priori authenticated url  
6086 \* [page-visibility] defines the following terms:  
6087 + visibility states  
6088 \* [secure-contexts] defines the following terms:  
6089 + secure contexts  
6090 \* [TokenBinding] defines the following terms:  
6091 + token binding  
6092 + token binding id  
6093 \* [URL] defines the following terms:  
6094 + domain  
6095 + empty host  
6096 + host  
6097 + ipv4 address  
6098 + ipv6 address  
6099 + opaque host  
6100 + url serializer  
6101 + valid domain  
6102 + valid domain string

6103 \* [WebIDL] defines the following terms:  
6104 + AbortError  
6105 + ArrayBuffer

5382 + BufferSource  
 5383 + ConstraintError  
 5384 + DOMException  
 5385 + DOMString  
 5386 + Exposed

5387 + NotAllowedError  
 5388 + NotSupportedError  
 5389 + Promise  
 5390 + SameObject  
 5391 + SecureContext  
 5392 + SecurityError  
 5393 + USVString  
 5394 + UnknownError  
 5395 + boolean

5396 + interface object  
 5397 + long  
 5398 + present  
 5399 + unsigned long  
 5400 \* [whatwg html] defines the following terms:  
 5401 + focus

5402  
 5403 References

5404 Normative References

5405 [CDDL]  
 5406 C. Vigano; H. Birkholz. CBOR data definition language (CDDL): a  
 5407 notational convention to express CBOR data structures. 21  
 5408 September 2016. Internet Draft (work in progress). URL:  
 5409 <https://tools.ietf.org/html/draft-greevenbosch-appsawg-cbor-cddl>

5410 [CREDENTIAL-MANAGEMENT-1]  
 5411 Mike West. Credential Management Level 1. 4 August 2017. WD.  
 5412 URL: <https://www.w3.org/TR/credential-management-1/>

5413 [DOM4]  
 5414 Anne van Kesteren. DOM Standard. Living Standard. URL:  
 5415 <https://dom.spec.whatwg.org/>

5416 [ECMAScript]  
 5417 ECMAScript Language Specification. URL:  
 5418 <https://tc39.github.io/ecma262/>

5419 [ENCODING]  
 5420 Anne van Kesteren. Encoding Standard. Living Standard. URL:  
 5421 <https://encoding.spec.whatwg.org/>

5422 [FETCH]  
 5423 Anne van Kesteren. Fetch Standard. Living Standard. URL:  
 5424 <https://fetch.spec.whatwg.org/>

5433 [FIDO-CTAP]  
 5434 R. Lindemann; et al. FIDO 2.0: Client to Authenticator Protocol.  
 5435 FIDO Alliance [Review Draft](https://fidoalliance.org/specs/fido-v2.0-rd-20170927/fido-client-to-authenticator-protocol-v2.0-rd-20170927.html). URL:  
 5436 [https://fidoalliance.org/specs/fido-v2.0-rd-20170927/fido-client](https://fidoalliance.org/specs/fido-v2.0-rd-20170927/fido-client-to-authenticator-protocol-v2.0-rd-20170927.html)  
 5437 [-to-authenticator-protocol-v2.0-rd-20170927.html](https://fidoalliance.org/specs/fido-v2.0-rd-20170927/fido-client-to-authenticator-protocol-v2.0-rd-20170927.html)

6106 + BufferSource  
 6107 + ConstraintError  
 6108 + DOMException  
 6109 + DOMString  
 6110 + Exposed  
 6111 + InvalidStateError  
 6112 + NotAllowedError  
 6113 + NotSupportedError  
 6114 + Promise  
 6115 + SameObject  
 6116 + SecureContext  
 6117 + SecurityError  
 6118 + USVString  
 6119 + UnknownError  
 6120 + boolean  
 6121 + float  
 6122 + interface object  
 6123 + long  
 6124 + present  
 6125 + unsigned long  
 6126 \* [whatwg html] defines the following terms:  
 6127 + focus  
 6128 + username

6129  
 6130 References

6131 Normative References

6132 [CDDL]  
 6133 C. Vigano; H. Birkholz. CBOR data definition language (CDDL): a  
 6134 notational convention to express CBOR data structures. 21  
 6135 September 2016. Internet Draft (work in progress). URL:  
 6136 <https://tools.ietf.org/html/draft-greevenbosch-appsawg-cbor-cddl>

6137 [CREDENTIAL-MANAGEMENT-1]  
 6138 Mike West. Credential Management Level 1. 4 August 2017. WD.  
 6139 URL: <https://www.w3.org/TR/credential-management-1/>

6140 [DOM4]  
 6141 Anne van Kesteren. DOM Standard. Living Standard. URL:  
 6142 <https://dom.spec.whatwg.org/>

6143 [ECMAScript]  
 6144 ECMAScript Language Specification. URL:  
 6145 <https://tc39.github.io/ecma262/>

6146 [ENCODING]  
 6147 Anne van Kesteren. Encoding Standard. Living Standard. URL:  
 6148 <https://encoding.spec.whatwg.org/>

6149 [FETCH]  
 6150 Anne van Kesteren. Fetch Standard. Living Standard. URL:  
 6151 <https://fetch.spec.whatwg.org/>

6152 [FIDO-APPID]  
 6153 D. Balfanz; et al. FIDO AppID and Facets. FIDO Alliance Proposed  
 6154 Standard. URL:  
 6155 [https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-appid-](https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-appid-and-facets-v2.0-ps-20170927.html)  
 6156 [and-facets-v2.0-ps-20170927.html](https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-appid-and-facets-v2.0-ps-20170927.html)

6157 [FIDO-CTAP]  
 6158 R. Lindemann; et al. FIDO 2.0: Client to Authenticator Protocol.  
 6159 FIDO Alliance [Proposed Standard](https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-client-to-authenticator-protocol-v2.0-ps-20170927.html). URL:  
 6160 [https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-client](https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-client-to-authenticator-protocol-v2.0-ps-20170927.html)  
 6161 [-to-authenticator-protocol-v2.0-ps-20170927.html](https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-client-to-authenticator-protocol-v2.0-ps-20170927.html)

6162 [FIDO-Privacy-Principles]  
 6163 FIDO Alliance. FIDO Privacy Principles. FIDO Alliance  
 6164 Whitepaper. URL:  
 6165 [https://fidoalliance.org/wp-content/uploads/2014/12/FIDO\\_Allianc](https://fidoalliance.org/wp-content/uploads/2014/12/FIDO_Allianc)

5438 [FIDO-U2F-Message-Formats]  
5439 D. Balfanz; J. Ehrensvard; J. Lang. FIDO U2F Raw Message  
5440 Formats. FIDO Alliance Implementation Draft. URL:  
5441 https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2  
5442 f-raw-message-formats-v1.1-id-20160915.html  
5443  
5444 [FIDOEcdaaAlgorithm]  
5445 R. Lindemann; et al. FIDO ECDA Algorithm. FIDO Alliance  
5446 Implementation Draft. URL:  
5447 https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ec  
5448 daa-algorithm-v1.1-id-20170202.html  
5449  
5450 [FIDOReg]  
5451 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO UAF Registry of  
5452 Predefined Values. FIDO Alliance Proposed Standard. URL:  
5453 https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua  
5454 f-reg-v1.0-ps-20141208.html  
5455  
5456  
5457 [HTML]  
5458 Anne van Kesteren; et al. HTML Standard. Living Standard. URL:  
5459 https://html.spec.whatwg.org/multipage/  
5460  
5461 [HTML52]  
5462 Steve Faulkner; et al. HTML 5.2. 2 November 2017. PR. URL:  
5463 https://www.w3.org/TR/html52/  
5464  
5465 [IANA-COSE-ALGS-REG]  
5466 IANA CBOR Object Signing and Encryption (COSE) Algorithms  
5467 Registry. URL:  
5468 https://www.iana.org/assignments/cose/cose.xhtml#algorithms  
5469  
5470 [INFRA]  
5471 Anne van Kesteren; Domenic Denicola. Infra Standard. Living  
5472 Standard. URL: https://infra.spec.whatwg.org/  
5473  
5474 [MIXED-CONTENT]  
5475 Mike West. Mixed Content. 2 August 2016. CR. URL:  
5476 https://www.w3.org/TR/mixed-content/  
5477  
5478 [PAGE-VISIBILITY]  
5479 Jatinder Mann; Arvind Jain. Page Visibility (Second Edition). 29  
5480 October 2013. REC. URL: https://www.w3.org/TR/page-visibility/  
5481  
5482 [RFC2119]  
5483 S. Bradner. Key words for use in RFCs to Indicate Requirement  
5484 Levels. March 1997. Best Current Practice. URL:  
5485 https://tools.ietf.org/html/rfc2119  
5486  
5487 [RFC4648]  
5488 S. Josefsson. The Base16, Base32, and Base64 Data Encodings.  
5489 October 2006. Proposed Standard. URL:  
5490 https://tools.ietf.org/html/rfc4648  
5491  
5492 [RFC5234]  
5493 D. Crocker, Ed.; P. Overell. Augmented BNF for Syntax  
5494 Specifications: ABNF. January 2008. Internet Standard. URL:  
5495 https://tools.ietf.org/html/rfc5234  
5496  
5497 [RFC5890]  
5498 J. Klensin. Internationalized Domain Names for Applications  
5499 (IDNA): Definitions and Document Framework. August 2010.  
5500 Proposed Standard. URL: https://tools.ietf.org/html/rfc5890  
5501  
5502 [RFC7049]

6176 e\_Whitepaper\_Privacy\_Principles.pdf  
6177  
6178 [FIDO-U2F-Message-Formats]  
6179 D. Balfanz; J. Ehrensvard; J. Lang. FIDO U2F Raw Message  
6180 Formats. FIDO Alliance Implementation Draft. URL:  
6181 https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2  
6182 f-raw-message-formats-v1.1-id-20160915.html  
6183  
6184 [FIDOEcdaaAlgorithm]  
6185 R. Lindemann; et al. FIDO ECDA Algorithm. FIDO Alliance  
6186 Implementation Draft. URL:  
6187 https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ec  
6188 daa-algorithm-v1.1-id-20170202.html  
6189  
6190 [FIDOReg]  
6191 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO UAF Registry of  
6192 Predefined Values. FIDO Alliance Proposed Standard. URL:  
6193 https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua  
6194 f-reg-v1.0-ps-20141208.html  
6195  
6196 [Geolocation-API]  
6197 Andrei Popescu. Geolocation API Specification 2nd Edition. 8  
6198 November 2016. REC. URL: https://www.w3.org/TR/geolocation-API/  
6199  
6200 [HTML]  
6201 Anne van Kesteren; et al. HTML Standard. Living Standard. URL:  
6202 https://html.spec.whatwg.org/multipage/  
6203  
6204 [HTML52]  
6205 Steve Faulkner; et al. HTML 5.2. 14 December 2017. REC. URL:  
6206 https://www.w3.org/TR/html52/  
6207  
6208 [IANA-COSE-ALGS-REG]  
6209 IANA CBOR Object Signing and Encryption (COSE) Algorithms  
6210 Registry. URL:  
6211 https://www.iana.org/assignments/cose/cose.xhtml#algorithms  
6212  
6213 [INFRA]  
6214 Anne van Kesteren; Domenic Denicola. Infra Standard. Living  
6215 Standard. URL: https://infra.spec.whatwg.org/  
6216  
6217 [MIXED-CONTENT]  
6218 Mike West. Mixed Content. 2 August 2016. CR. URL:  
6219 https://www.w3.org/TR/mixed-content/  
6220  
6221 [PAGE-VISIBILITY]  
6222 Jatinder Mann; Arvind Jain. Page Visibility (Second Edition). 29  
6223 October 2013. REC. URL: https://www.w3.org/TR/page-visibility/  
6224  
6225 [RFC2119]  
6226 S. Bradner. Key words for use in RFCs to Indicate Requirement  
6227 Levels. March 1997. Best Current Practice. URL:  
6228 https://tools.ietf.org/html/rfc2119  
6229  
6230 [RFC4648]  
6231 S. Josefsson. The Base16, Base32, and Base64 Data Encodings.  
6232 October 2006. Proposed Standard. URL:  
6233 https://tools.ietf.org/html/rfc4648  
6234  
6235 [RFC5234]  
6236 D. Crocker, Ed.; P. Overell. Augmented BNF for Syntax  
6237 Specifications: ABNF. January 2008. Internet Standard. URL:  
6238 https://tools.ietf.org/html/rfc5234  
6239  
6240 [RFC5890]  
6241 J. Klensin. Internationalized Domain Names for Applications  
6242 (IDNA): Definitions and Document Framework. August 2010.  
6243 Proposed Standard. URL: https://tools.ietf.org/html/rfc5890  
6244  
6245 [RFC7049]

5503 C. Bormann; P. Hoffman. Concise Binary Object Representation  
 5504 (CBOR). October 2013. Proposed Standard. URL:  
 5505 <https://tools.ietf.org/html/rfc7049>  
 5506  
 5507 [RFC8152]  
 5508 J. Schaad. CBOR Object Signing and Encryption (COSE). July 2017.  
 5509 Proposed Standard. URL: <https://tools.ietf.org/html/rfc8152>  
 5510  
 5511 [SEC1]  
 5512 SEC1: Elliptic Curve Cryptography, Version 2.0. URL:  
 5513 <http://www.secg.org/sec1-v2.pdf>  
 5514  
 5515 [SECURE-CONTEXTS]  
 5516 Mike West. Secure Contexts. 15 September 2016. CR. URL:  
 5517 <https://www.w3.org/TR/secure-contexts/>  
 5518  
 5519 [TokenBinding]  
 5520 A. Popov; et al. The Token Binding Protocol Version 1.0.  
 5521 February 16, 2017. Internet-Draft. URL:  
 5522 <https://tools.ietf.org/html/draft-ietf-tokbind-protocol>  
 5523  
 5524 [URL]  
 5525 Anne van Kesteren. URL Standard. Living Standard. URL:  
 5526 <https://url.spec.whatwg.org/>  
 5527  
 5528 [WebAuthn-Registries]  
 5529 Jeff Hodges; Giridhar Mandyam; Michael B. Jones. Registries for  
 5530 Web Authentication (WebAuthn). March 2017. Active  
 5531 Internet-Draft. URL:  
 5532 [https://xml2rfc.tools.ietf.org/cgi-bin/xml2rfc.cgi?modeAsFormat=  
 5533 html&url=https://raw.githubusercontent.com/w3c/webauthn/ma  
 5534 ster/draft-hodges-webauthn-registries.xml](https://xml2rfc.tools.ietf.org/cgi-bin/xml2rfc.cgi?modeAsFormat=html&url=https://raw.githubusercontent.com/w3c/webauthn/master/draft-hodges-webauthn-registries.xml)  
 5535  
 5536 [WebCryptoAPI]  
 5537 Mark Watson. Web Cryptography API. 26 January 2017. REC. URL:  
 5538 <https://www.w3.org/TR/WebCryptoAPI/>  
 5539  
 5540 [WebIDL]  
 5541 Cameron McCormack; Boris Zbarsky; Tobie Langel. Web IDL. 15  
 5542 December 2016. ED. URL: <https://heycam.github.io/webidl/>  
 5543  
 5544 [WebIDL-1]  
 5545 Cameron McCormack. WebIDL Level 1. 15 December 2016. REC. URL:  
 5546 <https://www.w3.org/TR/2016/REC-WebIDL-1-20161215/>  
 5547  
 5548 Informative References  
 5549  
 5550 [Ceremony]  
 5551 Carl Ellison. Ceremony Design and Analysis. 2007. URL:  
 5552 <https://eprint.iacr.org/2007/399.pdf>  
 5553

5554 [Feature-Policy]

6246 C. Bormann; P. Hoffman. Concise Binary Object Representation  
 6247 (CBOR). October 2013. Proposed Standard. URL:  
 6248 <https://tools.ietf.org/html/rfc7049>  
 6249  
 6250 [RFC8152]  
 6251 J. Schaad. CBOR Object Signing and Encryption (COSE). July 2017.  
 6252 Proposed Standard. URL: <https://tools.ietf.org/html/rfc8152>  
 6253  
 6254 [RFC8230]  
 6255 M. Jones. Using RSA Algorithms with CBOR Object Signing and  
 6256 Encryption (COSE) Messages. September 2017. Proposed Standard.  
 6257 URL: <https://tools.ietf.org/html/rfc8230>  
 6258  
 6259 [SEC1]  
 6260 SEC1: Elliptic Curve Cryptography, Version 2.0. URL:  
 6261 <http://www.secg.org/sec1-v2.pdf>  
 6262  
 6263 [SECURE-CONTEXTS]  
 6264 Mike West. Secure Contexts. 15 September 2016. CR. URL:  
 6265 <https://www.w3.org/TR/secure-contexts/>  
 6266  
 6267 [TCG-CMCPProfile-AIKCertEnroll]  
 6268 Scott Kelly; et al. TCG Infrastructure Working Group: A CMC  
 6269 Profile for AIK Certificate Enrollment. 24 March 2011.  
 6270 Published. URL:  
 6271 [https://trustedcomputinggroup.org/wp-content/uploads/IWG\\_CMC\\_Pro  
 6272 file\\_Cert\\_Enrollment\\_v1\\_r7.pdf](https://trustedcomputinggroup.org/wp-content/uploads/IWG_CMC_Profile_Cert_Enrollment_v1_r7.pdf)  
 6273  
 6274 [TokenBinding]  
 6275 A. Popov; et al. The Token Binding Protocol Version 1.0.  
 6276 February 16, 2017. Internet-Draft. URL:  
 6277 <https://tools.ietf.org/html/draft-ietf-tokbind-protocol>  
 6278  
 6279 [URL]  
 6280 Anne van Kesteren. URL Standard. Living Standard. URL:  
 6281 <https://url.spec.whatwg.org/>  
 6282  
 6283 [WebAuthn-Registries]  
 6284 Jeff Hodges; Giridhar Mandyam; Michael B. Jones. Registries for  
 6285 Web Authentication (WebAuthn). March 2017. Active  
 6286 Internet-Draft. URL:  
 6287 [https://xml2rfc.tools.ietf.org/cgi-bin/xml2rfc.cgi?modeAsFormat=  
 6288 html&url=https://raw.githubusercontent.com/w3c/webauthn/ma  
 6289 ster/draft-hodges-webauthn-registries.xml](https://xml2rfc.tools.ietf.org/cgi-bin/xml2rfc.cgi?modeAsFormat=html&url=https://raw.githubusercontent.com/w3c/webauthn/master/draft-hodges-webauthn-registries.xml)  
 6290

6291 [WebIDL]  
 6292 Cameron McCormack; Boris Zbarsky; Tobie Langel. Web IDL. 15  
 6293 December 2016. ED. URL: <https://heycam.github.io/webidl/>  
 6294  
 6295 [WebIDL-1]  
 6296 Cameron McCormack. WebIDL Level 1. 15 December 2016. REC. URL:  
 6297 <https://www.w3.org/TR/2016/REC-WebIDL-1-20161215/>  
 6298  
 6299 Informative References  
 6300  
 6301 [Ceremony]  
 6302 Carl Ellison. Ceremony Design and Analysis. 2007. URL:  
 6303 <https://eprint.iacr.org/2007/399.pdf>  
 6304

6305 [EduPersonObjectClassSpec]  
 6306 EduPerson Object Class Specification (200604a). May 15, 2007.  
 6307 URL:  
 6308 [https://www.internet2.edu/media/medialibrary/2013/09/04/internet  
 6309 2-mace-dir-eduperson-200604.html](https://www.internet2.edu/media/medialibrary/2013/09/04/internet2-mace-dir-eduperson-200604.html)  
 6310

6311 [Feature-Policy]

5555 Feature Policy. Draft Community Group Report. URL:  
 5556 <https://wicg.github.io/feature-policy/>  
 5557

5558 **[FIDO-APPID]**  
 5559 D. Balfanz; et al. FIDO AppID and Facets. FIDO Alliance Review  
 5560 Draft. URL:  
 5561 <https://fidoalliance.org/specs/fido-uaf-v1.1-rd-20161005/fido-appid-and-facets-v1.1-rd-20161005.html>  
 5562

5563 **[FIDO-UAF-AUTHNR-CMDS]**  
 5564 R. Lindemann; J. Kemp. FIDO UAF Authenticator Commands. FIDO  
 5565 Alliance Implementation Draft. URL:  
 5566 <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-authnr-cmds-v1.1-id-20170202.html>  
 5567

5570 **[FIDOMetadataService]**  
 5571 R. Lindemann; B. Hill; D. Baghdasaryan. FIDO Metadata Service  
 5572 v1.0. FIDO Alliance Proposed Standard. URL:  
 5573 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-metadata-service-v1.0-ps-20141208.html>  
 5574

5575 **[FIDOSecRef]**  
 5576 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO Security Reference.  
 5577 FIDO Alliance Proposed Standard. URL:  
 5578 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-security-ref-v1.0-ps-20141208.html>  
 5579

5580 **[GeoJSON]**  
 5581 The GeoJSON Format Specification. URL:  
 5582 <http://geojson.org/geojson-spec.html>  
 5583

5584

5585 **[ISOBiometricVocabulary]**  
 5586 ISO/IEC JTC1/SC37. Information technology -- Vocabulary --  
 5587 Biometrics. 15 December 2012. International Standard: ISO/IEC  
 5588 2382-37:2012(E) First Edition. URL:  
 5589 [http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194\\_ISOIEC\\_2382-37\\_2012.zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194_ISOIEC_2382-37_2012.zip)  
 5590

5591

5592

5593 **[RFC4949]**  
 5594 R. Shirey. Internet Security Glossary, Version 2. August 2007.  
 5595 Informational. URL: <https://tools.ietf.org/html/rfc4949>  
 5596

5597 **[RFC5280]**  
 5598 D. Cooper; et al. Internet X.509 Public Key Infrastructure  
 5599 Certificate and Certificate Revocation List (CRL) Profile. May  
 5600 2008. Proposed Standard. URL:  
 5601 <https://tools.ietf.org/html/rfc5280>  
 5602

5603 **[RFC6265]**  
 5604 A. Barth. HTTP State Management Mechanism. April 2011. Proposed  
 5605 Standard. URL: <https://tools.ietf.org/html/rfc6265>  
 5606

5607 **[RFC6454]**  
 5608 A. Barth. The Web Origin Concept. December 2011. Proposed  
 5609 Standard. URL: <https://tools.ietf.org/html/rfc6454>  
 5610

5611

6312 Feature Policy. Draft Community Group Report. URL:  
 6313 <https://wicg.github.io/feature-policy/>  
 6314

6315 **[FIDO-UAF-AUTHNR-CMDS]**  
 6316 R. Lindemann; J. Kemp. FIDO UAF Authenticator Commands. FIDO  
 6317 Alliance Implementation Draft. URL:  
 6318 <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-authnr-cmds-v1.1-id-20170202.html>  
 6319

6320 **[FIDOAuthnrSecReqs]**  
 6321 D. Biggs; et al. FIDO Authenticator Security Requirements. FIDO  
 6322 Alliance Final Documents. URL:  
 6323 <https://fidoalliance.org/specs/fido-security-requirements-v1.0-fd-20170524/>  
 6324

6325 **[FIDOMetadataService]**  
 6326 R. Lindemann; B. Hill; D. Baghdasaryan. FIDO Metadata Service  
 6327 v1.0. FIDO Alliance Proposed Standard. URL:  
 6328 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-metadata-service-v1.0-ps-20141208.html>  
 6329

6330 **[FIDOSecRef]**  
 6331 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO Security Reference.  
 6332 FIDO Alliance Proposed Standard. URL:  
 6333 <https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-security-ref-v1.2-ps-20170411.html>  
 6334

6335 **[FIDOUAFAuthenticatorMetadataStatements]**  
 6336 B. Hill; D. Baghdasaryan; J. Kemp. FIDO UAF Authenticator  
 6337 Metadata Statements v1.0. FIDO Alliance Proposed Standard. URL:  
 6338 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-authnr-metadata-v1.0-ps-20141208.html>  
 6339

6340 **[ISOBiometricVocabulary]**  
 6341 ISO/IEC JTC1/SC37. Information technology -- Vocabulary --  
 6342 Biometrics. 15 December 2012. International Standard: ISO/IEC  
 6343 2382-37:2012(E) First Edition. URL:  
 6344 [http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194\\_ISOIEC\\_2382-37\\_2012.zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194_ISOIEC_2382-37_2012.zip)  
 6345

6346

6347 **[RFC3279]**  
 6348 L. Bassham; W. Polk; R. Housley. Algorithms and Identifiers for  
 6349 the Internet X.509 Public Key Infrastructure Certificate and  
 6350 Certificate Revocation List (CRL) Profile. April 2002. Proposed  
 6351 Standard. URL: <https://tools.ietf.org/html/rfc3279>  
 6352

6353 **[RFC4949]**  
 6354 R. Shirey. Internet Security Glossary, Version 2. August 2007.  
 6355 Informational. URL: <https://tools.ietf.org/html/rfc4949>  
 6356

6357 **[RFC5280]**  
 6358 D. Cooper; et al. Internet X.509 Public Key Infrastructure  
 6359 Certificate and Certificate Revocation List (CRL) Profile. May  
 6360 2008. Proposed Standard. URL:  
 6361 <https://tools.ietf.org/html/rfc5280>  
 6362

6363 **[RFC6265]**  
 6364 A. Barth. HTTP State Management Mechanism. April 2011. Proposed  
 6365 Standard. URL: <https://tools.ietf.org/html/rfc6265>  
 6366

6367 **[RFC6454]**  
 6368 A. Barth. The Web Origin Concept. December 2011. Proposed  
 6369 Standard. URL: <https://tools.ietf.org/html/rfc6454>  
 6370

6371

6372

6373

6374

6375

5611 [RFC7515]  
5612 M. Jones; J. Bradley; N. Sakimura. JSON Web Signature (JWS). May  
5613 2015. Proposed Standard. URL:  
5614 https://tools.ietf.org/html/rfc7515  
5615  
5616 [RFC8017]  
5617 K. Moriarty, Ed.; et al. PKCS #1: RSA Cryptography  
5618 Specifications Version 2.2. November 2016. Informational. URL:  
5619 https://tools.ietf.org/html/rfc8017  
5620  
5621 [TPMv2-EK-Profile]  
5622 TCG EK Credential Profile for TPM Family 2.0. URL:  
5623 http://www.trustedcomputinggroup.org/wp-content/uploads/Credenti  
5624 al\_Profile\_EK\_V2.0\_R14\_published.pdf  
5625  
5626 [TPMv2-Part1]  
5627 Trusted Platform Module Library, Part 1: Architecture. URL:  
5628 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-  
5629 2.0-Part-1-Architecture-01.38.pdf  
5630  
5631 [TPMv2-Part2]  
5632 Trusted Platform Module Library, Part 2: Structures. URL:  
5633 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-  
5634 2.0-Part-2-Structures-01.38.pdf  
5635  
5636 [TPMv2-Part3]  
5637 Trusted Platform Module Library, Part 3: Commands. URL:  
5638 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-  
5639 2.0-Part-3-Commands-01.38.pdf  
5640  
5641 [UAFProtocol]  
5642 R. Lindemann; et al. FIDO UAF Protocol Specification v1.0. FIDO  
5643 Alliance Proposed Standard. URL:  
5644 https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua  
5645 f-protocol-v1.0-ps-20141208.html  
5646  
5647 IDL Index  
5648  
5649 [SecureContext, Exposed=Window]  
5650 interface PublicKeyCredential : Credential {  
5651 [SameObject] readonly attribute ArrayBuffer rawId;  
5652 [SameObject] readonly attribute AuthenticatorResponse response;  
5653 AuthenticationExtensions getClientExtensionResults();  
5654 };  
5655  
5656 partial dictionary CredentialCreationOptions {  
5657 MakePublicKeyCredentialOptions publicKey;  
5658 };  
5659  
5660 partial dictionary CredentialRequestOptions {  
5661 PublicKeyCredentialRequestOptions publicKey;  
5662 };  
5663  
5664 partial interface PublicKeyCredential {  
5665 static Promise < boolean > isUserVerifyingPlatformAuthenticatorAvailable();  
5666 };  
5667  
5668 [SecureContext, Exposed=Window]  
5669 interface AuthenticatorResponse {  
5670 [SameObject] readonly attribute ArrayBuffer clientDataJSON;  
5671 };  
5672  
5673 [SecureContext, Exposed=Window]  
5674 interface AuthenticatorAttestationResponse : AuthenticatorResponse {  
5675 [SameObject] readonly attribute ArrayBuffer attestationObject;  
5676 };  
5677  
5678 [SecureContext, Exposed=Window]  
5679 interface AuthenticatorAssertionResponse : AuthenticatorResponse {  
5680 [SameObject] readonly attribute ArrayBuffer authenticatorData;

6376 [RFC7515]  
6377 M. Jones; J. Bradley; N. Sakimura. JSON Web Signature (JWS). May  
6378 2015. Proposed Standard. URL:  
6379 https://tools.ietf.org/html/rfc7515  
6380  
6381 [RFC8017]  
6382 K. Moriarty, Ed.; et al. PKCS #1: RSA Cryptography  
6383 Specifications Version 2.2. November 2016. Informational. URL:  
6384 https://tools.ietf.org/html/rfc8017  
6385  
6386 [TPMv2-EK-Profile]  
6387 TCG EK Credential Profile for TPM Family 2.0. URL:  
6388 http://www.trustedcomputinggroup.org/wp-content/uploads/Credenti  
6389 al\_Profile\_EK\_V2.0\_R14\_published.pdf  
6390  
6391 [TPMv2-Part1]  
6392 Trusted Platform Module Library, Part 1: Architecture. URL:  
6393 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-  
6394 2.0-Part-1-Architecture-01.38.pdf  
6395  
6396 [TPMv2-Part2]  
6397 Trusted Platform Module Library, Part 2: Structures. URL:  
6398 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-  
6399 2.0-Part-2-Structures-01.38.pdf  
6400  
6401 [TPMv2-Part3]  
6402 Trusted Platform Module Library, Part 3: Commands. URL:  
6403 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-  
6404 2.0-Part-3-Commands-01.38.pdf  
6405  
6406 [UAFProtocol]  
6407 R. Lindemann; et al. FIDO UAF Protocol Specification v1.0. FIDO  
6408 Alliance Proposed Standard. URL:  
6409 https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua  
6410 f-protocol-v1.0-ps-20141208.html  
6411  
6412 IDL Index  
6413  
6414 [SecureContext, Exposed=Window]  
6415 interface PublicKeyCredential : Credential {  
6416 [SameObject] readonly attribute ArrayBuffer rawId;  
6417 [SameObject] readonly attribute AuthenticatorResponse response;  
6418 AuthenticationExtensionsClientOutputs getClientExtensionResults();  
6419 };  
6420  
6421 partial dictionary CredentialCreationOptions {  
6422 PublicKeyCredentialCreationOptions publicKey;  
6423 };  
6424  
6425 partial dictionary CredentialRequestOptions {  
6426 PublicKeyCredentialRequestOptions publicKey;  
6427 };  
6428  
6429 partial interface PublicKeyCredential {  
6430 static Promise < boolean > isUserVerifyingPlatformAuthenticatorAvailable();  
6431 };  
6432  
6433 [SecureContext, Exposed=Window]  
6434 interface AuthenticatorResponse {  
6435 [SameObject] readonly attribute ArrayBuffer clientDataJSON;  
6436 };  
6437  
6438 [SecureContext, Exposed=Window]  
6439 interface AuthenticatorAttestationResponse : AuthenticatorResponse {  
6440 [SameObject] readonly attribute ArrayBuffer attestationObject;  
6441 };  
6442  
6443 [SecureContext, Exposed=Window]  
6444 interface AuthenticatorAssertionResponse : AuthenticatorResponse {  
6445 [SameObject] readonly attribute ArrayBuffer authenticatorData;

```

5681 [SameObject] readonly attribute ArrayBuffer signature;
5682 [SameObject] readonly attribute ArrayBuffer userHandle;
5683 };
5684
5685 dictionary PublicKeyCredentialParameters {
5686   required PublicKeyCredentialType type;
5687   required COSEAlgorithmIdentifier alg;
5688 };
5689
5690 dictionary MakePublicKeyCredentialOptions {
5691   required PublicKeyCredentialRpEntity rp;
5692   required PublicKeyCredentialUserEntity user;
5693
5694   required BufferSource challenge;
5695   required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
5696
5697   unsigned long timeout;
5698   sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
5699   AuthenticatorSelectionCriteria authenticatorSelection;
5700   AttestationConveyancePreference attestation = "none";
5701   AuthenticationExtensions extensions;
5702 };
5703
5704 dictionary PublicKeyCredentialEntity {
5705   required DOMString name;
5706   USVString icon;
5707 };
5708
5709 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
5710   DOMString id;
5711 };
5712
5713 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
5714   required BufferSource id;
5715   required DOMString displayName;
5716 };
5717
5718 dictionary AuthenticatorSelectionCriteria {
5719   AuthenticatorAttachment authenticatorAttachment;
5720   boolean requireResidentKey = false;
5721   UserVerificationRequirement userVerification = "preferred";
5722 };
5723
5724 enum AuthenticatorAttachment {
5725   "platform" // Platform attachment
5726   "cross-platform" // Cross-platform attachment
5727 };
5728
5729 enum AttestationConveyancePreference {
5730   "none",
5731   "indirect",
5732   "direct"
5733 };
5734
5735 dictionary PublicKeyCredentialRequestOptions {
5736   required BufferSource challenge;
5737   unsigned long timeout;
5738   USVString rpId;
5739   sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
5740   UserVerificationRequirement userVerification = "preferred";
5741   AuthenticationExtensions extensions;
5742 };
5743
5744 typedef record<DOMString, any> AuthenticationExtensions;

```

```

6446 [SameObject] readonly attribute ArrayBuffer signature;
6447 [SameObject] readonly attribute ArrayBuffer? userHandle;
6448 };
6449
6450 dictionary PublicKeyCredentialParameters {
6451   required PublicKeyCredentialType type;
6452   required COSEAlgorithmIdentifier alg;
6453 };
6454
6455 dictionary PublicKeyCredentialCreationOptions {
6456   required PublicKeyCredentialRpEntity rp;
6457   required PublicKeyCredentialUserEntity user;
6458
6459   required BufferSource challenge;
6460   required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
6461
6462   unsigned long timeout;
6463   sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
6464   AuthenticatorSelectionCriteria authenticatorSelection;
6465   AttestationConveyancePreference attestation = "none";
6466   AuthenticationExtensionsClientInputs extensions;
6467 };
6468
6469 dictionary PublicKeyCredentialEntity {
6470   required DOMString name;
6471   USVString icon;
6472 };
6473
6474 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
6475   DOMString id;
6476 };
6477
6478 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
6479   required BufferSource id;
6480   required DOMString displayName;
6481 };
6482
6483 dictionary AuthenticatorSelectionCriteria {
6484   AuthenticatorAttachment authenticatorAttachment;
6485   boolean requireResidentKey = false;
6486   UserVerificationRequirement userVerification = "preferred";
6487 };
6488
6489 enum AuthenticatorAttachment {
6490   "platform" // Platform attachment
6491   "cross-platform" // Cross-platform attachment
6492 };
6493
6494 enum AttestationConveyancePreference {
6495   "none",
6496   "indirect",
6497   "direct"
6498 };
6499
6500 dictionary PublicKeyCredentialRequestOptions {
6501   required BufferSource challenge;
6502   unsigned long timeout;
6503   USVString rpId;
6504   sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
6505   UserVerificationRequirement userVerification = "preferred";
6506   AuthenticationExtensionsClientInputs extensions;
6507 };
6508
6509 dictionary AuthenticationExtensionsClientInputs {
6510 };
6511
6512 dictionary AuthenticationExtensionsClientOutputs {
6513 };
6514
6515 typedef record<DOMString, DOMString> AuthenticationExtensionsAuthenticatorInputs

```

```

5745 dictionary CollectedClientData {
5746     required DOMString type;
5747     required DOMString challenge;
5748     required DOMString origin;
5749     required DOMString hashAlgorithm;
5750     DOMString tokenBindingId;
5751     AuthenticationExtensions clientExtensions;
5752     AuthenticationExtensions authenticatorExtensions;
5753 };
5754 };
5755
5756 enum PublicKeyCredentialType {
5757     "public-key"
5758 };
5759
5760 dictionary PublicKeyCredentialDescriptor {
5761     required PublicKeyCredentialType type;
5762     required BufferSource id;
5763     sequence<AuthenticatorTransport> transports;
5764 };
5765
5766 enum AuthenticatorTransport {
5767     "usb",
5768     "nfc",
5769     "ble"
5770 };
5771
5772 typedef long COSEAlgorithmIdentifier;
5773
5774 enum UserVerificationRequirement {
5775     "required",
5776     "preferred",
5777     "discouraged"
5778 };
5779

```

```

6516 ;
6517
6518 dictionary CollectedClientData {
6519     required DOMString type;
6520     required DOMString challenge;
6521     required DOMString origin;
6522     TokenBinding tokenBinding;
6523 };
6524
6525 dictionary TokenBinding {
6526     required TokenBindingStatus status;
6527     DOMString id;
6528 };
6529
6530 enum TokenBindingStatus { "present", "supported", "not-supported" };
6531
6532 enum PublicKeyCredentialType {
6533     "public-key"
6534 };
6535
6536 dictionary PublicKeyCredentialDescriptor {
6537     required PublicKeyCredentialType type;
6538     required BufferSource id;
6539     sequence<AuthenticatorTransport> transports;
6540 };
6541
6542 enum AuthenticatorTransport {
6543     "usb",
6544     "nfc",
6545     "ble"
6546 };
6547
6548 typedef long COSEAlgorithmIdentifier;
6549
6550 enum UserVerificationRequirement {
6551     "required",
6552     "preferred",
6553     "discouraged"
6554 };
6555
6556 partial dictionary AuthenticationExtensionsClientInputs {
6557     USVString appId;
6558 };
6559
6560 partial dictionary AuthenticationExtensionsClientOutputs {
6561     boolean appId;
6562 };
6563
6564 partial dictionary AuthenticationExtensionsClientInputs {
6565     USVString txAuthSimple;
6566 };
6567
6568 partial dictionary AuthenticationExtensionsClientOutputs {
6569     USVString txAuthSimple;
6570 };
6571
6572 dictionary txAuthGenericArg {
6573     required USVString contentType; // MIME-Type of the content, e.g., "image
6574     /png"
6575     required ArrayBuffer content;
6576 };
6577
6578 partial dictionary AuthenticationExtensionsClientInputs {
6579     txAuthGenericArg txAuthGeneric;
6580 };
6581
6582 partial dictionary AuthenticationExtensionsClientOutputs {
6583     ArrayBuffer txAuthGeneric;
6584 };
6585

```

5780 typedef sequence<AAGUID> AuthenticatorSelectionList;

5781

5782 typedef BufferSource AAGUID;

5783

5784 Issues Index

5785 The definitions of "lifetime of" and "becomes available" are intended  
5786 to represent how devices are hotplugged into (USB) or discovered by  
5787 (NFC) browsers, and are under-specified. Resolving this with good  
5788 definitions or some other means will be addressed by resolving Issue  
5790 #613. RET

5792 need to define "blinding". See also #462.  
5793 <<https://github.com/w3c/webauthn/issues/694>> RET

5794 @balfanz wishes to add to the "direct" case: If the authenticator  
5795 violates the privacy requirements of the attestation type it is using,  
5796 the client SHOULD terminate this algorithm with a  
5797 "AttestationNotPrivateError". RET

5798 The definitions of "lifetime of" and "becomes available" are intended  
5799 to represent how devices are hotplugged into (USB) or discovered by

6586 typedef sequence<AAGUID> AuthenticatorSelectionList;

6587

6588 partial dictionary AuthenticationExtensionsClientInputs {  
6589 AuthenticatorSelectionList authnSel;  
6590 };

6591

6592 typedef BufferSource AAGUID;

6593

6594 partial dictionary AuthenticationExtensionsClientOutputs {  
6595 boolean authnSel;  
6596 };

6597

6598 partial dictionary AuthenticationExtensionsClientInputs {  
6599 boolean exts;  
6600 };

6601

6602 typedef sequence<USVString> AuthenticationExtensionsSupported;

6603

6604 partial dictionary AuthenticationExtensionsClientOutputs {  
6605 AuthenticationExtensionsSupported exts;  
6606 };

6607

6608 partial dictionary AuthenticationExtensionsClientInputs {  
6609 boolean uvi;  
6610 };

6611

6612 partial dictionary AuthenticationExtensionsClientOutputs {  
6613 ArrayBuffer uvi;  
6614 };

6615

6616 partial dictionary AuthenticationExtensionsClientInputs {  
6617 boolean loc;  
6618 };

6619

6620 partial dictionary AuthenticationExtensionsClientOutputs {  
6621 Coordinates loc;  
6622 };

6623

6624 partial dictionary AuthenticationExtensionsClientInputs {  
6625 boolean uvm;  
6626 };

6627

6628 typedef sequence<unsigned long> UvmEntry;  
6629 typedef sequence<UvmEntry> UvmEntries;

6630

6631 partial dictionary AuthenticationExtensionsClientOutputs {  
6632 UvmEntries uvm;  
6633 };

6634

6635 dictionary authenticatorBiometricPerfBounds{  
6636 float FAR;  
6637 float FRR;  
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

6586 Issues Index

6587 The definitions of "lifetime of" and "becomes available" are intended  
6588 to represent how devices are hot-plugged into (USB) or discovered by  
6589 (NFC) browsers, and are underspecified. Resolving this with good  
6590 definitions or some other means will be addressed by resolving Issue  
6592 #613. RET

6594 @balfanz wishes to add to the "direct" case: If the authenticator  
6595 violates the privacy requirements of the attestation type it is using,  
6596 the client SHOULD terminate this algorithm with an  
6597 "AttestationNotPrivateError". RET

6598 The definitions of "lifetime of" and "becomes available" are intended  
6599 to represent how devices are hot-plugged into (USB) or discovered by

5800 (NFC) browsers, and are under-specified. Resolving this with good  
5801 definitions or some other means will be addressed by resolving Issue  
5802 #613. RET  
5803 The foregoing step `_may_` be incorrect, in that we are attempting to  
5804 create `savedCredentialId` here and use it later below, and we do not  
5805 have a global in which to allocate a place for it. Perhaps this is good  
5806 enough? addendum: @jcyjones feels the above step is likely good enough.  
5807 RET  
5808 The WHATWG HTML WG is discussing whether to provide a hook when a  
5809 browsing context gains or loses focuses. If a hook is provided, the  
5810 above paragraph will be updated to include the hook. See WHATWG HTML WG  
5811 Issue #2711 for more details. RET  
5812  
5813 #base64url-encodingReferenced in:  
5814 \* 5.1. PublicKeyCredential Interface  
5815 \* 5.1.3. Create a new credential - PublicKeyCredential's  
5816 `[[Create]](origin, options, sameOriginWithAncestors) method (2)`  
5817 \* 5.1.4.1. PublicKeyCredential's  
5818 `[[DiscoverFromExternalSource]](origin, options,`  
5819 `sameOriginWithAncestors) method (2)`  
5820 \* 7.2. Verifying an authentication assertion

5821 #cborReferenced in:  
5822

5823 \* 5.1.3. Create a new credential - PublicKeyCredential's  
5824 `[[Create]](origin, options, sameOriginWithAncestors) method`  
5825 \* 5.1.4.1. PublicKeyCredential's  
5826 `[[DiscoverFromExternalSource]](origin, options,`  
5827 `sameOriginWithAncestors) method`  
5828 \* 6.1. Authenticator data (2)  
5829 \* 9. WebAuthn Extensions (2) (3)

5830 \* 9.2. Defining extensions (2)  
5831 \* 9.3. Extending request parameters  
5832 \* 9.4. Client extension processing (2)  
5833 \* 9.5. Authenticator extension processing (2) (3) (4) (5)

5834 #attestationReferenced in:

5835 \* 4. Terminology (2)  
5836 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
5837 `AttestationConveyancePreference)` (2)  
5838 \* 6. WebAuthn Authenticator `model` (2)  
5839 \* 6.3. Attestation (2) (3) (4)

5840 \* 11.1. WebAuthn Attestation Statement Format Identifier  
5841 Registrations  
5842

5843 #attestation-certificateReferenced in:

5844 \* 4. Terminology (2)  
5845 \* 5.1.3. Create a new credential - PublicKeyCredential's  
5846 `[[Create]](origin, options, sameOriginWithAncestors) method`  
5847 \* 8.3.1. TPM attestation statement certificate requirements

5848 #attestation-key-pairReferenced in:

5849 \* 4. Terminology (2)  
5850 \* 6.3. Attestation  
5851  
5852

5853 #attestation-private-keyReferenced in:  
5854

6654 (NFC) browsers, and are underspecified. Resolving this with good  
6655 definitions or some other means will be addressed by resolving Issue  
6656 #613. RET  
6657 The foregoing step `_may_` be incorrect, in that we are attempting to  
6658 create `savedCredentialId` here and use it later below, and we do not  
6659 have a global in which to allocate a place for it. Perhaps this is good  
6660 enough? addendum: @jcyjones feels the above step is likely good enough.  
6661 RET  
6662 The WHATWG HTML WG is discussing whether to provide a hook when a  
6663 browsing context gains or loses focuses. If a hook is provided, the  
6664 above paragraph will be updated to include the hook. See WHATWG HTML WG  
6665 Issue #2711 for more details. RET  
6666  
6667 #base64url-encodingReferenced in:  
6668 \* 5.1. PublicKeyCredential Interface  
6669 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6670 `[[Create]](origin, options, sameOriginWithAncestors) method (2)`  
6671 \* 5.1.4.1. PublicKeyCredential's  
6672 `[[DiscoverFromExternalSource]](origin, options,`  
6673 `sameOriginWithAncestors) method (2)`  
6674 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
6675 `CollectedClientData)`  
6676 \* 7.1. Registering a new credential  
6677 \* 7.2. Verifying an authentication assertion (2)

6678 #cborReferenced in:

6679 \* 2.4. All Conformance Classes  
6680 \* 3. Dependencies  
6681 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6682 `[[Create]](origin, options, sameOriginWithAncestors) method (2)`  
6683 \* 5.1.4.1. PublicKeyCredential's  
6684 `[[DiscoverFromExternalSource]](origin, options,`  
6685 `sameOriginWithAncestors) method`  
6686 \* 6.1. Authenticator data (2)  
6687 \* 6.2.2. The authenticator `MakeCredential` operation  
6688 \* 6.2.3. The authenticator `GetAssertion` operation  
6689 \* 9. WebAuthn Extensions (2) (3) (4) (5) (6) (7)  
6690 \* 9.2. Defining extensions (2)  
6691 \* 9.3. Extending request parameters  
6692 \* 9.4. Client extension processing (2)  
6693 \* 9.5. Authenticator extension processing (2)

6694 #assertionReferenced in:

6695 \* 7.1. Registering a new credential  
6696 \* 10.1. FIDO AppID Extension (`appid`)  
6697 \* 13.3. `credentialId` Unsigned

6700 #attestationReferenced in:

6701 \* 4. Terminology (2)  
6702 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
6703 `AttestationConveyancePreference)` (2)  
6704 \* 6. WebAuthn Authenticator `Model` (2)  
6705 \* 6.3. Attestation (2) (3) (4)  
6706 \* 8.2. Packed Attestation Statement Format  
6707 \* 11.1. WebAuthn Attestation Statement Format Identifier  
6708 Registrations  
6709 \* 13. Security Considerations

6710 #attestation-certificateReferenced in:

6711 \* 4. Terminology (2)  
6712 \* 6.3.3. Attestation Types  
6713  
6714 \* 8.3.1. TPM attestation statement certificate requirements

6715 #attestation-key-pairReferenced in:

6716 \* 4. Terminology (2)  
6717 \* 6.3. Attestation  
6718 \* 6.3.3. Attestation Types

6719 #attestation-private-keyReferenced in:  
6720  
6721  
6722

5855 \* 6. WebAuthn Authenticator **model**  
5856 \* 6.3. Attestation

5857  
5858 #attestation-public-keyReferenced in:  
5859 \* 6.3. Attestation

5860  
5861 #authenticationReferenced in:  
5862 \* 1. Introduction (2)  
5863 \* 4. Terminology (2) (3) (4) (5) (6) (7)  
5864 \* 7.2. Verifying an authentication assertion (2) (3)

5865  
5866 #authentication-assertionReferenced in:  
5867 \* 1. Introduction  
5868 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8)  
5869 \* 5.1. PublicKeyCredential Interface  
5870 \* 5.2.2. Web Authentication Assertion (interface  
5871 AuthenticatorAssertionResponse)  
5872 \* 5.5. Options for Assertion Generation (dictionary  
5873 PublicKeyCredentialRequestOptions)  
5874 \* 9. WebAuthn Extensions

5875  
5876 #authenticatorReferenced in:  
5877 \* 1. Introduction (2) (3) (4)  
5878 \* 1.1. Use Cases  
5879 \* 2.2. Authenticators

5880 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)  
5881 (14) (15) (16) (17)  
5882 \* 5. Web Authentication API (2) (3)  
5883 \* 5.1. PublicKeyCredential Interface  
5884 \* 5.1.3. Create a new credential - PublicKeyCredential's  
5885 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)

5886 \* 5.1.4.1. PublicKeyCredential's  
5887 [[DiscoverFromExternalSource]](origin, options,  
5888 sameOriginWithAncestors) method (2) (3) (4) (5)  
5889 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
5890 \* 5.2.1. Information about Public Key Credential (interface  
5891 AuthenticatorAttestationResponse) (2)  
5892 \* 5.2.2. Web Authentication Assertion (interface  
5893 AuthenticatorAssertionResponse)  
5894 \* 5.4.1. Public Key Entity Description (dictionary  
5895 PublicKeyCredentialEntity) (2)  
5896 \* 5.4.3. User Account Parameters for Credential Generation  
5897 (dictionary PublicKeyCredentialUserEntity)  
5898 \* 5.4.5. Authenticator Attachment enumeration (enum  
5899 AuthenticatorAttachment)  
5900 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
5901 AttestationConveyancePreference) (2)  
5902 \* 5.5. Options for Assertion Generation (dictionary  
5903 PublicKeyCredentialRequestOptions)  
5904 \* 6. WebAuthn Authenticator **model** (2) (3) (4) (5) (6)  
5905 \* 6.1. Authenticator data  
5906 \* 6.2.1. **The authenticatorMakeCredential operation (2) (3)**  
5907 \* 6.2.2. The authenticator**GetAssertion** operation (2) (3)

5908 \* 6.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)  
5909 \* 6.3.2. Attestation Statement Formats

5910 \* 6.3.4. Generating an Attestation Object  
5911 \* **6.3.5.1. Privacy**  
5912 \* **6.3.5.2. Attestation Certificate and Attestation Certificate CA**  
5913 **Compromise**  
5914 \* **7.1. Registering a new credential**  
5915 \* 8.2. Packed Attestation Statement Format  
5916 \* 8.4. Android Key Attestation Statement Format

6723 \* 6. WebAuthn Authenticator **Model**  
6724 \* 6.3. Attestation  
6725 \* **8.2. Packed Attestation Statement Format**

6726  
6727 #attestation-public-keyReferenced in:  
6728 \* 6.3. Attestation  
6729 \* **8.2. Packed Attestation Statement Format**

6730  
6731 #authenticationReferenced in:  
6732 \* 1. Introduction (2)  
6733 \* 4. Terminology (2) (3) (4) (5) (6) (7)  
6734 \* 7.2. Verifying an authentication assertion (2) (3) (4)  
6735 \* **13. Security Considerations**  
6736 \* **14.3. Authentication Ceremony Privacy**

6737  
6738 #authentication-assertionReferenced in:  
6739 \* 1. Introduction  
6740 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8)  
6741 \* 5.1. PublicKeyCredential Interface  
6742 \* 5.2.2. Web Authentication Assertion (interface  
6743 AuthenticatorAssertionResponse)  
6744 \* 5.5. Options for Assertion Generation (dictionary  
6745 PublicKeyCredentialRequestOptions)  
6746 \* 9. WebAuthn Extensions

6747  
6748 #authenticatorReferenced in:  
6749 \* 1. Introduction (2) (3) (4)  
6750 \* 1.1. Use Cases  
6751 \* 2.2. Authenticators  
6752 \* **2.2.1. Backwards Compatibility with FIDO U2F**  
6753 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)  
6754 (14) (15) (16) (17) (18) (19)  
6755 \* 5. Web Authentication API (2) (3)  
6756 \* 5.1. PublicKeyCredential Interface  
6757 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6758 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6759 (4)  
6760 \* 5.1.4.1. PublicKeyCredential's  
6761 [[DiscoverFromExternalSource]](origin, options,  
6762 sameOriginWithAncestors) method (2) (3) (4) (5) (6)  
6763 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
6764 \* 5.2.1. Information about Public Key Credential (interface  
6765 AuthenticatorAttestationResponse) (2)  
6766 \* 5.2.2. Web Authentication Assertion (interface  
6767 AuthenticatorAssertionResponse)  
6768 \* 5.4.1. Public Key Entity Description (dictionary  
6769 PublicKeyCredentialEntity) (2)  
6770 \* 5.4.3. User Account Parameters for Credential Generation  
6771 (dictionary PublicKeyCredentialUserEntity)  
6772 \* 5.4.5. Authenticator Attachment enumeration (enum  
6773 AuthenticatorAttachment)  
6774 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
6775 AttestationConveyancePreference) (2)  
6776 \* 5.5. Options for Assertion Generation (dictionary  
6777 PublicKeyCredentialRequestOptions)  
6778 \* 6. WebAuthn Authenticator **Model** (2) (3) (4) (5) (6)  
6779 \* 6.1. Authenticator data  
6780 \* 6.2.1. **Lookup Credential Source by Credential ID algorithm**  
6781 \* 6.2.2. The authenticator**MakeCredential** operation (2)  
6782 \* **6.2.3. The authenticatorGetAssertion operation (2) (3) (4)**  
6783 \* 6.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)  
6784 \* 6.3.2. Attestation Statement Formats  
6785 \* **6.3.3. Attestation Types (2) (3) (4)**  
6786 \* 6.3.4. Generating an Attestation Object  
6787 \* **7.1. Registering a new credential (2)**  
6788 \* **7.2. Verifying an authentication assertion**

6789 \* 8.2. Packed Attestation Statement Format  
6790 \* 8.4. Android Key Attestation Statement Format

5917 \* 8.5. Android SafetyNet Attestation Statement Format

5918 \* 10.5. Supported Extensions Extension (exts)

5919 \* 10.6. User Verification Index Extension (uvi)

5920 \* 10.7. Location Extension (loc) (2) (3) (4)

5921 \* 10.8. User Verification Method Extension (uvm)

5922 \* 12. Sample scenarios

5923

5924 #authorization-gestureReferenced in:

5925 \* 1.1.1. Registration

5926 \* 1.1.2. Authentication

5927 \* 1.1.3. Other use cases and configurations

5928 \* 4. Terminology (2) (3) (4) (5) (6)

5929 \* 5.1.4. Use an existing credential to make an assertion -

5930 PublicKeyCredential's [[Get]](options) method (2)

5931

5932 #biometric-recognitionReferenced in:

5933 \* 4. Terminology (2)

5934

5935 #ceremonyReferenced in:

5936 \* 1. Introduction

5937 \* 4. Terminology (2) (3) (4) (5) (6) (7)

5938 \* 7.1. Registering a new credential

5939 \* 7.2. Verifying an authentication assertion

5940

5941 #clientReferenced in:

5942 \* 4. Terminology

5943 \* 5.1.6. Availability of User-Verifying Platform Authenticator -

5944 PublicKeyCredential's

5945 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)

5946

5947 #client-side-resident-credential-private-keyReferenced in:

5948 \* 4. Terminology (2)

5949 \* 5.1.3. Create a new credential - PublicKeyCredential's

5950 [[Create]](origin, options, sameOriginWithAncestors) method

5951 \* 5.4.4. Authenticator Selection Criteria (dictionary

5952 AuthenticatorSelectionCriteria) (2)

5953 \* 6.2.1. The authenticatorMakeCredential operation

5954

5955 #conforming-user-agentReferenced in:

5956 \* 1. Introduction

5957 \* 2.1. User Agents

5958 \* 2.2. Authenticators

5959 \* 4. Terminology (2)

5960

5961 #credential-idReferenced in:

5962 \* 4. Terminology (2) (3) (4)

5963

5964 \* 5.1.4.1. PublicKeyCredential's

[[DiscoverFromExternalSource]](origin, options,

6791 \* 8.5. Android SafetyNet Attestation Statement Format

6792 \* 8.7. None Attestation Statement Format

6793 \* 10.5. Supported Extensions Extension (exts)

6794 \* 10.6. User Verification Index Extension (uvi)

6795

6796 \* 10.8. User Verification Method Extension (uvm)

6797 \* 12. Sample scenarios

6798 \* 13. Security Considerations (2) (3) (4) (5)

6799 \* 13.2.2. Attestation Certificate and Attestation Certificate CA

6800 Compromise

6801 \* 13.3. credentialId Unsigned

6802 \* 14.1. Attestation Privacy (2) (3)

6803 \* 14.2. Registration Ceremony Privacy (2) (3) (4) (5) (6)

6804

6805 #authorization-gestureReferenced in:

6806 \* 1.1.1. Registration

6807 \* 1.1.2. Authentication

6808 \* 1.1.3. Other use cases and configurations

6809 \* 4. Terminology (2) (3) (4) (5) (6)

6810 \* 5.1.4. Use an existing credential to make an assertion -

6811 PublicKeyCredential's [[Get]](options) method (2)

6812 \* 5.1.6. Preventing silent access to an existing credential -

6813 PublicKeyCredential's [[preventSilentAccess]](credential,

6814 sameOriginWithAncestors) method

6815

6816 #biometric-recognitionReferenced in:

6817 \* 4. Terminology (2) (3)

6818

6819 #biometric-authenticatorReferenced in:

6820 \* 10.9. Biometric Authenticator Performance Bounds Extension

6821 (biometricPerfBounds)

6822

6823 #ceremonyReferenced in:

6824 \* 1. Introduction

6825 \* 4. Terminology (2) (3) (4) (5) (6) (7)

6826 \* 7.1. Registering a new credential (2)

6827 \* 7.2. Verifying an authentication assertion (2)

6828 \* 13. Security Considerations

6829 \* 14.2. Registration Ceremony Privacy

6830 \* 14.3. Authentication Ceremony Privacy (2)

6831

6832 #clientReferenced in:

6833 \* 4. Terminology

6834 \* 5.1.7. Availability of User-Verifying Platform Authenticator -

6835 PublicKeyCredential's

6836 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)

6837 \* 5.4.5. Authenticator Attachment enumeration (enum

6838 AuthenticatorAttachment) (2) (3)

6839 \* 7.1. Registering a new credential

6840 \* 7.2. Verifying an authentication assertion

6841

6842 #client-side-resident-credential-private-keyReferenced in:

6843 \* 4. Terminology (2)

6844 \* 5.1.3. Create a new credential - PublicKeyCredential's

6845 [[Create]](origin, options, sameOriginWithAncestors) method

6846 \* 5.4.4. Authenticator Selection Criteria (dictionary

6847 AuthenticatorSelectionCriteria) (2)

6848 \* 6.2.2. The authenticatorMakeCredential operation (2)

6849

6850 #conforming-user-agentReferenced in:

6851 \* 1. Introduction

6852 \* 2.1. User Agents

6853 \* 2.2. Authenticators

6854 \* 4. Terminology (2)

6855

6856 #credential-idReferenced in:

6857 \* 4. Terminology (2) (3) (4)

6858 \* 5.1. PublicKeyCredential Interface (2)

6859 \* 5.1.4.1. PublicKeyCredential's

[[DiscoverFromExternalSource]](origin, options,

5965 sameOriginWithAncestors) method  
5966 \* 5.2.1. Information about Public Key Credential (interface  
5967 AuthenticatorAttestationResponse)  
5968 \* 6.2.2. The authenticatorGetAssertion operation (2)

5969 \* 6.3.1. Attested credential data  
5970 \* 7.1. Registering a new credential  
5971 \* 8.6. FIDO U2F Attestation Statement Format  
5972 \* 12.1. Registration  
5973 \* 12.3. Authentication (2) (3)

5974  
5975 #credential-public-keyReferenced in:  
5976 \* 4. Terminology (2) (3) (4) (5) (6) (7)  
5977 \* 5.2.1. Information about Public Key Credential (interface  
5978 AuthenticatorAttestationResponse)  
5979 \* 6. WebAuthn Authenticator model  
5980 \* 6.3. Attestation (2) (3)  
5981 \* 6.3.1. Attested credential data (2)  
5982 \* 12.1. Registration (2)

5983  
5984 #credential-key-pairReferenced in:  
5985 \* 4. Terminology (2) (3)

5986  
5987 #credential-private-keyReferenced in:  
5988 \* 4. Terminology (2) (3) (4) (5) (6)  
5989 \* 5.1. PublicKeyCredential Interface  
5990 \* 5.2.2. Web Authentication Assertion (interface  
5991 AuthenticatorAssertionResponse)  
5992 \* 6. WebAuthn Authenticator model  
5993 \* 6.2.2. The authenticatorGetAssertion operation  
5994 \* 6.3. Attestation (2)  
5995 \* 7.2. Verifying an authentication assertion  
5996

5997  
5998 #public-key-credential-sourceReferenced in:  
5999 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8)  
6000 \* 5.1.3. Create a new credential - PublicKeyCredential's  
[[Create]](origin, options, sameOriginWithAncestors) method

6860 sameOriginWithAncestors) method  
6861 \* 5.2.1. Information about Public Key Credential (interface  
6862 AuthenticatorAttestationResponse)  
6863 \* 5.10.3. Credential Descriptor (dictionary  
6864 PublicKeyCredentialDescriptor)  
6865 \* 6.2.1. Lookup Credential Source by Credential ID algorithm  
6866 \* 6.2.2. The authenticatorMakeCredential operation  
6867 \* 6.2.3. The authenticatorGetAssertion operation  
6868 \* 6.3.1. Attested credential data  
6869 \* 7.1. Registering a new credential  
6870 \* 8.6. FIDO U2F Attestation Statement Format  
6871 \* 12.1. Registration  
6872 \* 12.3. Authentication (2) (3)  
6873 \* 13.3. credentialId Unsigned (2) (3)

6874  
6875 #credential-public-keyReferenced in:  
6876 \* 4. Terminology (2) (3) (4) (5) (6) (7)  
6877 \* 5.2.1. Information about Public Key Credential (interface  
6878 AuthenticatorAttestationResponse)  
6879 \* 6. WebAuthn Authenticator Model  
6880 \* 6.3. Attestation (2) (3)  
6881 \* 6.3.1. Attested credential data (2) (3)  
6882 \* 12.1. Registration (2)  
6883 \* 13.3. credentialId Unsigned

6884  
6885 #credential-key-pairReferenced in:  
6886 \* 4. Terminology (2) (3)

6887  
6888 #credential-private-keyReferenced in:  
6889 \* 4. Terminology (2) (3) (4) (5) (6)  
6890 \* 5.1. PublicKeyCredential Interface  
6891 \* 5.2.2. Web Authentication Assertion (interface  
6892 AuthenticatorAssertionResponse)  
6893 \* 6. WebAuthn Authenticator Model  
6894 \* 6.3. Attestation (2)  
6895 \* 7.2. Verifying an authentication assertion

6896  
6897 #human-palatabilityReferenced in:  
6898 \* 4. Terminology  
6899 \* 5.4.1. Public Key Entity Description (dictionary  
6900 PublicKeyCredentialEntity) (2)

6901  
6902 #public-key-credential-sourceReferenced in:  
6903 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11)  
6904 \* 5.1.3. Create a new credential - PublicKeyCredential's  
[[Create]](origin, options, sameOriginWithAncestors) method  
6905 \* 6. WebAuthn Authenticator Model  
6906 \* 6.2.1. Lookup Credential Source by Credential ID algorithm (2)  
6907 \* 6.2.2. The authenticatorMakeCredential operation  
6908 \* 6.2.3. The authenticatorGetAssertion operation (2)

6909  
6910 #public-key-credential-source-typeReferenced in:  
6911 \* 6.2.2. The authenticatorMakeCredential operation (2)

6912  
6913 #public-key-credential-source-idReferenced in:  
6914 \* 6.2.1. Lookup Credential Source by Credential ID algorithm (2)  
6915 \* 6.2.2. The authenticatorMakeCredential operation  
6916 \* 6.2.3. The authenticatorGetAssertion operation

6917  
6918 #public-key-credential-source-privatekeyReferenced in:  
6919 \* 6.2.2. The authenticatorMakeCredential operation  
6920 \* 6.2.3. The authenticatorGetAssertion operation

6921  
6922 #public-key-credential-source-rpidReferenced in:  
6923 \* 6. WebAuthn Authenticator Model  
6924 \* 6.2.2. The authenticatorMakeCredential operation  
6925 \* 6.2.3. The authenticatorGetAssertion operation

6926  
6927 #public-key-credential-source-userhandleReferenced in:

6001 #public-key-credential-source-managing-authenticatorReferenced in:  
6002 \* 4. Terminology  
6003  
6004 #public-key-credentialReferenced in:  
6005 \* 1. Introduction (2) (3) (4) (5)  
6006 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8)  
6007 \* 5. Web Authentication API (2) (3) (4)  
6008 \* 5.1. PublicKeyCredential Interface  
6009 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6010 [[Create]](origin, options, sameOriginWithAncestors) method  
6011 \* 5.1.4. Use an existing credential to make an assertion -  
6012 PublicKeyCredential's [[Get]](options) method  
6013 \* 5.1.4.1. PublicKeyCredential's  
6014 [[DiscoverFromExternalSource]](origin, options,  
6015 sameOriginWithAncestors) method (2)  
6016 \* 5.2.1. Information about Public Key Credential (interface  
6017 AuthenticatorAttestationResponse)  
6018 \* 5.4.1. Public Key Entity Description (dictionary  
6019 PublicKeyCredentialEntity)  
6020 \* 5.4.4. Authenticator Selection Criteria (dictionary  
6021 AuthenticatorSelectionCriteria)  
6022  
6023 \* 5.5. Options for Assertion Generation (dictionary  
6024 PublicKeyCredentialRequestOptions)  
6025 \* 5.8. Supporting Data Structures  
6026 \* 6. WebAuthn Authenticator model (2) (3) (4) (5)  
6027 \* 6.2.2. The authenticatorGetAssertion operation (2) (3)  
  
6028 \* 6.3. Attestation (2)  
6029 \* 6.3.2. Attestation Statement Formats  
6030 \* 6.3.3. Attestation Types  
6031 \* 6.3.5.2. Attestation Certificate and Attestation Certificate CA  
6032 Compromise (2)  
6033 \* 7.1. Registering a new credential  
  
6034 \* 9. WebAuthn Extensions (2)  
6035 \* 12. Sample scenarios  
  
6036 #registrationReferenced in:  
6037 \* 1. Introduction (2)  
6038 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)  
6039 \* 7.1. Registering a new credential  
  
6040 #relying-partyReferenced in:  
6041 \* 1. Introduction (2) (3) (4) (5) (6) (7)  
6042 \* 1.1.3. Other use cases and configurations  
6043 \* 2.3. Relying Parties  
6044 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)  
6045 (14) (15) (16) (17) (18) (19) (20) (21) (22) (23) (24) (25) (26)  
6046 (27) (28) (29) (30)  
6047 \* 5. Web Authentication API (2) (3) (4) (5) (6) (7)  
6048 \* 5.1. PublicKeyCredential Interface (2)  
6049 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6050 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6051  
6052

6929 \* 6. WebAuthn Authenticator Model  
6930 \* 6.2.2. The authenticatorMakeCredential operation  
6931 \* 6.2.3. The authenticatorGetAssertion operation (2)  
6932  
6933 #public-key-credential-source-otheruiReferenced in:  
6934 \* 6.2.2. The authenticatorMakeCredential operation  
6935  
6936 #public-key-credential-source-managing-authenticatorReferenced in:  
6937 \* 4. Terminology  
6938  
6939 #public-key-credentialReferenced in:  
6940 \* 1. Introduction (2) (3) (4) (5)  
6941 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8)  
6942 \* 5. Web Authentication API (2) (3) (4)  
6943 \* 5.1. PublicKeyCredential Interface  
6944 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6945 [[Create]](origin, options, sameOriginWithAncestors) method  
6946 \* 5.1.4. Use an existing credential to make an assertion -  
6947 PublicKeyCredential's [[Get]](options) method  
6948 \* 5.1.4.1. PublicKeyCredential's  
6949 [[DiscoverFromExternalSource]](origin, options,  
6950 sameOriginWithAncestors) method (2)  
6951 \* 5.2.1. Information about Public Key Credential (interface  
6952 AuthenticatorAttestationResponse)  
6953 \* 5.4.1. Public Key Entity Description (dictionary  
6954 PublicKeyCredentialEntity)  
6955 \* 5.4.4. Authenticator Selection Criteria (dictionary  
6956 AuthenticatorSelectionCriteria)  
6957 \* 5.4.5. Authenticator Attachment enumeration (enum  
6958 AuthenticatorAttachment) (2) (3)  
6959 \* 5.5. Options for Assertion Generation (dictionary  
6960 PublicKeyCredentialRequestOptions)  
6961 \* 5.10. Supporting Data Structures  
6962 \* 5.10.3. Credential Descriptor (dictionary  
6963 PublicKeyCredentialDescriptor) (2) (3)  
6964 \* 6. WebAuthn Authenticator Model (2)  
6965 \* 6.2.3. The authenticatorGetAssertion operation  
6966 \* 6.3. Attestation (2)  
6967 \* 6.3.2. Attestation Statement Formats  
6968 \* 6.3.3. Attestation Types  
  
6969 \* 7.1. Registering a new credential  
6970 \* 7.2. Verifying an authentication assertion (2)  
6971 \* 9. WebAuthn Extensions (2)  
6972 \* 12. Sample scenarios  
6973 \* 13.2.2. Attestation Certificate and Attestation Certificate CA  
6974 Compromise (2)  
6975 \* 14.2. Registration Ceremony Privacy (2) (3)  
6976 \* 14.3. Authentication Ceremony Privacy (2) (3) (4) (5)  
6977  
6978 #registrationReferenced in:  
6979 \* 1. Introduction (2)  
6980 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)  
6981 \* 7.1. Registering a new credential (2) (3)  
6982 \* 10.9. Biometric Authenticator Performance Bounds Extension  
6983 (biometricPerfBounds)  
6984 \* 13. Security Considerations  
6985  
6986 #relying-partyReferenced in:  
6987 \* 1. Introduction (2) (3) (4) (5) (6) (7)  
6988 \* 1.1.3. Other use cases and configurations  
6989 \* 2.3. Relying Parties  
6990 \* 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)  
6991 (14) (15) (16) (17) (18) (19) (20) (21) (22) (23) (24) (25) (26)  
6992 (27) (28) (29) (30)  
6993 \* 5. Web Authentication API (2) (3) (4) (5) (6) (7)  
6994 \* 5.1. PublicKeyCredential Interface (2)  
6995 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6996 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)

6053 \* 5.1.4. Use an existing credential to make an assertion -  
6054 PublicKeyCredential's [[Get]](options) method (2)  
6055 \* 5.1.4.1. PublicKeyCredential's  
6056 [[DiscoverFromExternalSource]](origin, options,  
6057 sameOriginWithAncestors) method (2) (3) (4)  
6058 \* 5.1.6. Availability of User-Verifying Platform Authenticator -  
6059 PublicKeyCredential's  
6060 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3)  
6061 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
6062 \* 5.2.1. Information about Public Key Credential (interface  
6063 AuthenticatorAttestationResponse) (2)  
6064 \* 5.2.2. Web Authentication Assertion (interface  
6065 AuthenticatorAssertionResponse)  
6066 \* 5.4. Options for Credential Creation (dictionary  
6067 MakePublicKeyCredentialOptions) (2) (3) (4) (5) (6) (7)  
6068 \* 5.4.1. Public Key Entity Description (dictionary  
6069 PublicKeyCredentialEntity) (2) (3)  
6070 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
6071 PublicKeyCredentialRpEntity) (2)  
  
6072 \* 5.4.4. Authenticator Selection Criteria (dictionary  
6073 AuthenticatorSelectionCriteria) (2) (3)  
6074 \* 5.4.5. Authenticator Attachment enumeration (enum  
6075 AuthenticatorAttachment) (2) (3) (4)  
6076 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
6077 AttestationConveyancePreference) (2) (3) (4) (5) (6) (7)  
6078 \* 5.5. Options for Assertion Generation (dictionary  
6079 PublicKeyCredentialRequestOptions)  
6080 \* 5.8.1. Client data used in WebAuthn signatures (dictionary  
6081 CollectedClientData) (2) (3) (4)  
6082 \* 5.8.4. Authenticator Transport enumeration (enum  
6083 AuthenticatorTransport) (2)  
6084 \* 5.8.6. User Verification Requirement enumeration (enum  
6085 UserVerificationRequirement) (2) (3) (4)  
6086 \* 6. WebAuthn Authenticator model (2)  
6087 \* 6.1. Authenticator data (2)  
6088 \* 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6)  
6089 \* 6.2.1. The authenticatorMakeCredential operation (2) (3) (4) (5)  
6090 (6)  
6091 \* 6.2.2. The authenticatorGetAssertion operation (2) (3)  
6092 \* 6.3. Attestation (2) (3) (4) (5) (6)  
6093 \* 6.3.5.1. Privacy  
6094 \* 6.3.5.2. Attestation Certificate and Attestation Certificate CA  
6095 Compromise (2) (3) (4) (5) (6)  
6096 \* 7. Relying Party Operations (2) (3) (4)  
6097 \* 7.1. Registering a new credential (2) (3) (4) (5) (6) (7) (8) (9)  
6098 (10) (11) (12)  
6099 \* 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6) (7)  
6100 (8)  
6101 \* 8.4. Android Key Attestation Statement Format  
6102 \* 9. WebAuthn Extensions (2) (3) (4)  
  
6103 \* 9.2. Defining extensions (2)  
6104 \* 9.3. Extending request parameters (2) (3) (4)  
6105 \* 9.6. Example Extension (2) (3)  
6106 \* 10.1. FIDO Appid Extension (appid) (2)  
6107 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
6108 \* 10.4. Authenticator Selection Extension (authnSel) (2) (3)  
6109 \* 10.5. Supported Extensions Extension (exts) (2)  
6110 \* 10.6. User Verification Index Extension (uvi)  
6111 \* 10.7. Location Extension (loc) (2)  
  
6112 \* 11.2. WebAuthn Extension Identifier Registrations (2)  
6113 \* 12.1. Registration (2) (3) (4) (5)  
6114 \* 12.2. Registration Specifically with User Verifying Platform  
6115 Authenticator (2) (3)

6997 (4) (5)  
6998 \* 5.1.4. Use an existing credential to make an assertion -  
6999 PublicKeyCredential's [[Get]](options) method (2)  
7000 \* 5.1.4.1. PublicKeyCredential's  
7001 [[DiscoverFromExternalSource]](origin, options,  
7002 sameOriginWithAncestors) method (2) (3) (4)  
7003 \* 5.1.7. Availability of User-Verifying Platform Authenticator -  
7004 PublicKeyCredential's  
7005 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3)  
7006 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
7007 \* 5.2.1. Information about Public Key Credential (interface  
7008 AuthenticatorAttestationResponse) (2)  
7009 \* 5.2.2. Web Authentication Assertion (interface  
7010 AuthenticatorAssertionResponse)  
7011 \* 5.4. Options for Credential Creation (dictionary  
7012 PublicKeyCredentialCreationOptions) (2) (3) (4) (5)  
7013 \* 5.4.1. Public Key Entity Description (dictionary  
7014 PublicKeyCredentialEntity) (2) (3) (4) (5)  
7015 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
7016 PublicKeyCredentialRpEntity) (2)  
7017 \* 5.4.3. User Account Parameters for Credential Generation  
7018 (dictionary PublicKeyCredentialUserEntity)  
7019 \* 5.4.4. Authenticator Selection Criteria (dictionary  
7020 AuthenticatorSelectionCriteria) (2) (3)  
7021 \* 5.4.5. Authenticator Attachment enumeration (enum  
7022 AuthenticatorAttachment) (2) (3) (4) (5) (6)  
7023 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
7024 AttestationConveyancePreference) (2) (3) (4) (5) (6) (7)  
7025 \* 5.5. Options for Assertion Generation (dictionary  
7026 PublicKeyCredentialRequestOptions)  
7027 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
7028 CollectedClientData) (2) (3) (4)  
7029 \* 5.10.4. Authenticator Transport enumeration (enum  
7030 AuthenticatorTransport) (2)  
7031 \* 5.10.6. User Verification Requirement enumeration (enum  
7032 UserVerificationRequirement) (2) (3) (4)  
7033 \* 6. WebAuthn Authenticator Model (2) (3)  
7034 \* 6.1. Authenticator data (2)  
7035 \* 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6)  
7036 \* 6.2.2. The authenticatorMakeCredential operation (2) (3) (4) (5)  
7037 (6)  
7038 \* 6.2.3. The authenticatorGetAssertion operation (2) (3)  
7039 \* 6.3. Attestation (2) (3) (4) (5) (6)  
7040 \* 6.3.3. Attestation Types  
  
7041 \* 7. Relying Party Operations (2) (3) (4)  
7042 \* 7.1. Registering a new credential (2) (3) (4) (5) (6) (7) (8) (9)  
7043 (10) (11) (12) (13)  
7044 \* 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6) (7)  
7045 (8)  
7046 \* 8.4. Android Key Attestation Statement Format  
7047 \* 8.7. None Attestation Statement Format  
7048 \* 9. WebAuthn Extensions (2) (3) (4) (5)  
7049 \* 9.2. Defining extensions (2)  
7050 \* 9.3. Extending request parameters (2) (3) (4)  
7051 \* 10.1. FIDO Appid Extension (appid) (2)  
7052 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
7053 \* 10.4. Authenticator Selection Extension (authnSel) (2) (3)  
7054 \* 10.5. Supported Extensions Extension (exts) (2)  
7055 \* 10.6. User Verification Index Extension (uvi)  
7056 \* 10.7. Location Extension (loc) (2)  
7057 \* 10.8. User Verification Method Extension (uvm)  
7058 \* 10.9. Biometric Authenticator Performance Bounds Extension  
7059 (biometricPerfBounds) (2) (3)  
7060 \* 11.2. WebAuthn Extension Identifier Registrations (2)  
7061 \* 12.1. Registration (2) (3) (4) (5)  
7062 \* 12.2. Registration Specifically with User Verifying Platform  
7063 Authenticator (2) (3)

6116 \* 12.3. Authentication (2) (3) (4) (5)  
6117 \* 12.5. Decommissioning (2)

6118 \* 13.1. Cryptographic Challenges

6119  
6120 #relying-party-identifierReferenced in:  
6121 \* 4. Terminology  
6122 \* 5. Web Authentication API  
6123 \* 5.4. Options for Credential Creation (dictionary  
6124 MakePublicKeyCredentialOptions)  
6125 \* 5.5. Options for Assertion Generation (dictionary  
6126 PublicKeyCredentialRequestOptions)  
6127 \* 6. WebAuthn Authenticator model

6128  
6129 #rp-idReferenced in:  
6130 \* 4. Terminology (2) (3) (4) (5)  
6131 \* 5. Web Authentication API (2) (3) (4) (5)  
6132 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6133 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6134 \* 5.1.4.1. PublicKeyCredential's  
6135 [[DiscoverFromExternalSource]](origin, options,  
6136 sameOriginWithAncestors) method (2)  
6137 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
6138 PublicKeyCredentialRpEntity)  
6139 \* 6. WebAuthn Authenticator model  
6140 \* 6.1. Authenticator data (2) (3) (4) (5) (6)  
6141 \* 6.1.1. Signature Counter Considerations  
6142 \* 6.2.1. The authenticatorMakeCredential operation (2)  
6143 \* 6.2.2. The authenticatorGetAssertion operation (2)  
6144 \* 7.1. Registering a new credential (2)  
6145 \* 7.2. Verifying an authentication assertion  
6146 \* 8.4. Android Key Attestation Statement Format  
6147 \* 8.6. FIDO U2F Attestation Statement Format

6148  
6149 #test-of-user-presenceReferenced in:  
6150 \* 4. Terminology (2) (3) (4) (5) (6)  
6151 \* 6.2.1. The authenticatorMakeCredential operation  
6152 \* 6.2.2. The authenticatorGetAssertion operation  
6153 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
6154 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
6155

6156 #user-consentReferenced in:  
6157 \* 1. Introduction (2)  
6158 \* 4. Terminology (2)  
6159 \* 5. Web Authentication API

6160 \* 5.1.4. Use an existing credential to make an assertion -  
6161 PublicKeyCredential's [[Get]](options) method

6162 \* 5.2.2. Web Authentication Assertion (interface  
6163 AuthenticatorAssertionResponse)  
6164 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
6165 AttestationConveyancePreference)  
6166 \* 6. WebAuthn Authenticator model (2) (3)  
6167 \* 6.2.1. The authenticatorMakeCredential operation (2) (3) (4) (5)  
6168 (6)  
6169 \* 6.2.2. The authenticatorGetAssertion operation (2) (3) (4) (5)  
6170 \* 11.2. WebAuthn Extension Identifier Registrations

7064 \* 12.3. Authentication (2) (3) (4) (5)  
7065 \* 12.5. Decommissioning (2)  
7066 \* 13. Security Considerations (2) (3) (4)  
7067 \* 13.1. Cryptographic Challenges  
7068 \* 13.2.2. Attestation Certificate and Attestation Certificate CA  
7069 Compromise (2) (3) (4) (5) (6)  
7070 \* 13.3. credentialId Unsigned  
7071 \* 14.1. Attestation Privacy  
7072 \* 14.2. Registration Ceremony Privacy (2) (3) (4)  
7073 \* 14.3. Authentication Ceremony Privacy (2) (3) (4)

7074  
7075 #relying-party-identifierReferenced in:  
7076 \* 4. Terminology  
7077 \* 5. Web Authentication API  
7078 \* 5.4. Options for Credential Creation (dictionary  
7079 PublicKeyCredentialCreationOptions)  
7080 \* 5.5. Options for Assertion Generation (dictionary  
7081 PublicKeyCredentialRequestOptions)

7082  
7083 #rp-idReferenced in:  
7084 \* 4. Terminology (2) (3) (4) (5)  
7085 \* 5. Web Authentication API (2) (3) (4) (5)  
7086 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7087 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
7088 \* 5.1.4.1. PublicKeyCredential's  
7089 [[DiscoverFromExternalSource]](origin, options,  
7090 sameOriginWithAncestors) method (2)  
7091 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
7092 PublicKeyCredentialRpEntity)  
7093 \* 6.1. Authenticator data (2) (3) (4) (5) (6)  
7094 \* 6.1.1. Signature Counter Considerations  
7095 \* 6.2.2. The authenticatorMakeCredential operation (2) (3)  
7096 \* 6.2.3. The authenticatorGetAssertion operation (2)  
7097 \* 7.1. Registering a new credential (2)  
7098 \* 7.2. Verifying an authentication assertion  
7099 \* 8.4. Android Key Attestation Statement Format  
7100 \* 8.6. FIDO U2F Attestation Statement Format  
7101 \* 10.1. FIDO AppID Extension (appid)

7102  
7103 #test-of-user-presenceReferenced in:  
7104 \* 4. Terminology (2) (3) (4) (5) (6)  
7105 \* 6.2.2. The authenticatorMakeCredential operation (2)  
7106 \* 6.2.3. The authenticatorGetAssertion operation  
7107 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
7108 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
7109

7110 #user-consentReferenced in:  
7111 \* 1. Introduction (2)  
7112 \* 4. Terminology (2)  
7113 \* 5. Web Authentication API  
7114 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7115 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
7116 \* 5.1.4. Use an existing credential to make an assertion -  
7117 PublicKeyCredential's [[Get]](options) method  
7118 \* 5.1.4.1. PublicKeyCredential's  
7119 [[DiscoverFromExternalSource]](origin, options,  
7120 sameOriginWithAncestors) method  
7121 \* 5.2.2. Web Authentication Assertion (interface  
7122 AuthenticatorAssertionResponse)  
7123 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
7124 AttestationConveyancePreference)  
7125 \* 6. WebAuthn Authenticator Model (2) (3)  
7126 \* 6.2.2. The authenticatorMakeCredential operation (2) (3) (4) (5)  
7127 (6) (7) (8)  
7128 \* 6.2.3. The authenticatorGetAssertion operation (2) (3) (4) (5)  
7129 \* 11.2. WebAuthn Extension Identifier Registrations  
7130 \* 14.2. Registration Ceremony Privacy (2)  
7131 \* 14.3. Authentication Ceremony Privacy (2) (3)

```

6171 #user-handleReferenced in:
6172
6173 * 4. Terminology
6174 * 5.1.4.1. PublicKeyCredential's
6175 [[DiscoverFromExternalSource]](origin, options,
6176 sameOriginWithAncestors) method
6177 * 5.2.2. Web Authentication Assertion (interface
6178 AuthenticatorAssertionResponse)
6179 * 5.4. Options for Credential Creation (dictionary
6180 MakePublicKeyCredentialOptions)
6181 * 5.4.3. User Account Parameters for Credential Generation
6182 (dictionary PublicKeyCredentialUserEntity)
6183 * 6.2.1. The authenticatorMakeCredential operation
6184 * 6.2.2. The authenticatorGetAssertion operation
6185
6186 #user-verificationReferenced in:
6187 * 1. Introduction
6188 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
6189 * 5.1.3. Create a new credential - PublicKeyCredential's
6190 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
6191 * 5.1.4.1. PublicKeyCredential's
6192 [[DiscoverFromExternalSource]](origin, options,
6193 sameOriginWithAncestors) method (2) (3)
6194 * 5.1.6. Availability of User-Verifying Platform Authenticator -
6195 PublicKeyCredential's
6196 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)
6197 (5)
6198 * 5.4.4. Authenticator Selection Criteria (dictionary
6199 AuthenticatorSelectionCriteria)
6200 * 5.5. Options for Assertion Generation (dictionary
6201 PublicKeyCredentialRequestOptions)
6202 * 5.8.6. User Verification Requirement enumeration (enum
6203 UserVerificationRequirement) (2) (3) (4)
6204 * 6.2.1. The authenticatorMakeCredential operation (2) (3)
6205 * 6.2.2. The authenticatorGetAssertion operation (2) (3)
6206
6207 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
6208 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
6209 * 12.2. Registration Specifically with User Verifying Platform
6210 Authenticator
6211
6212 #concept-user-presentReferenced in:
6213 * 4. Terminology
6214 * 6.1. Authenticator data (2) (3)
6215
6216 #upReferenced in:
6217 * 6.1. Authenticator data
6218
6219 #concept-user-verifiedReferenced in:
6220 * 4. Terminology
6221 * 6.1. Authenticator data (2) (3)
6222
6223 #uvReferenced in:
6224 * 5.8.6. User Verification Requirement enumeration (enum
6225 UserVerificationRequirement) (2)
6226 * 6.1. Authenticator data
6227
6228 #webauthn-clientReferenced in:
6229 * 4. Terminology (2)
6230 * 6.2.1. The authenticatorMakeCredential operation
6231 * 6.2.2. The authenticatorGetAssertion operation

```

```

7132 #user-handleReferenced in:
7133 * 2.2.1. Backwards Compatibility with FIDO U2F
7134 * 4. Terminology
7135 * 5.1.4.1. PublicKeyCredential's
7136 [[DiscoverFromExternalSource]](origin, options,
7137 sameOriginWithAncestors) method (2)
7138 * 5.2.2. Web Authentication Assertion (interface
7139 AuthenticatorAssertionResponse) (2)
7140
7141 * 5.4.3. User Account Parameters for Credential Generation
7142 (dictionary PublicKeyCredentialUserEntity)
7143 * 6.2.2. The authenticatorMakeCredential operation
7144
7145 #user-verificationReferenced in:
7146 * 1. Introduction
7147 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
7148 * 5.1.3. Create a new credential - PublicKeyCredential's
7149 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
7150 * 5.1.4.1. PublicKeyCredential's
7151 [[DiscoverFromExternalSource]](origin, options,
7152 sameOriginWithAncestors) method (2) (3)
7153 * 5.1.7. Availability of User-Verifying Platform Authenticator -
7154 PublicKeyCredential's
7155 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)
7156 (5)
7157 * 5.4.4. Authenticator Selection Criteria (dictionary
7158 AuthenticatorSelectionCriteria)
7159 * 5.5. Options for Assertion Generation (dictionary
7160 PublicKeyCredentialRequestOptions)
7161 * 5.10.6. User Verification Requirement enumeration (enum
7162 UserVerificationRequirement) (2) (3) (4)
7163 * 6.2.2. The authenticatorMakeCredential operation (2) (3)
7164 * 6.2.3. The authenticatorGetAssertion operation
7165 * 7.1. Registering a new credential (2)
7166 * 7.2. Verifying an authentication assertion (2)
7167 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
7168 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
7169 * 12.2. Registration Specifically with User Verifying Platform
7170 Authenticator
7171
7172 #concept-user-presentReferenced in:
7173 * 4. Terminology
7174 * 6.1. Authenticator data (2) (3)
7175 * 7.1. Registering a new credential
7176 * 7.2. Verifying an authentication assertion
7177
7178 #upReferenced in:
7179 * 6.1. Authenticator data
7180
7181 #concept-user-verifiedReferenced in:
7182 * 4. Terminology
7183 * 6.1. Authenticator data (2) (3)
7184 * 7.1. Registering a new credential
7185 * 7.2. Verifying an authentication assertion
7186
7187 #uvReferenced in:
7188 * 5.10.6. User Verification Requirement enumeration (enum
7189 UserVerificationRequirement) (2)
7190 * 6.1. Authenticator data
7191
7192 #webauthn-clientReferenced in:
7193 * 4. Terminology (2) (3) (4)
7194 * 6.2. Authenticator operations
7195 * 6.2.2. The authenticatorMakeCredential operation
7196 * 6.2.3. The authenticatorGetAssertion operation
7197 * 13. Security Considerations
7198

```

```

6232 #web-authentication-apiReferenced in:
6233 * 1. Introduction (2) (3)
6234 * 4. Terminology (2)

6235
6236 #publickeycredentialReferenced in:
6237 * 1. Introduction
6238 * 5.1. PublicKeyCredential Interface (2) (3) (4) (5) (6) (7) (8)
6239 * 5.1.3. Create a new credential - PublicKeyCredential's
6240 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6241 * 5.1.4.1. PublicKeyCredential's
6242 [[DiscoverFromExternalSource]](origin, options,
6243 sameOriginWithAncestors) method (2)
6244 * 5.1.5. Store an existing credential - PublicKeyCredential's
6245 [[Store]](credential, sameOriginWithAncestors) method (2)
6246 * 5.1.6. Availability of User-Verifying Platform Authenticator -
6247 PublicKeyCredential's
6248 isUserVerifyingPlatformAuthenticatorAvailable() method
6249 * 5.8.3. Credential Descriptor (dictionary
6250 PublicKeyCredentialDescriptor)
6251 * 7. Relying Party Operations
6252 * 7.2. Verifying an authentication assertion
6253
6254 #dom-publickeycredential-rawidReferenced in:
6255 * 5.1. PublicKeyCredential Interface
6256 * 7.2. Verifying an authentication assertion
6257
6258 #dom-publickeycredential-getclientextensionresultsReferenced in:
6259 * 5.1. PublicKeyCredential Interface
6260 * 9.4. Client extension processing
6261
6262 #dom-publickeycredential-responseReferenced in:
6263 * 5.1. PublicKeyCredential Interface
6264 * 5.1.3. Create a new credential - PublicKeyCredential's
6265 [[Create]](origin, options, sameOriginWithAncestors) method
6266 * 5.1.4.1. PublicKeyCredential's
6267 [[DiscoverFromExternalSource]](origin, options,
6268 sameOriginWithAncestors) method
6269 * 7.2. Verifying an authentication assertion
6270
6271 #dom-publickeycredential-identifier-slotReferenced in:
6272 * 5.1. PublicKeyCredential Interface (2)
6273 * 5.1.3. Create a new credential - PublicKeyCredential's
6274 [[Create]](origin, options, sameOriginWithAncestors) method
6275 * 5.1.4.1. PublicKeyCredential's
6276 [[DiscoverFromExternalSource]](origin, options,
6277 sameOriginWithAncestors) method
6278
6279 #dom-publickeycredential-clientextensionsresults-slotReferenced in:
6280 * 5.1. PublicKeyCredential Interface
6281 * 5.1.3. Create a new credential - PublicKeyCredential's
6282 [[Create]](origin, options, sameOriginWithAncestors) method
6283 * 5.1.4.1. PublicKeyCredential's
6284 [[DiscoverFromExternalSource]](origin, options,
6285 sameOriginWithAncestors) method
6286
6287 #dom-credentialcreationoptions-publickeyReferenced in:
6288 * 5.1.3. Create a new credential - PublicKeyCredential's
6289 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
6290
6291 #dom-credentialrequestoptions-publickeyReferenced in:
6292 * 5.1.4.1. PublicKeyCredential's
6293 [[DiscoverFromExternalSource]](origin, options,
6294 sameOriginWithAncestors) method (2) (3)
6295
6296 #dom-publickeycredential-create-slotReferenced in:
6297
6298 * 5.1. PublicKeyCredential Interface

```

```

7199 #web-authentication-apiReferenced in:
7200 * 1. Introduction (2) (3)
7201 * 4. Terminology (2) (3) (4)
7202 * 13. Security Considerations

7203
7204 #publickeycredentialReferenced in:
7205 * 1. Introduction
7206 * 5.1. PublicKeyCredential Interface (2) (3) (4) (5) (6) (7) (8)
7207 * 5.1.3. Create a new credential - PublicKeyCredential's
7208 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7209 * 5.1.4.1. PublicKeyCredential's
7210 [[DiscoverFromExternalSource]](origin, options,
7211 sameOriginWithAncestors) method
7212 * 5.1.5. Store an existing credential - PublicKeyCredential's
7213 [[Store]](credential, sameOriginWithAncestors) method (2)
7214 * 5.1.7. Availability of User-Verifying Platform Authenticator -
7215 PublicKeyCredential's
7216 isUserVerifyingPlatformAuthenticatorAvailable() method
7217 * 5.10.3. Credential Descriptor (dictionary
7218 PublicKeyCredentialDescriptor)
7219 * 7. Relying Party Operations
7220 * 7.2. Verifying an authentication assertion

7221
7222 #dom-publickeycredential-rawidReferenced in:
7223 * 5.1. PublicKeyCredential Interface
7224 * 7.2. Verifying an authentication assertion
7225
7226 #dom-publickeycredential-getclientextensionresultsReferenced in:
7227 * 5.1. PublicKeyCredential Interface
7228 * 9.4. Client extension processing
7229
7230 #dom-publickeycredential-responseReferenced in:
7231 * 5.1. PublicKeyCredential Interface
7232 * 5.1.3. Create a new credential - PublicKeyCredential's
7233 [[Create]](origin, options, sameOriginWithAncestors) method
7234 * 5.1.4.1. PublicKeyCredential's
7235 [[DiscoverFromExternalSource]](origin, options,
7236 sameOriginWithAncestors) method
7237 * 7.2. Verifying an authentication assertion (2)
7238
7239 #dom-publickeycredential-identifier-slotReferenced in:
7240 * 5.1. PublicKeyCredential Interface (2)
7241 * 5.1.3. Create a new credential - PublicKeyCredential's
7242 [[Create]](origin, options, sameOriginWithAncestors) method
7243 * 5.1.4.1. PublicKeyCredential's
7244 [[DiscoverFromExternalSource]](origin, options,
7245 sameOriginWithAncestors) method
7246
7247 #dom-publickeycredential-clientextensionsresults-slotReferenced in:
7248 * 5.1. PublicKeyCredential Interface
7249 * 5.1.3. Create a new credential - PublicKeyCredential's
7250 [[Create]](origin, options, sameOriginWithAncestors) method
7251 * 5.1.4.1. PublicKeyCredential's
7252 [[DiscoverFromExternalSource]](origin, options,
7253 sameOriginWithAncestors) method
7254
7255 #dom-credentialcreationoptions-publickeyReferenced in:
7256 * 5.1.3. Create a new credential - PublicKeyCredential's
7257 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
7258
7259 #dom-credentialrequestoptions-publickeyReferenced in:
7260 * 5.1.4.1. PublicKeyCredential's
7261 [[DiscoverFromExternalSource]](origin, options,
7262 sameOriginWithAncestors) method (2) (3)
7263
7264 #dom-publickeycredential-create-slotReferenced in:
7265 * 4. Terminology
7266 * 5.1. PublicKeyCredential Interface
7267 * 5.4.5. Authenticator Attachment enumeration (enum
7268 AuthenticatorAttachment)

```

629E \* 5.6. Abort operations with AbortSignal (2) (3) (4) (5)  
629F \* 6.2.1. The authenticatorMakeCredential operation

630C  
6301 #dom-publickeycredential-create-origin-options-sameoriginwithancestors-  
6302 originReferenced in:  
6303 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6304 [[Create]](origin, options, sameOriginWithAncestors) method  
6305  
6306 #dom-publickeycredential-create-origin-options-sameoriginwithancestors-  
6307 optionsReferenced in:  
6308 \* 7.1. Registering a new credential

630E  
631C #effective-user-verification-requirement-for-credential-creationReferen  
6311 ced in:  
6312 \* 6.2.1. The authenticatorMakeCredential operation

6313  
6314 #credentialcreationdata-attestationobjectresultReferenced in:  
6315 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6316 [[Create]](origin, options, sameOriginWithAncestors) method

6317  
6318 #credentialcreationdata-clientdatajsonresultReferenced in:  
6319 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6320 [[Create]](origin, options, sameOriginWithAncestors) method

6321  
6322 #credentialcreationdata-attestationconveyancepreferenceoptionReferenced  
6323 in:  
6324 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6325 [[Create]](origin, options, sameOriginWithAncestors) method

6326  
6327 #credentialcreationdata-clientextensionresultsReferenced in:  
6328 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6329 [[Create]](origin, options, sameOriginWithAncestors) method

6330  
6331 #dom-publickeycredential-collectfromcredentialstore-slotReferenced in:  
6332 \* 5.1.4. Use an existing credential to make an assertion -  
6333 PublicKeyCredential's [[Get]](options) method

6334  
6335 #dom-publickeycredential-discoverfromexternalsource-slotReferenced in:  
6336 \* 5.1. PublicKeyCredential Interface  
6337 \* 5.1.4. Use an existing credential to make an assertion -  
6338 PublicKeyCredential's [[Get]](options) method

6339  
633E \* 5.6. Abort operations with AbortSignal (2) (3) (4) (5)  
6340 \* 6.2.2. The authenticatorGetAssertion operation

6341  
6342 #dom-publickeycredential-discoverfromexternalsource-origin-options-same  
6343 originwithancestors-originReferenced in:  
6344 \* 5.1.4.1. PublicKeyCredential's  
6345 [[DiscoverFromExternalSource]](origin, options,  
6346 sameOriginWithAncestors) method

6347  
6348 #effective-user-verification-requirement-for-assertionReferenced in:  
6349 \* 6.2.2. The authenticatorGetAssertion operation

6350  
6351 #assertioncreationdata-credentialidresultReferenced in:  
6352 \* 5.1.4.1. PublicKeyCredential's  
6353 [[DiscoverFromExternalSource]](origin, options,  
6354 sameOriginWithAncestors) method (2) (3)

6355  
6356 #assertioncreationdata-clientdatajsonresultReferenced in:  
6357 \* 5.1.4.1. PublicKeyCredential's  
6358 [[DiscoverFromExternalSource]](origin, options,  
6359 sameOriginWithAncestors) method

6360  
6361 #assertioncreationdata-authenticatordataresultReferenced in:

726E \* 5.6. Abort operations with AbortSignal (2) (3) (4) (5)  
7270 \* 6.2.2. The authenticatorMakeCredential operation  
7271 \* 14.2. Registration Ceremony Privacy

7272  
7273 #dom-publickeycredential-create-origin-options-sameoriginwithancestors-  
7274 originReferenced in:  
7275 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7276 [[Create]](origin, options, sameOriginWithAncestors) method

7277  
7278 #dom-publickeycredential-create-origin-options-sameoriginwithancestors-  
7279 optionsReferenced in:  
7280 \* 7.1. Registering a new credential

7281  
7282 #effective-user-verification-requirement-for-credential-creationReferen  
7283 ced in:  
7284 \* 6.2.2. The authenticatorMakeCredential operation

7285  
7286 #credentialcreationdata-attestationobjectresultReferenced in:  
7287 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7288 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
7289 (4) (5)

7290  
7291 #credentialcreationdata-clientdatajsonresultReferenced in:  
7292 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7293 [[Create]](origin, options, sameOriginWithAncestors) method

7294  
7295 #credentialcreationdata-attestationconveyancepreferenceoptionReferenced  
7296 in:  
7297 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7298 [[Create]](origin, options, sameOriginWithAncestors) method

7299  
7300 #credentialcreationdata-clientextensionresultsReferenced in:  
7301 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7302 [[Create]](origin, options, sameOriginWithAncestors) method

7303  
7304 #dom-publickeycredential-collectfromcredentialstore-slotReferenced in:  
7305 \* 5.1.4. Use an existing credential to make an assertion -  
7306 PublicKeyCredential's [[Get]](options) method

7307  
7308 #dom-publickeycredential-discoverfromexternalsource-slotReferenced in:  
7309 \* 4. Terminology  
7310 \* 5.1. PublicKeyCredential Interface  
7311 \* 5.1.4. Use an existing credential to make an assertion -  
7312 PublicKeyCredential's [[Get]](options) method  
7313 \* 5.4.5. Authenticator Attachment enumeration (enum  
7314 AuthenticatorAttachment)

7315  
7316 \* 5.6. Abort operations with AbortSignal (2) (3) (4) (5)  
7317 \* 6.2.3. The authenticatorGetAssertion operation  
7318 \* 14.3. Authentication Ceremony Privacy

7319  
731E #dom-publickeycredential-discoverfromexternalsource-origin-options-same  
7320 originwithancestors-originReferenced in:  
7321 \* 5.1.4.1. PublicKeyCredential's  
7322 [[DiscoverFromExternalSource]](origin, options,  
7323 sameOriginWithAncestors) method

7324  
7325 #effective-user-verification-requirement-for-assertionReferenced in:  
7326 \* 6.2.3. The authenticatorGetAssertion operation

7327  
7328 #assertioncreationdata-credentialidresultReferenced in:  
7329 \* 5.1.4.1. PublicKeyCredential's  
7330 [[DiscoverFromExternalSource]](origin, options,  
7331 sameOriginWithAncestors) method (2) (3)

7332  
7333 #assertioncreationdata-clientdatajsonresultReferenced in:  
7334 \* 5.1.4.1. PublicKeyCredential's  
7335 [[DiscoverFromExternalSource]](origin, options,  
7336 sameOriginWithAncestors) method

7337  
733E #assertioncreationdata-authenticatordataresultReferenced in:

6362 \* 5.1.4.1. PublicKeyCredential's  
6363 [[DiscoverFromExternalSource]](origin, options,  
6364 sameOriginWithAncestors) method  
6365  
6366 #assertioncreationdata-signatureresultReferenced in:  
6367 \* 5.1.4.1. PublicKeyCredential's  
6368 [[DiscoverFromExternalSource]](origin, options,  
6369 sameOriginWithAncestors) method  
6370  
6371 #assertioncreationdata-userhandlerresultReferenced in:  
6372 \* 5.1.4.1. PublicKeyCredential's  
6373 [[DiscoverFromExternalSource]](origin, options,  
6374 sameOriginWithAncestors) method  
6375  
6376 #assertioncreationdata-clientextensionresultsReferenced in:  
6377 \* 5.1.4.1. PublicKeyCredential's  
6378 [[DiscoverFromExternalSource]](origin, options,  
6379 sameOriginWithAncestors) method  
6380  
6381 #authenticatorresponseReferenced in:  
6382 \* 5.1. PublicKeyCredential Interface (2)  
6383 \* 5.2. Authenticator Responses (interface AuthenticatorResponse) (2)  
6384 \* 5.2.1. Information about Public Key Credential (interface  
6385 AuthenticatorAttestationResponse) (2)  
6386 \* 5.2.2. Web Authentication Assertion (interface  
6387 AuthenticatorAssertionResponse) (2)  
6388  
6389 #dom-authenticatorresponse-clientdatajsonReferenced in:  
6390 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6391 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6392 \* 5.1.4.1. PublicKeyCredential's  
6393 [[DiscoverFromExternalSource]](origin, options,  
6394 sameOriginWithAncestors) method (2)  
6395 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
6396 \* 5.2.1. Information about Public Key Credential (interface  
6397 AuthenticatorAttestationResponse)  
6398 \* 5.2.2. Web Authentication Assertion (interface  
6399 AuthenticatorAssertionResponse)  
6400 \* 7.1. Registering a new credential (2)  
6401 \* 7.2. Verifying an authentication assertion  
6402  
6403 #authenticatorattestationresponseReferenced in:  
6404 \* 5.1. PublicKeyCredential Interface  
6405 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6406 [[Create]](origin, options, sameOriginWithAncestors) method  
6407 \* 5.2.1. Information about Public Key Credential (interface  
6408 AuthenticatorAttestationResponse) (2)  
6409 \* 7. Relying Party Operations  
6410 \* 7.1. Registering a new credential (2) (3)  
6411  
6412 #dom-authenticatorattestationresponse-attestationobjectReferenced in:  
6413 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6414 [[Create]](origin, options, sameOriginWithAncestors) method  
6415 \* 5.2.1. Information about Public Key Credential (interface  
6416 AuthenticatorAttestationResponse)  
6417 \* 7.1. Registering a new credential  
6418  
6419 #authenticatorassertionresponseReferenced in:  
6420 \* 4. Terminology  
6421 \* 5.1. PublicKeyCredential Interface  
6422 \* 5.1.4.1. PublicKeyCredential's  
6423 [[DiscoverFromExternalSource]](origin, options,  
6424 sameOriginWithAncestors) method  
6425 \* 5.2.2. Web Authentication Assertion (interface  
6426 AuthenticatorAssertionResponse) (2)  
6427 \* 7. Relying Party Operations  
6428  
6429 #dom-authenticatorassertionresponse-authenticatordataReferenced in:  
6430 \* 5.1.4.1. PublicKeyCredential's

7339 \* 5.1.4.1. PublicKeyCredential's  
7340 [[DiscoverFromExternalSource]](origin, options,  
7341 sameOriginWithAncestors) method  
7342  
7343 #assertioncreationdata-signatureresultReferenced in:  
7344 \* 5.1.4.1. PublicKeyCredential's  
7345 [[DiscoverFromExternalSource]](origin, options,  
7346 sameOriginWithAncestors) method  
7347  
7348 #assertioncreationdata-userhandlerresultReferenced in:  
7349 \* 5.1.4.1. PublicKeyCredential's  
7350 [[DiscoverFromExternalSource]](origin, options,  
7351 sameOriginWithAncestors) method (2) (3) (4)  
7352 \* 6.2.3. The authenticatorGetAssertion operation  
7353  
7354 #assertioncreationdata-clientextensionresultsReferenced in:  
7355 \* 5.1.4.1. PublicKeyCredential's  
7356 [[DiscoverFromExternalSource]](origin, options,  
7357 sameOriginWithAncestors) method  
7358  
7359 #authenticatorresponseReferenced in:  
7360 \* 5.1. PublicKeyCredential Interface (2)  
7361 \* 5.2. Authenticator Responses (interface AuthenticatorResponse) (2)  
7362 \* 5.2.1. Information about Public Key Credential (interface  
7363 AuthenticatorAttestationResponse) (2)  
7364 \* 5.2.2. Web Authentication Assertion (interface  
7365 AuthenticatorAssertionResponse) (2)  
7366  
7367 #dom-authenticatorresponse-clientdatajsonReferenced in:  
7368 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7369 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
7370 \* 5.1.4.1. PublicKeyCredential's  
7371 [[DiscoverFromExternalSource]](origin, options,  
7372 sameOriginWithAncestors) method (2)  
7373 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
7374 \* 5.2.1. Information about Public Key Credential (interface  
7375 AuthenticatorAttestationResponse)  
7376 \* 5.2.2. Web Authentication Assertion (interface  
7377 AuthenticatorAssertionResponse)  
7378 \* 7.1. Registering a new credential (2)  
7379 \* 7.2. Verifying an authentication assertion  
7380  
7381 #authenticatorattestationresponseReferenced in:  
7382 \* 5.1. PublicKeyCredential Interface  
7383 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7384 [[Create]](origin, options, sameOriginWithAncestors) method  
7385 \* 5.2.1. Information about Public Key Credential (interface  
7386 AuthenticatorAttestationResponse) (2)  
7387 \* 7. Relying Party Operations  
7388 \* 7.1. Registering a new credential (2)  
7389  
7390 #dom-authenticatorattestationresponse-attestationobjectReferenced in:  
7391 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7392 [[Create]](origin, options, sameOriginWithAncestors) method  
7393 \* 5.2.1. Information about Public Key Credential (interface  
7394 AuthenticatorAttestationResponse)  
7395 \* 7.1. Registering a new credential  
7396  
7397 #authenticatorassertionresponseReferenced in:  
7398 \* 4. Terminology  
7399 \* 5.1. PublicKeyCredential Interface  
7400 \* 5.1.4.1. PublicKeyCredential's  
7401 [[DiscoverFromExternalSource]](origin, options,  
7402 sameOriginWithAncestors) method  
7403 \* 5.2.2. Web Authentication Assertion (interface  
7404 AuthenticatorAssertionResponse) (2)  
7405 \* 7. Relying Party Operations  
7406  
7407 #dom-authenticatorassertionresponse-authenticatordataReferenced in:  
7408 \* 5.1.4.1. PublicKeyCredential's

6431 [[DiscoverFromExternalSource]](origin, options,  
6432 sameOriginWithAncestors) method  
6433 \* 5.2.2. Web Authentication Assertion (interface  
6434 AuthenticatorAssertionResponse)  
6435 \* 7.2. Verifying an authentication assertion  
6436  
6437 #dom-authenticatorassertionresponse-signatureReferenced in:  
6438 \* 5.1.4.1. PublicKeyCredential's  
6439 [[DiscoverFromExternalSource]](origin, options,  
6440 sameOriginWithAncestors) method  
6441 \* 5.2.2. Web Authentication Assertion (interface  
6442 AuthenticatorAssertionResponse)  
6443 \* 7.2. Verifying an authentication assertion  
6444  
6445 #dom-authenticatorassertionresponse-userhandleReferenced in:  
6446  
6447 \* 5.1.4.1. PublicKeyCredential's  
6448 [[DiscoverFromExternalSource]](origin, options,  
6449 sameOriginWithAncestors) method  
6450 \* 5.2.2. Web Authentication Assertion (interface  
6451 AuthenticatorAssertionResponse)  
6452  
6453 #dictdef-publickeycredentialparametersReferenced in:  
6454 \* 5.3. Parameters for Credential Generation (dictionary  
6455 PublicKeyCredentialParameters)  
6456 \* 5.4. Options for Credential Creation (dictionary  
6457 MakePublicKeyCredentialOptions) (2)  
6458  
6459 #dom-publickeycredentialparameters-typeReferenced in:  
6460 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6461 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6462 \* 5.3. Parameters for Credential Generation (dictionary  
6463 PublicKeyCredentialParameters)  
6464  
6465 #dom-publickeycredentialparameters-algReferenced in:  
6466 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6467 [[Create]](origin, options, sameOriginWithAncestors) method  
6468 \* 5.3. Parameters for Credential Generation (dictionary  
6469 PublicKeyCredentialParameters)  
6470  
6471 #dictdef-makepublickeycredentialoptionsReferenced in:  
6472 \* 5.1.1. CredentialCreationOptions Extension  
6473 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6474 [[Create]](origin, options, sameOriginWithAncestors) method  
6475 \* 5.4. Options for Credential Creation (dictionary  
6476 MakePublicKeyCredentialOptions)  
6477  
6478 #dom-makepublickeycredentialoptions-rpReferenced in:  
6479 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6480 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6481 (4) (5) (6)  
6482 \* 5.4. Options for Credential Creation (dictionary  
6483 MakePublicKeyCredentialOptions)  
6484  
6485 #dom-makepublickeycredentialoptions-userReferenced in:  
6486 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6487 [[Create]](origin, options, sameOriginWithAncestors) method  
6488 \* 5.4. Options for Credential Creation (dictionary  
6489 MakePublicKeyCredentialOptions)  
6490 \* 7.1. Registering a new credential  
6491  
6492 #dom-makepublickeycredentialoptions-challengeReferenced in:  
6493 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6494 [[Create]](origin, options, sameOriginWithAncestors) method  
6495 \* 5.4. Options for Credential Creation (dictionary  
6496 MakePublicKeyCredentialOptions)  
6497  
6498 #dom-makepublickeycredentialoptions-pubkeycredparamsReferenced in:

7409 [[DiscoverFromExternalSource]](origin, options,  
7410 sameOriginWithAncestors) method  
7411 \* 5.2.2. Web Authentication Assertion (interface  
7412 AuthenticatorAssertionResponse)  
7413 \* 7.2. Verifying an authentication assertion  
7414  
7415 #dom-authenticatorassertionresponse-signatureReferenced in:  
7416 \* 5.1.4.1. PublicKeyCredential's  
7417 [[DiscoverFromExternalSource]](origin, options,  
7418 sameOriginWithAncestors) method  
7419 \* 5.2.2. Web Authentication Assertion (interface  
7420 AuthenticatorAssertionResponse)  
7421 \* 7.2. Verifying an authentication assertion  
7422  
7423 #dom-authenticatorassertionresponse-userhandleReferenced in:  
7424 \* 2.2.1. Backwards Compatibility with FIDO U2F  
7425 \* 5.1.4.1. PublicKeyCredential's  
7426 [[DiscoverFromExternalSource]](origin, options,  
7427 sameOriginWithAncestors) method  
7428 \* 5.2.2. Web Authentication Assertion (interface  
7429 AuthenticatorAssertionResponse)  
7430 \* 7.2. Verifying an authentication assertion  
7431  
7432 #dictdef-publickeycredentialparametersReferenced in:  
7433 \* 5.3. Parameters for Credential Generation (dictionary  
7434 PublicKeyCredentialParameters)  
7435 \* 5.4. Options for Credential Creation (dictionary  
7436 PublicKeyCredentialCreationOptions) (2)  
7437  
7438 #dom-publickeycredentialparameters-typeReferenced in:  
7439 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7440 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
7441 \* 5.3. Parameters for Credential Generation (dictionary  
7442 PublicKeyCredentialParameters)  
7443  
7444 #dom-publickeycredentialparameters-algReferenced in:  
7445 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7446 [[Create]](origin, options, sameOriginWithAncestors) method  
7447 \* 5.3. Parameters for Credential Generation (dictionary  
7448 PublicKeyCredentialParameters)  
7449  
7450 #dictdef-publickeycredentialcreationoptionsReferenced in:  
7451 \* 5.1.1. CredentialCreationOptions Dictionary Extension  
7452 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7453 [[Create]](origin, options, sameOriginWithAncestors) method  
7454 \* 5.4. Options for Credential Creation (dictionary  
7455 PublicKeyCredentialCreationOptions)  
7456  
7457 #dom-publickeycredentialcreationoptions-rpReferenced in:  
7458 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7459 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
7460 (4) (5) (6)  
7461 \* 5.4. Options for Credential Creation (dictionary  
7462 PublicKeyCredentialCreationOptions)  
7463  
7464 #dom-publickeycredentialcreationoptions-userReferenced in:  
7465 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7466 [[Create]](origin, options, sameOriginWithAncestors) method  
7467 \* 5.4. Options for Credential Creation (dictionary  
7468 PublicKeyCredentialCreationOptions)  
7469 \* 7.1. Registering a new credential  
7470  
7471 #dom-publickeycredentialcreationoptions-challengeReferenced in:  
7472 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7473 [[Create]](origin, options, sameOriginWithAncestors) method  
7474 \* 5.4. Options for Credential Creation (dictionary  
7475 PublicKeyCredentialCreationOptions)  
7476 \* 13.1. Cryptographic Challenges  
7477  
7478 #dom-publickeycredentialcreationoptions-pubkeycredparamsReferenced in:

649f \* 5.1.3. Create a new credential - PublicKeyCredential's  
649f [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6500 \* 5.4. Options for Credential Creation (dictionary  
6500 MakePublicKeyCredentialOptions)  
6502 #dom-makepublickeycredentialoptions-timeoutReferenced in:  
6503 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6504 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6505 \* 5.4. Options for Credential Creation (dictionary  
6506 MakePublicKeyCredentialOptions)  
6507 #dom-makepublickeycredentialoptions-excludecredentialsReferenced in:  
6508  
6509  
6510 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6511 [[Create]](origin, options, sameOriginWithAncestors) method  
6512 \* 5.4. Options for Credential Creation (dictionary  
6513 MakePublicKeyCredentialOptions)  
6514  
6515 #dom-makepublickeycredentialoptions-authenticatorselectionReferenced  
6516 in:  
6517 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6518 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
6519 (4) (5) (6)  
6520 \* 5.4. Options for Credential Creation (dictionary  
6521 MakePublicKeyCredentialOptions)  
6522 \* 6.2.1. The authenticatorMakeCredential operation  
6523  
6524 #dom-makepublickeycredentialoptions-attestationReferenced in:  
6525 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6526 [[Create]](origin, options, sameOriginWithAncestors) method  
6527 \* 5.4. Options for Credential Creation (dictionary  
6528 MakePublicKeyCredentialOptions)  
6529  
6530 #dom-makepublickeycredentialoptions-extensionsReferenced in:  
6531 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6532 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
6533 \* 5.4. Options for Credential Creation (dictionary  
6534 MakePublicKeyCredentialOptions)  
6535  
6536 \* 9.3. Extending request parameters  
6537 #dictdef-publickeycredentialentityReferenced in:  
6538 \* 5.4.1. Public Key Entity Description (dictionary  
6539 PublicKeyCredentialEntity) (2)  
6540 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
6541 PublicKeyCredentialRpEntity)  
6542 \* 5.4.3. User Account Parameters for Credential Generation  
6543 (dictionary PublicKeyCredentialUserEntity)  
6544  
6545 #dom-publickeycredentialentity-nameReferenced in:  
6546 \* 5.4. Options for Credential Creation (dictionary  
6547 MakePublicKeyCredentialOptions) (2)  
6548 \* 5.4.1. Public Key Entity Description (dictionary  
6549 PublicKeyCredentialEntity)  
6550 \* 6.2.1. The authenticatorMakeCredential operation (2)  
6551  
6552 #dom-publickeycredentialentity-iconReferenced in:  
6553 \* 5.4.1. Public Key Entity Description (dictionary  
6554 PublicKeyCredentialEntity)  
6555  
6556 #dictdef-publickeycredentialrpentityReferenced in:  
6557 \* 5.4. Options for Credential Creation (dictionary  
6558 MakePublicKeyCredentialOptions) (2)  
6559  
6560 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
6561 PublicKeyCredentialRpEntity) (2)  
6562 \* 6.2.1. The authenticatorMakeCredential operation

747f \* 5.1.3. Create a new credential - PublicKeyCredential's  
7480 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
7481 \* 5.4. Options for Credential Creation (dictionary  
7482 PublicKeyCredentialCreationOptions)  
7483  
7484 #dom-publickeycredentialcreationoptions-timeoutReferenced in:  
7485 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7486 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
7487 \* 5.4. Options for Credential Creation (dictionary  
7488 PublicKeyCredentialCreationOptions)  
7489  
7490 #dom-publickeycredentialcreationoptions-excludecredentialsReferenced  
7491 in:  
7492 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7493 [[Create]](origin, options, sameOriginWithAncestors) method  
7494 \* 5.4. Options for Credential Creation (dictionary  
7495 PublicKeyCredentialCreationOptions)  
7496 \* 14.2. Registration Ceremony Privacy (2)  
7497  
7498 #dom-publickeycredentialcreationoptions-authenticatorselectionReference  
7499 d in:  
7500 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7501 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
7502 (4) (5) (6)  
7503 \* 5.4. Options for Credential Creation (dictionary  
7504 PublicKeyCredentialCreationOptions)  
7505 \* 6.2.2. The authenticatorMakeCredential operation  
7506  
7507 #dom-publickeycredentialcreationoptions-attestationReferenced in:  
7508 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7509 [[Create]](origin, options, sameOriginWithAncestors) method  
7510 \* 5.4. Options for Credential Creation (dictionary  
7511 PublicKeyCredentialCreationOptions)  
7512  
7513 #dom-publickeycredentialcreationoptions-extensionsReferenced in:  
7514 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7515 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
7516 \* 5.4. Options for Credential Creation (dictionary  
7517 PublicKeyCredentialCreationOptions)  
7518 \* 7.1. Registering a new credential (2)  
7519 \* 7.2. Verifying an authentication assertion  
7520 \* 9.3. Extending request parameters  
7521  
7522 #dictdef-publickeycredentialentityReferenced in:  
7523 \* 5.4.1. Public Key Entity Description (dictionary  
7524 PublicKeyCredentialEntity) (2) (3)  
7525 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
7526 PublicKeyCredentialRpEntity)  
7527 \* 5.4.3. User Account Parameters for Credential Generation  
7528 (dictionary PublicKeyCredentialUserEntity)  
7529  
7530 #dom-publickeycredentialentity-nameReferenced in:  
7531 \* 5.4. Options for Credential Creation (dictionary  
7532 PublicKeyCredentialCreationOptions) (2)  
7533 \* 5.4.1. Public Key Entity Description (dictionary  
7534 PublicKeyCredentialEntity) (2) (3) (4)  
7535 \* 6.2.2. The authenticatorMakeCredential operation (2)  
7536  
7537 #dom-publickeycredentialentity-iconReferenced in:  
7538 \* 5.4.1. Public Key Entity Description (dictionary  
7539 PublicKeyCredentialEntity)  
7540  
7541 #dictdef-publickeycredentialrpentityReferenced in:  
7542 \* 5.4. Options for Credential Creation (dictionary  
7543 PublicKeyCredentialCreationOptions) (2)  
7544 \* 5.4.1. Public Key Entity Description (dictionary  
7545 PublicKeyCredentialEntity)  
7546 \* 5.4.2. RP Parameters for Credential Generation (dictionary  
7547 PublicKeyCredentialRpEntity) (2)  
7548 \* 6.2.2. The authenticatorMakeCredential operation

```

6562 #dom-publickeycredentialrpentity-idReferenced in:
6563
6564 * 5.4. Options for Credential Creation (dictionary
6565 MakePublicKeyCredentialOptions)
6566 * 5.4.2. RP Parameters for Credential Generation (dictionary
6567 PublicKeyCredentialRpEntity)
6568 * 6.2.1. The authenticatorMakeCredential operation (2) (3) (4)
6569
6570 #dictdef-publickeycredentialuserentityReferenced in:
6571 * 5.4. Options for Credential Creation (dictionary
6572 MakePublicKeyCredentialOptions) (2)
6573
6574 * 5.4.3. User Account Parameters for Credential Generation
6575 (dictionary PublicKeyCredentialUserEntity) (2)
6576 * 6.2.1. The authenticatorMakeCredential operation
6577
6578 #dom-publickeycredentialuserentity-idReferenced in:
6579 * 5.4. Options for Credential Creation (dictionary
6580 MakePublicKeyCredentialOptions)
6581 * 5.4.3. User Account Parameters for Credential Generation
6582 (dictionary PublicKeyCredentialUserEntity)
6583 * 6.2.1. The authenticatorMakeCredential operation
6584
6585 #dom-publickeycredentialuserentity-displaynameReferenced in:
6586 * 4. Terminology
6587 * 5.4. Options for Credential Creation (dictionary
6588 MakePublicKeyCredentialOptions)
6589
6590 * 5.4.3. User Account Parameters for Credential Generation
6591 (dictionary PublicKeyCredentialUserEntity)
6592 * 6.2.1. The authenticatorMakeCredential operation
6593
6594 #dictdef-authenticatorselectioncriteriaReferenced in:
6595 * 5.4. Options for Credential Creation (dictionary
6596 MakePublicKeyCredentialOptions) (2)
6597 * 5.4.4. Authenticator Selection Criteria (dictionary
6598 AuthenticatorSelectionCriteria) (2)
6599
6600 #dom-authenticatorselectioncriteria-authenticatorattachmentReferenced
6601 in:
6602 * 5.1.3. Create a new credential - PublicKeyCredential's
6603 \[\[Create\]\]\(origin, options, sameOriginWithAncestors\) method
6604 * 5.4.4. Authenticator Selection Criteria (dictionary
6605 AuthenticatorSelectionCriteria)
6606
6607 #dom-authenticatorselectioncriteria-requireresidentkeyReferenced in:
6608 * 5.1.3. Create a new credential - PublicKeyCredential's
6609 \[\[Create\]\]\(origin, options, sameOriginWithAncestors\) method (2)
6610 * 5.4.4. Authenticator Selection Criteria (dictionary
6611 AuthenticatorSelectionCriteria)
6612 * 6.2.1. The authenticatorMakeCredential operation
6613
6614 #dom-authenticatorselectioncriteria-userverificationReferenced in:
6615 * 5.1.3. Create a new credential - PublicKeyCredential's
6616 \[\[Create\]\]\(origin, options, sameOriginWithAncestors\) method (2)
6617 * 5.4.4. Authenticator Selection Criteria (dictionary
6618 AuthenticatorSelectionCriteria)
6619
6620 #enumdef-authenticatorattachmentReferenced in:
6621 * 5.4.4. Authenticator Selection Criteria (dictionary
6622 AuthenticatorSelectionCriteria) (2)
6623 * 5.4.5. Authenticator Attachment enumeration (enum
6624 AuthenticatorAttachment) (2)

```

```

7549 #dom-publickeycredentialrpentity-idReferenced in:
7550 * 5.1.3. Create a new credential - PublicKeyCredential's
7551 \[\[Create\]\]\(origin, options, sameOriginWithAncestors\) method (2) (3)
7552 (4) (5)
7553 * 5.4. Options for Credential Creation (dictionary
7554 PublicKeyCredentialCreationOptions)
7555 * 5.4.2. RP Parameters for Credential Generation (dictionary
7556 PublicKeyCredentialRpEntity)
7557 * 6.2.2. The authenticatorMakeCredential operation (2) (3) (4)
7558
7559 #dictdef-publickeycredentialuserentityReferenced in:
7560 * 5.4. Options for Credential Creation (dictionary
7561 PublicKeyCredentialCreationOptions) (2)
7562 * 5.4.1. Public Key Entity Description (dictionary
7563 PublicKeyCredentialEntity) (2)
7564 * 5.4.3. User Account Parameters for Credential Generation
7565 (dictionary PublicKeyCredentialUserEntity) (2)
7566 * 6.2.2. The authenticatorMakeCredential operation
7567
7568 #dom-publickeycredentialuserentity-idReferenced in:
7569 * 5.4. Options for Credential Creation (dictionary
7570 PublicKeyCredentialCreationOptions)
7571 * 5.4.3. User Account Parameters for Credential Generation
7572 (dictionary PublicKeyCredentialUserEntity)
7573 * 6.2.2. The authenticatorMakeCredential operation
7574
7575 #dom-publickeycredentialuserentity-displaynameReferenced in:
7576 * 4. Terminology
7577 * 5.4. Options for Credential Creation (dictionary
7578 PublicKeyCredentialCreationOptions)
7579 * 5.4.1. Public Key Entity Description (dictionary
7580 PublicKeyCredentialEntity)
7581 * 5.4.3. User Account Parameters for Credential Generation
7582 (dictionary PublicKeyCredentialUserEntity) (2) (3)
7583 * 6.2.2. The authenticatorMakeCredential operation
7584
7585 #dictdef-authenticatorselectioncriteriaReferenced in:
7586 * 5.4. Options for Credential Creation (dictionary
7587 PublicKeyCredentialCreationOptions) (2)
7588 * 5.4.4. Authenticator Selection Criteria (dictionary
7589 AuthenticatorSelectionCriteria) (2)
7590
7591 #dom-authenticatorselectioncriteria-authenticatorattachmentReferenced
7592 in:
7593 * 5.1.3. Create a new credential - PublicKeyCredential's
7594 \[\[Create\]\]\(origin, options, sameOriginWithAncestors\) method
7595 * 5.4.4. Authenticator Selection Criteria (dictionary
7596 AuthenticatorSelectionCriteria)
7597
7598 #dom-authenticatorselectioncriteria-requireresidentkeyReferenced in:
7599 * 5.1.3. Create a new credential - PublicKeyCredential's
7600 \[\[Create\]\]\(origin, options, sameOriginWithAncestors\) method (2)
7601 * 5.4.4. Authenticator Selection Criteria (dictionary
7602 AuthenticatorSelectionCriteria)
7603 * 6.2.2. The authenticatorMakeCredential operation
7604
7605 #dom-authenticatorselectioncriteria-userverificationReferenced in:
7606 * 5.1.3. Create a new credential - PublicKeyCredential's
7607 \[\[Create\]\]\(origin, options, sameOriginWithAncestors\) method (2)
7608 * 5.4.4. Authenticator Selection Criteria (dictionary
7609 AuthenticatorSelectionCriteria)
7610
7611 #enumdef-authenticatorattachmentReferenced in:
7612 * 5.4.4. Authenticator Selection Criteria (dictionary
7613 AuthenticatorSelectionCriteria) (2)
7614 * 5.4.5. Authenticator Attachment enumeration (enum
7615 AuthenticatorAttachment) (2)
7616
7617 #attachment-modalityReferenced in:
7618

```

6624 #platform-authenticatorsReferenced in:  
 6625 \* 5.1.6. Availability of User-Verifying Platform Authenticator -  
 6626 PublicKeyCredential's  
 6627 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)  
 6628 (5)  
 6629 \* 5.4.5. Authenticator Attachment enumeration (enum  
 6630 AuthenticatorAttachment) (2)  
 6631 \* 12.1. Registration  
 6632 \* 12.2. Registration Specifically with User Verifying Platform  
 6633 Authenticator (2)

6634 #roaming-authenticatorsReferenced in:  
 6635 \* 1.1.3. Other use cases and configurations  
 6636 \* 5.4.5. Authenticator Attachment enumeration (enum  
 6637 AuthenticatorAttachment) (2)  
 6638 \* 12.1. Registration

6641 #platform-attachmentReferenced in:  
 6642 \* 5.4.5. Authenticator Attachment enumeration (enum  
 6643 AuthenticatorAttachment)  
 6644

6645 #cross-platform-attachedReferenced in:  
 6646 \* 5.4.5. Authenticator Attachment enumeration (enum  
 6647 AuthenticatorAttachment) (2)  
 6648

6649 #attestation-conveyanceReferenced in:  
 6650 \* 4. Terminology  
 6651 \* 5.4. Options for Credential Creation (dictionary  
 6652 MakePublicKeyCredentialOptions)  
 6653 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
 6654 AttestationConveyancePreference)  
 6655

6656 #enumdef-attestationconveyancepreferenceReferenced in:  
 6657 \* 5.4. Options for Credential Creation (dictionary  
 6658 MakePublicKeyCredentialOptions) (2)  
 6659 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
 6660 AttestationConveyancePreference) (2)  
 6661

6662 #dom-attestationconveyancepreference-noneReferenced in:  
 6663 \* 5.4. Options for Credential Creation (dictionary  
 6664 MakePublicKeyCredentialOptions)  
 6665 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
 6666 AttestationConveyancePreference)  
 6667

6668 #dom-attestationconveyancepreference-indirectReferenced in:  
 6669 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
 6670 AttestationConveyancePreference)  
 6671

6672 #dom-attestationconveyancepreference-directReferenced in:  
 6673 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
 6674 AttestationConveyancePreference)  
 6675

6676 #dictdef-publickeycredentialrequestoptionsReferenced in:  
 6677 \* 5.1.2. CredentialRequestOptions Dictionary Extension  
 6678 \* 5.1.4.1. PublicKeyCredential's  
 6679 [[DiscoverFromExternalSource]](origin, options,  
 6680 sameOriginWithAncestors) method  
 6681 \* 5.5. Options for Assertion Generation (dictionary

7619 \* 5.4.5. Authenticator Attachment enumeration (enum  
 7620 AuthenticatorAttachment) (2)  
 7621  
 7622 #platform-authenticatorsReferenced in:  
 7623 \* 5.1.7. Availability of User-Verifying Platform Authenticator -  
 7624 PublicKeyCredential's  
 7625 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)  
 7626 (5)  
 7627 \* 5.4.5. Authenticator Attachment enumeration (enum  
 7628 AuthenticatorAttachment) (2)  
 7629 \* 12.1. Registration  
 7630 \* 12.2. Registration Specifically with User Verifying Platform  
 7631 Authenticator (2)  
 7632 \* 14.2. Registration Ceremony Privacy

7633 #roaming-authenticatorsReferenced in:  
 7634 \* 1.1.3. Other use cases and configurations  
 7635 \* 5.4.5. Authenticator Attachment enumeration (enum  
 7636 AuthenticatorAttachment) (2)  
 7637 \* 12.1. Registration

7638 #platform-attachmentReferenced in:  
 7639 \* 5.4.5. Authenticator Attachment enumeration (enum  
 7640 AuthenticatorAttachment)  
 7641

7642 #platform-credentialReferenced in:  
 7643 \* 5.4.5. Authenticator Attachment enumeration (enum  
 7644 AuthenticatorAttachment) (2)  
 7645

7646 #cross-platform-attachedReferenced in:  
 7647 \* 5.4.5. Authenticator Attachment enumeration (enum  
 7648 AuthenticatorAttachment) (2)  
 7649

7650 #roaming-credentialReferenced in:  
 7651 \* 5.4.5. Authenticator Attachment enumeration (enum  
 7652 AuthenticatorAttachment) (2)  
 7653

7654 #attestation-conveyanceReferenced in:  
 7655 \* 4. Terminology  
 7656 \* 5.4. Options for Credential Creation (dictionary  
 7657 PublicKeyCredentialCreationOptions)  
 7658 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
 7659 AttestationConveyancePreference)  
 7660

7661 #enumdef-attestationconveyancepreferenceReferenced in:  
 7662 \* 5.4. Options for Credential Creation (dictionary  
 7663 PublicKeyCredentialCreationOptions) (2)  
 7664 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
 7665 AttestationConveyancePreference) (2)  
 7666

7667 #dom-attestationconveyancepreference-noneReferenced in:  
 7668 \* 5.4. Options for Credential Creation (dictionary  
 7669 PublicKeyCredentialCreationOptions)  
 7670 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
 7671 AttestationConveyancePreference)  
 7672

7673 #dom-attestationconveyancepreference-indirectReferenced in:  
 7674 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
 7675 AttestationConveyancePreference)  
 7676

7677 #dom-attestationconveyancepreference-directReferenced in:  
 7678 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
 7679 AttestationConveyancePreference)  
 7680

7681 #dictdef-publickeycredentialrequestoptionsReferenced in:  
 7682 \* 5.1.2. CredentialRequestOptions Dictionary Extension  
 7683 \* 5.1.4.1. PublicKeyCredential's  
 7684 [[DiscoverFromExternalSource]](origin, options,  
 7685 sameOriginWithAncestors) method  
 7686 \* 5.5. Options for Assertion Generation (dictionary

6682 PublicKeyCredentialRequestOptions) (2)  
6683 \* 7.2. Verifying an authentication assertion  
6684  
6685 #dom-publickeycredentialrequestoptions-challengeReferenced in:  
6686 \* 5.1.4.1. PublicKeyCredential's  
6687 [[DiscoverFromExternalSource]](origin, options,  
6688 sameOriginWithAncestors) method  
6689 \* 5.5. Options for Assertion Generation (dictionary  
6690 PublicKeyCredentialRequestOptions) (2)  
6691 \* 13.1. Cryptographic Challenges  
6692  
6693 #dom-publickeycredentialrequestoptions-timeoutReferenced in:  
6694 \* 5.1.4.1. PublicKeyCredential's  
6695 [[DiscoverFromExternalSource]](origin, options,  
6696 sameOriginWithAncestors) method (2)  
6697 \* 5.5. Options for Assertion Generation (dictionary  
6698 PublicKeyCredentialRequestOptions)  
6699  
6700 #dom-publickeycredentialrequestoptions-rpidReferenced in:  
6701 \* 5.1.4.1. PublicKeyCredential's  
6702 [[DiscoverFromExternalSource]](origin, options,  
6703 sameOriginWithAncestors) method (2) (3) (4)  
6704 \* 5.5. Options for Assertion Generation (dictionary  
6705 PublicKeyCredentialRequestOptions)  
6706 \* 10.1. FIDO AppID Extension (appid)  
6707  
6708 #dom-publickeycredentialrequestoptions-allowcredentialsReferenced in:  
6709 \* 5.1.4.1. PublicKeyCredential's  
6710 [[DiscoverFromExternalSource]](origin, options,  
6711 sameOriginWithAncestors) method (2) (3) (4)  
6712 \* 5.5. Options for Assertion Generation (dictionary  
6713 PublicKeyCredentialRequestOptions)  
6714  
6715 #dom-publickeycredentialrequestoptions-userverificationReferenced in:  
6716 \* 5.1.4.1. PublicKeyCredential's  
6717 [[DiscoverFromExternalSource]](origin, options,  
6718 sameOriginWithAncestors) method (2)  
6719 \* 5.5. Options for Assertion Generation (dictionary  
6720 PublicKeyCredentialRequestOptions)  
6721  
6722 #dom-publickeycredentialrequestoptions-extensionsReferenced in:  
6723 \* 5.1.4.1. PublicKeyCredential's  
6724 [[DiscoverFromExternalSource]](origin, options,  
6725 sameOriginWithAncestors) method (2)  
6726 \* 5.5. Options for Assertion Generation (dictionary  
6727 PublicKeyCredentialRequestOptions)  
6728  
6729 #typedef-authenticationextensionsReferenced in:  
6730  
6731 \* 5.1. PublicKeyCredential Interface  
6732 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6733 [[Create]](origin, options, sameOriginWithAncestors) method  
6734 \* 5.1.4.1. PublicKeyCredential's  
6735 [[DiscoverFromExternalSource]](origin, options,

7689 PublicKeyCredentialRequestOptions) (2)  
7690 \* 7.2. Verifying an authentication assertion  
7691  
7692 #dom-publickeycredentialrequestoptions-challengeReferenced in:  
7693 \* 5.1.4.1. PublicKeyCredential's  
7694 [[DiscoverFromExternalSource]](origin, options,  
7695 sameOriginWithAncestors) method  
7696 \* 5.5. Options for Assertion Generation (dictionary  
7697 PublicKeyCredentialRequestOptions) (2)  
7698 \* 13.1. Cryptographic Challenges  
7699  
7700 #dom-publickeycredentialrequestoptions-timeoutReferenced in:  
7701 \* 5.1.4.1. PublicKeyCredential's  
7702 [[DiscoverFromExternalSource]](origin, options,  
7703 sameOriginWithAncestors) method (2)  
7704 \* 5.5. Options for Assertion Generation (dictionary  
7705 PublicKeyCredentialRequestOptions)  
7706  
7707 #dom-publickeycredentialrequestoptions-rpidReferenced in:  
7708 \* 5.1.4.1. PublicKeyCredential's  
7709 [[DiscoverFromExternalSource]](origin, options,  
7710 sameOriginWithAncestors) method (2) (3) (4)  
7711 \* 5.5. Options for Assertion Generation (dictionary  
7712 PublicKeyCredentialRequestOptions)  
7713  
7714 #dom-publickeycredentialrequestoptions-allowcredentialsReferenced in:  
7715 \* 5.1.4.1. PublicKeyCredential's  
7716 [[DiscoverFromExternalSource]](origin, options,  
7717 sameOriginWithAncestors) method (2) (3) (4)  
7718 \* 5.5. Options for Assertion Generation (dictionary  
7719 PublicKeyCredentialRequestOptions)  
7720 \* 7.2. Verifying an authentication assertion (2)  
7721 \* 14.3. Authentication Ceremony Privacy (2)  
7722  
7723 #dom-publickeycredentialrequestoptions-userverificationReferenced in:  
7724 \* 5.1.4.1. PublicKeyCredential's  
7725 [[DiscoverFromExternalSource]](origin, options,  
7726 sameOriginWithAncestors) method (2)  
7727 \* 5.5. Options for Assertion Generation (dictionary  
7728 PublicKeyCredentialRequestOptions)  
7729  
7730 #dom-publickeycredentialrequestoptions-extensionsReferenced in:  
7731 \* 5.1.4.1. PublicKeyCredential's  
7732 [[DiscoverFromExternalSource]](origin, options,  
7733 sameOriginWithAncestors) method (2)  
7734 \* 5.5. Options for Assertion Generation (dictionary  
7735 PublicKeyCredentialRequestOptions)  
7736 \* 7.2. Verifying an authentication assertion  
7737  
7738 #dictdef-authenticationextensionsclientinputsReferenced in:  
7739 \* 5.4. Options for Credential Creation (dictionary  
7740 PublicKeyCredentialCreationOptions) (2)  
7741 \* 5.5. Options for Assertion Generation (dictionary  
7742 PublicKeyCredentialRequestOptions) (2)  
7743 \* 10.1. FIDO AppID Extension (appid)  
7744 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
7745 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
7746 \* 10.4. Authenticator Selection Extension (authnSel)  
7747 \* 10.5. Supported Extensions Extension (exts)  
7748 \* 10.6. User Verification Index Extension (uvi)  
7749 \* 10.7. Location Extension (loc)  
7750 \* 10.8. User Verification Method Extension (uvm)  
7751  
7752 #dictdef-authenticationextensionsclientoutputsReferenced in:  
7753 \* 5.1. PublicKeyCredential Interface  
7754 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7755 [[Create]](origin, options, sameOriginWithAncestors) method  
7756 \* 5.1.4.1. PublicKeyCredential's  
7757 [[DiscoverFromExternalSource]](origin, options,

6735 sameOriginWithAncestors) method  
 6736 \* 5.4. Options for Credential Creation (dictionary  
 6737 MakePublicKeyCredentialOptions) (2)  
 6738 \* 5.5. Options for Assertion Generation (dictionary  
 6739 PublicKeyCredentialRequestOptions) (2)  
 6740 \* 5.8.1. Client data used in WebAuthn signatures (dictionary  
 6741 CollectedClientData) (2)

6742  
 6743 #dictdef-collectedclientdataReferenced in:  
 6744 \* 5.1.3. Create a new credential - PublicKeyCredential's  
 6745 [[Create]](origin, options, sameOriginWithAncestors) method  
 6746 \* 5.1.4.1. PublicKeyCredential's  
 6747 [[DiscoverFromExternalSource]](origin, options,  
 6748 sameOriginWithAncestors) method  
 6749 \* 5.8.1. Client data used in WebAuthn signatures (dictionary  
 6750 CollectedClientData) (2)  
 6751  
 6752 #client-dataReferenced in:  
 6753 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
 6754 \* 6. WebAuthn Authenticator model (2) (3) (4)  
 6755 \* 6.1. Authenticator data (2)  
 6756 \* 7.1. Registering a new credential  
 6757 \* 7.2. Verifying an authentication assertion  
 6758 \* 9. WebAuthn Extensions  
 6759 \* 9.4. Client extension processing  
 6760 \* 9.6. Example Extension

6761  
 6762 #dom-collectedclientdata-typeReferenced in:  
 6763 \* 5.1.3. Create a new credential - PublicKeyCredential's  
 6764 [[Create]](origin, options, sameOriginWithAncestors) method  
 6765 \* 5.1.4.1. PublicKeyCredential's  
 6766 [[DiscoverFromExternalSource]](origin, options,  
 6767 sameOriginWithAncestors) method  
 6768 \* 5.8.1. Client data used in WebAuthn signatures (dictionary  
 6769 CollectedClientData)

6770 \* 7.1. Registering a new credential  
 6771 \* 7.2. Verifying an authentication assertion

6772  
 6773 #dom-collectedclientdata-challengeReferenced in:  
 6774 \* 5.1.3. Create a new credential - PublicKeyCredential's  
 6775 [[Create]](origin, options, sameOriginWithAncestors) method  
 6776 \* 5.1.4.1. PublicKeyCredential's  
 6777 [[DiscoverFromExternalSource]](origin, options,  
 6778 sameOriginWithAncestors) method  
 6779 \* 5.8.1. Client data used in WebAuthn signatures (dictionary  
 6780 CollectedClientData)  
 6781 \* 7.1. Registering a new credential  
 6782 \* 7.2. Verifying an authentication assertion

6783  
 6784 #dom-collectedclientdata-originReferenced in:  
 6785 \* 5.1.3. Create a new credential - PublicKeyCredential's  
 6786 [[Create]](origin, options, sameOriginWithAncestors) method  
 6787 \* 5.1.4.1. PublicKeyCredential's  
 6788 [[DiscoverFromExternalSource]](origin, options,  
 6789 sameOriginWithAncestors) method  
 6790 \* 5.8.1. Client data used in WebAuthn signatures (dictionary  
 6791 CollectedClientData)  
 6792 \* 7.1. Registering a new credential  
 6793 \* 7.2. Verifying an authentication assertion

6794  
 6795 #dom-collectedclientdata-hashAlgorithmReferenced in:  
 6796 \* 5.1.3. Create a new credential - PublicKeyCredential's  
 6797 [[Create]](origin, options, sameOriginWithAncestors) method  
 6798 \* 5.1.4.1. PublicKeyCredential's

7758 sameOriginWithAncestors) method  
 7759 \* 7.1. Registering a new credential  
 7760 \* 7.2. Verifying an authentication assertion  
 7761 \* 10.1. FIDO AppID Extension (appid)  
 7762 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
 7763 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
 7764 \* 10.4. Authenticator Selection Extension (authnSel)  
 7765 \* 10.5. Supported Extensions Extension (exts)  
 7766 \* 10.6. User Verification Index Extension (uvi)  
 7767 \* 10.7. Location Extension (loc)  
 7768 \* 10.8. User Verification Method Extension (uvm)

7769  
 7770 #dictdef-collectedclientdataReferenced in:  
 7771 \* 5.1.3. Create a new credential - PublicKeyCredential's  
 7772 [[Create]](origin, options, sameOriginWithAncestors) method  
 7773 \* 5.1.4.1. PublicKeyCredential's  
 7774 [[DiscoverFromExternalSource]](origin, options,  
 7775 sameOriginWithAncestors) method  
 7776 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
 7777 CollectedClientData) (2)

7778  
 7779 #client-dataReferenced in:  
 7780 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
 7781 \* 6. WebAuthn Authenticator Model (2) (3) (4)  
 7782 \* 6.1. Authenticator data (2)  
 7783 \* 7.1. Registering a new credential  
 7784 \* 7.2. Verifying an authentication assertion  
 7785 \* 9. WebAuthn Extensions

7786  
 7787 #dictdef-tokenbindingReferenced in:  
 7788 \* 5.10.1. Client data used in WebAuthn signatures (dictionary

7789 CollectedClientData)

7790  
 7791 #dom-tokenbinding-statusReferenced in:  
 7792 \* 7.1. Registering a new credential  
 7793 \* 7.2. Verifying an authentication assertion

7794  
 7795 #dom-tokenbinding-idReferenced in:

7796 \* 7.1. Registering a new credential  
 7797 \* 7.2. Verifying an authentication assertion

7798  
 7799 #enumdef-tokenbindingstatusReferenced in:  
 7800 \* 5.10.1. Client data used in WebAuthn signatures (dictionary

7801 CollectedClientData)

7802  
 7803 #dom-collectedclientdata-typeReferenced in:  
 7804 \* 5.1.3. Create a new credential - PublicKeyCredential's  
 7805 [[Create]](origin, options, sameOriginWithAncestors) method  
 7806 \* 5.1.4.1. PublicKeyCredential's

679f [[DiscoverFromExternalSource]](origin, options,  
6800 sameOriginWithAncestors) method  
6801 \* 5.8.1. Client data used in WebAuthn signatures (dictionary  
6802 CollectedClientData) (2)  
6803 \* 7.1. Registering a new credential  
6804 \* 7.2. Verifying an authentication assertion  
6805  
6806 #dom-collectedclientdata-tokenbindingidReferenced in:  
6807 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6808 [[Create]](origin, options, sameOriginWithAncestors) method  
6809 \* 5.1.4.1. PublicKeyCredential's  
6810 [[DiscoverFromExternalSource]](origin, options,  
6811 sameOriginWithAncestors) method  
6812 \* 5.8.1. Client data used in WebAuthn signatures (dictionary  
6813 CollectedClientData)  
6814 \* 7.1. Registering a new credential  
6815 \* 7.2. Verifying an authentication assertion  
6816  
6817 #dom-collectedclientdata-clientextensionsReferenced in:  
6818 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6819 [[Create]](origin, options, sameOriginWithAncestors) method  
6820 \* 5.1.4.1. PublicKeyCredential's  
6821 [[DiscoverFromExternalSource]](origin, options,  
6822 sameOriginWithAncestors) method  
6823 \* 5.8.1. Client data used in WebAuthn signatures (dictionary  
6824 CollectedClientData)  
6825 \* 7.1. Registering a new credential  
6826 \* 7.2. Verifying an authentication assertion  
6827 \* 9.4. Client extension processing  
6828  
6829 #dom-collectedclientdata-authenticatorextensionsReferenced in:  
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 \* 5.8.1. Client data used in WebAuthn signatures (dictionary  
6836 CollectedClientData)  
6837 \* 7.1. Registering a new credential  
6838 \* 7.2. Verifying an authentication assertion  
6839  
6840 #collectedclientdata-json-serialized-client-dataReferenced in:  
6841 \* 5.1.3. Create a new credential - PublicKeyCredential's  
6842 [[Create]](origin, options, sameOriginWithAncestors) method  
6843 \* 5.1.4.1. PublicKeyCredential's  
6844 [[DiscoverFromExternalSource]](origin, options,  
6845 sameOriginWithAncestors) method  
6846 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
6847 \* 5.2.1. Information about Public Key Credential (interface  
6848 AuthenticatorAttestationResponse) (2)  
6849 \* 5.2.2. Web Authentication Assertion (interface  
6850 AuthenticatorAssertionResponse)  
6851 \* 5.8.1. Client data used in WebAuthn signatures (dictionary  
6852 CollectedClientData)  
6853  
6854 #collectedclientdata-hash-of-the-serialized-client-dataReferenced 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 \* 5.2.1. Information about Public Key Credential (interface  
6861 AuthenticatorAttestationResponse)  
6862 \* 5.2.2. Web Authentication Assertion (interface  
6863 AuthenticatorAssertionResponse)  
6864 \* 5.8.1. Client data used in WebAuthn signatures (dictionary  
6865 CollectedClientData)  
6866 \* 6. WebAuthn Authenticator model  
6867 \* 6.2.1. The authenticatorMakeCredential operation  
6868 \* 6.2.2. The authenticatorGetAssertion operation (2)

7807 [[DiscoverFromExternalSource]](origin, options,  
7808 sameOriginWithAncestors) method  
7809 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
7810 CollectedClientData)  
7811 \* 7.1. Registering a new credential  
7812 \* 7.2. Verifying an authentication assertion  
7813  
7814 #dom-collectedclientdata-challengeReferenced in:  
7815 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7816 [[Create]](origin, options, sameOriginWithAncestors) method  
7817 \* 5.1.4.1. PublicKeyCredential's  
7818 [[DiscoverFromExternalSource]](origin, options,  
7819 sameOriginWithAncestors) method  
7820 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
7821 CollectedClientData)  
7822 \* 7.1. Registering a new credential  
7823 \* 7.2. Verifying an authentication assertion  
7824  
7825 #dom-collectedclientdata-originReferenced in:  
7826 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7827 [[Create]](origin, options, sameOriginWithAncestors) method  
7828 \* 5.1.4.1. PublicKeyCredential's  
7829 [[DiscoverFromExternalSource]](origin, options,  
7830 sameOriginWithAncestors) method  
7831 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
7832 CollectedClientData)  
7833 \* 7.1. Registering a new credential  
7834 \* 7.2. Verifying an authentication assertion  
7835  
7836 #dom-collectedclientdata-tokenbindingReferenced in:  
7837 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7838 [[Create]](origin, options, sameOriginWithAncestors) method  
7839 \* 5.1.4.1. PublicKeyCredential's  
7840 [[DiscoverFromExternalSource]](origin, options,  
7841 sameOriginWithAncestors) method  
7842 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
7843 CollectedClientData)  
7844 \* 7.1. Registering a new credential (2)  
7845 \* 7.2. Verifying an authentication assertion (2)  
7846  
7847 #collectedclientdata-json-serialized-client-dataReferenced in:  
7848 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7849 [[Create]](origin, options, sameOriginWithAncestors) method  
7850 \* 5.1.4.1. PublicKeyCredential's  
7851 [[DiscoverFromExternalSource]](origin, options,  
7852 sameOriginWithAncestors) method  
7853 \* 5.2. Authenticator Responses (interface AuthenticatorResponse)  
7854 \* 5.2.1. Information about Public Key Credential (interface  
7855 AuthenticatorAttestationResponse) (2)  
7856 \* 5.2.2. Web Authentication Assertion (interface  
7857 AuthenticatorAssertionResponse)  
7858 \* 5.10.1. Client data used in WebAuthn signatures (dictionary  
7859 CollectedClientData)  
7860  
7861 #collectedclientdata-hash-of-the-serialized-client-dataReferenced in:  
7862 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7863 [[Create]](origin, options, sameOriginWithAncestors) method  
7864 \* 5.1.4.1. PublicKeyCredential's  
7865 [[DiscoverFromExternalSource]](origin, options,  
7866 sameOriginWithAncestors) method  
7867 \* 5.2.1. Information about Public Key Credential (interface  
7868 AuthenticatorAttestationResponse)  
7869 \* 5.2.2. Web Authentication Assertion (interface  
7870 AuthenticatorAssertionResponse)  
7871 \* 6. WebAuthn Authenticator Model  
7872 \* 6.2.2. The authenticatorMakeCredential operation  
7873 \* 6.2.3. The authenticatorGetAssertion operation (2)

```

686E * 6.3.2. Attestation Statement Formats (2)
6870 * 6.3.4. Generating an Attestation Object
6871 * 7.1. Registering a new credential
6872 * 8.2. Packed Attestation Statement Format
6873 * 8.3. TPM Attestation Statement Format
6874 * 8.4. Android Key Attestation Statement Format
6875 * 8.5. Android SafetyNet Attestation Statement Format
6876 * 8.6. FIDO U2F Attestation Statement Format
6877
687E #enumdef-publickeycredentialtypeReferenced in:
687F
6880 * 5.1.3. Create a new credential - PublicKeyCredential's
6881 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6882 * 5.3. Parameters for Credential Generation (dictionary
6883 PublicKeyCredentialParameters)
6884 * 5.8.2. Credential Type enumeration (enum PublicKeyCredentialType)
6885 * 5.8.3. Credential Descriptor (dictionary
6886 PublicKeyCredentialDescriptor)
6887 * 6.2.1. The authenticatorMakeCredential operation (2) (3)
6888
688E #dom-publickeycredentialtype-public-keyReferenced in:
688F * 5.8.2. Credential Type enumeration (enum PublicKeyCredentialType)
6890
689C #dictdef-publickeycredentialdescriptorReferenced in:
689D * 5.1.4.1. PublicKeyCredential's
689E [[DiscoverFromExternalSource]](origin, options,
689F sameOriginWithAncestors) method
68A0 * 5.4. Options for Credential Creation (dictionary
68A1 MakePublicKeyCredentialOptions) (2)
68A2 * 5.5. Options for Assertion Generation (dictionary
68A3 PublicKeyCredentialRequestOptions) (2) (3)
68A4 * 5.8.3. Credential Descriptor (dictionary
68A5 PublicKeyCredentialDescriptor)
68A6 * 6.2.1. The authenticatorMakeCredential operation
68A7 * 6.2.2. The authenticatorGetAssertion operation
68A8
68AC #dom-publickeycredentialdescriptor-transportReferenced in:
68AD * 5.1.3. Create a new credential - PublicKeyCredential's
68AE [[Create]](origin, options, sameOriginWithAncestors) method (2)
68AF * 5.1.4.1. PublicKeyCredential's
68B0 [[DiscoverFromExternalSource]](origin, options,
68B1 sameOriginWithAncestors) method (2)
68B2
68B4 #dom-publickeycredentialdescriptor-typeReferenced in:
68B5 * 5.1.4.1. PublicKeyCredential's
68B6 [[DiscoverFromExternalSource]](origin, options,
68B7 sameOriginWithAncestors) method
68B8 * 5.8.3. Credential Descriptor (dictionary
68B9 PublicKeyCredentialDescriptor)
68BA * 6.2.1. The authenticatorMakeCredential operation
68BB * 6.2.2. The authenticatorGetAssertion operation
68BC
68BE #dom-publickeycredentialdescriptor-idReferenced in:
68BF * 5.1.4.1. PublicKeyCredential's
68C0 [[DiscoverFromExternalSource]](origin, options,
68C1 sameOriginWithAncestors) method (2)
68C2 * 5.8.3. Credential Descriptor (dictionary
68C3 PublicKeyCredentialDescriptor)
68C4 * 6.2.1. The authenticatorMakeCredential operation
68C5 * 6.2.2. The authenticatorGetAssertion operation
68C6
68C8 #enumdef-authenticatortransportReferenced in:
68C9 * 5.8.3. Credential Descriptor (dictionary
68CA PublicKeyCredentialDescriptor)
68CB * 5.8.4. Authenticator Transport enumeration (enum
68CC AuthenticatorTransport)
68CD
68CF #dom-authenticatortransport-usbReferenced in:

```

```

7874 * 6.3.2. Attestation Statement Formats (2)
7875 * 6.3.4. Generating an Attestation Object
7876 * 7.1. Registering a new credential
7877 * 8.2. Packed Attestation Statement Format
7878 * 8.3. TPM Attestation Statement Format
7879 * 8.4. Android Key Attestation Statement Format
7880 * 8.5. Android SafetyNet Attestation Statement Format
7881 * 8.6. FIDO U2F Attestation Statement Format
7882
7884 #enumdef-publickeycredentialtypeReferenced in:
7885 * 4. Terminology
7886 * 5.1.3. Create a new credential - PublicKeyCredential's
7887 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7888 * 5.3. Parameters for Credential Generation (dictionary
7889 PublicKeyCredentialParameters)
7890 * 5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)
7891 * 5.10.3. Credential Descriptor (dictionary
7892 PublicKeyCredentialDescriptor)
7893 * 6.2.2. The authenticatorMakeCredential operation (2) (3)
7894
7896 #dom-publickeycredentialtype-public-keyReferenced in:
7897 * 4. Terminology
7898 * 5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)
7899 * 6.2.2. The authenticatorMakeCredential operation
7900
7902 #dictdef-publickeycredentialdescriptorReferenced in:
7903 * 5.1.4.1. PublicKeyCredential's
7904 [[DiscoverFromExternalSource]](origin, options,
7905 sameOriginWithAncestors) method
7906 * 5.4. Options for Credential Creation (dictionary
7907 PublicKeyCredentialCreationOptions) (2)
7908 * 5.5. Options for Assertion Generation (dictionary
7909 PublicKeyCredentialRequestOptions) (2) (3)
7910 * 5.10.3. Credential Descriptor (dictionary
7911 PublicKeyCredentialDescriptor)
7912 * 6.2.2. The authenticatorMakeCredential operation
7913 * 6.2.3. The authenticatorGetAssertion operation
7914
7916 #dom-publickeycredentialdescriptor-transportReferenced in:
7917 * 5.1.3. Create a new credential - PublicKeyCredential's
7918 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7919 * 5.1.4.1. PublicKeyCredential's
7920 [[DiscoverFromExternalSource]](origin, options,
7921 sameOriginWithAncestors) method (2)
7922
7924 #dom-publickeycredentialdescriptor-typeReferenced in:
7925 * 5.1.4.1. PublicKeyCredential's
7926 [[DiscoverFromExternalSource]](origin, options,
7927 sameOriginWithAncestors) method
7928 * 5.10.3. Credential Descriptor (dictionary
7929 PublicKeyCredentialDescriptor)
7930 * 6.2.2. The authenticatorMakeCredential operation
7931 * 6.2.3. The authenticatorGetAssertion operation
7932
7934 #dom-publickeycredentialdescriptor-idReferenced in:
7935 * 5.1.4.1. PublicKeyCredential's
7936 [[DiscoverFromExternalSource]](origin, options,
7937 sameOriginWithAncestors) method (2)
7938 * 5.10.3. Credential Descriptor (dictionary
7939 PublicKeyCredentialDescriptor)
7940 * 6.2.2. The authenticatorMakeCredential operation
7941 * 6.2.3. The authenticatorGetAssertion operation
7942
7944 #enumdef-authenticatortransportReferenced in:
7945 * 5.10.3. Credential Descriptor (dictionary
7946 PublicKeyCredentialDescriptor)
7947 * 5.10.4. Authenticator Transport enumeration (enum
7948 AuthenticatorTransport)
7949
7951 #dom-authenticatortransport-usbReferenced in:

```

```

6936 * 5.8.4. Authenticator Transport enumeration (enum
6937 AuthenticatorTransport)
6938
6939 #dom-authenticatortransport-nfcReferenced in:
6940 * 5.8.4. Authenticator Transport enumeration (enum
6941 AuthenticatorTransport)
6942
6943 #dom-authenticatortransport-bleReferenced in:
6944 * 5.8.4. Authenticator Transport enumeration (enum
6945 AuthenticatorTransport)
6946
6947 #typedefdef-cosealgorithmidentifierReferenced in:
6948 * 5.1.3. Create a new credential - PublicKeyCredential's
6949 [[Create]](origin, options, sameOriginWithAncestors) method
6950 * 5.3. Parameters for Credential Generation (dictionary
6951 PublicKeyCredentialParameters)
6952 * 5.8.5. Cryptographic Algorithm Identifier (typedef
6953 COSEAlgorithmIdentifier)
6954 * 6.2.1. The authenticatorMakeCredential operation
6955 * 6.3.1. Attested credential data
6956 * 8.2. Packed Attestation Statement Format
6957 * 8.3. TPM Attestation Statement Format
6958
6959 #enumdef-userverificationrequirementReferenced in:
6960 * 5.4.4. Authenticator Selection Criteria (dictionary
6961 AuthenticatorSelectionCriteria) (2)
6962 * 5.5. Options for Assertion Generation (dictionary
6963 PublicKeyCredentialRequestOptions) (2)
6964 * 5.8.6. User Verification Requirement enumeration (enum
6965 UserVerificationRequirement)
6966
6967 #dom-userverificationrequirement-requiredReferenced in:
6968 * 5.1.3. Create a new credential - PublicKeyCredential's
6969 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6970 * 5.1.4.1. PublicKeyCredential's
6971 [[DiscoverFromExternalSource]](origin, options,
6972 sameOriginWithAncestors) method (2)
6973 * 5.8.6. User Verification Requirement enumeration (enum
6974 UserVerificationRequirement)
6975
6976 #dom-userverificationrequirement-preferredReferenced in:
6977 * 5.1.3. Create a new credential - PublicKeyCredential's
6978 [[Create]](origin, options, sameOriginWithAncestors) method
6979 * 5.1.4.1. PublicKeyCredential's
6980 [[DiscoverFromExternalSource]](origin, options,
6981 sameOriginWithAncestors) method
6982 * 5.8.6. User Verification Requirement enumeration (enum
6983 UserVerificationRequirement)
6984
6985 #dom-userverificationrequirement-discouragedReferenced in:
6986 * 5.1.3. Create a new credential - PublicKeyCredential's
6987 [[Create]](origin, options, sameOriginWithAncestors) method
6988 * 5.1.4.1. PublicKeyCredential's
6989 [[DiscoverFromExternalSource]](origin, options,
6990 sameOriginWithAncestors) method
6991 * 5.8.6. User Verification Requirement enumeration (enum
6992 UserVerificationRequirement)
6993
6994 #attestation-signatureReferenced in:
6995 * 4. Terminology
6996 * 6. WebAuthn Authenticator Model (2) (3)
6997 * 6.3. Attestation

```

```

7943 * 5.10.4. Authenticator Transport enumeration (enum
7944 AuthenticatorTransport)
7945
7946 #dom-authenticatortransport-nfcReferenced in:
7947 * 5.10.4. Authenticator Transport enumeration (enum
7948 AuthenticatorTransport)
7949
7950 #dom-authenticatortransport-bleReferenced in:
7951 * 5.10.4. Authenticator Transport enumeration (enum
7952 AuthenticatorTransport)
7953
7954 #typedefdef-cosealgorithmidentifierReferenced in:
7955 * 5.1.3. Create a new credential - PublicKeyCredential's
7956 [[Create]](origin, options, sameOriginWithAncestors) method
7957 * 5.3. Parameters for Credential Generation (dictionary
7958 PublicKeyCredentialParameters)
7959 * 5.10.5. Cryptographic Algorithm Identifier (typedef
7960 COSEAlgorithmIdentifier)
7961 * 6.2.2. The authenticatorMakeCredential operation
7962 * 6.3.1. Attested credential data
7963 * 8.2. Packed Attestation Statement Format
7964 * 8.3. TPM Attestation Statement Format
7965
7966 #enumdef-userverificationrequirementReferenced in:
7967 * 5.4.4. Authenticator Selection Criteria (dictionary
7968 AuthenticatorSelectionCriteria) (2)
7969 * 5.5. Options for Assertion Generation (dictionary
7970 PublicKeyCredentialRequestOptions) (2)
7971 * 5.10.6. User Verification Requirement enumeration (enum
7972 UserVerificationRequirement)
7973
7974 #dom-userverificationrequirement-requiredReferenced in:
7975 * 5.1.3. Create a new credential - PublicKeyCredential's
7976 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7977 * 5.1.4.1. PublicKeyCredential's
7978 [[DiscoverFromExternalSource]](origin, options,
7979 sameOriginWithAncestors) method (2)
7980 * 5.10.6. User Verification Requirement enumeration (enum
7981 UserVerificationRequirement)
7982
7983 #dom-userverificationrequirement-preferredReferenced in:
7984 * 5.1.3. Create a new credential - PublicKeyCredential's
7985 [[Create]](origin, options, sameOriginWithAncestors) method
7986 * 5.1.4.1. PublicKeyCredential's
7987 [[DiscoverFromExternalSource]](origin, options,
7988 sameOriginWithAncestors) method
7989 * 5.10.6. User Verification Requirement enumeration (enum
7990 UserVerificationRequirement)
7991
7992 #dom-userverificationrequirement-discouragedReferenced in:
7993 * 5.1.3. Create a new credential - PublicKeyCredential's
7994 [[Create]](origin, options, sameOriginWithAncestors) method
7995 * 5.1.4.1. PublicKeyCredential's
7996 [[DiscoverFromExternalSource]](origin, options,
7997 sameOriginWithAncestors) method
7998 * 5.10.6. User Verification Requirement enumeration (enum
7999 UserVerificationRequirement)
8000
8001 #authenticator-modelReferenced in:
8002 * 6. WebAuthn Authenticator Model
8003
8004 #authenticator-credentials-mapReferenced in:
8005 * 6.2.1. Lookup Credential Source by Credential ID algorithm
8006 * 6.2.2. The authenticatorMakeCredential operation
8007 * 6.2.3. The authenticatorGetAssertion operation
8008
8009 #attestation-signatureReferenced in:
8010 * 4. Terminology
8011 * 6. WebAuthn Authenticator Model (2) (3)
8012 * 6.3. Attestation

```

699f \* 7.1. Registering a new credential  
699f \* 8.6. FIDO U2F Attestation Statement Format  
700c  
7001 #assertion-signatureReferenced in:  
7002 \* 6. WebAuthn Authenticator model (2)  
7003 \* 6.2.2. The authenticatorGetAssertion operation (2) (3)  
7004  
7005 #authenticator-dataReferenced in:  
7006 \* 5.1.4.1. PublicKeyCredential's  
7007 [[DiscoverFromExternalSource]](origin, options,  
7008 sameOriginWithAncestors) method  
7009 \* 5.2.1. Information about Public Key Credential (interface  
7010 AuthenticatorAttestationResponse) (2)  
7011 \* 5.2.2. Web Authentication Assertion (interface  
7012 AuthenticatorAssertionResponse)  
7013 \* 6. WebAuthn Authenticator model (2)  
7014 \* 6.1. Authenticator data (2) (3) (4) (5) (6) (7) (8) (9)  
7015 \* 6.1.1. Signature Counter Considerations (2)  
7016 \* 6.2.1. The authenticatorMakeCredential operation  
7017 \* 6.2.2. The authenticatorGetAssertion operation  
7018 \* 6.3. Attestation (2)  
7019 \* 6.3.1. Attested credential data  
7020 \* 6.3.2. Attestation Statement Formats (2)  
7021 \* 6.3.4. Generating an Attestation Object  
7022 \* 6.3.5.3. Attestation Certificate Hierarchy  
7023 \* 7.1. Registering a new credential  
7024 \* 8.5. Android SafetyNet Attestation Statement Format  
7025 \* 9.5. Authenticator extension processing  
7026 \* 9.6. Example Extension (2)  
7027 \* 10.6. User Verification Index Extension (uvi)  
7028 \* 10.7. Location Extension (loc)  
7029 \* 10.8. User Verification Method Extension (uvm)  
  
703c  
7031 #rpidhashReferenced in:  
7032 \* 7.2. Verifying an authentication assertion  
7033  
7034 #flagsReferenced in:  
7035 \* 5.8.6. User Verification Requirement enumeration (enum  
7036 UserVerificationRequirement) (2)  
7037 \* 6.1. Authenticator data  
  
703e  
703e #signcountReferenced in:  
7040 \* 6.1.1. Signature Counter Considerations (2)  
7041 \* 7.2. Verifying an authentication assertion (2) (3)  
7042  
7043 #attestedcredentialdataReferenced in:  
7044 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7045 [[Create]](origin, options, sameOriginWithAncestors) method  
7046 \* 6.1. Authenticator data (2)  
7047 \* 6.2.1. The authenticatorMakeCredential operation  
7048 \* 6.2.2. The authenticatorGetAssertion operation  
7049 \* 7.1. Registering a new credential (2)  
7050 \* 8.3. TPM Attestation Statement Format  
7051 \* 8.4. Android Key Attestation Statement Format  
7052 \* 8.6. FIDO U2F Attestation Statement Format  
7053  
7054 #authdataextensionsReferenced in:  
7055 \* 6.1. Authenticator data  
7056 \* 6.2.1. The authenticatorMakeCredential operation  
7057 \* 6.2.2. The authenticatorGetAssertion operation  
  
705e  
705e #signature-counterReferenced in:  
706c \* 6.1. Authenticator data

8013 \* 7.1. Registering a new credential  
8014 \* 8.6. FIDO U2F Attestation Statement Format  
801e  
801e #assertion-signatureReferenced in:  
8017 \* 6. WebAuthn Authenticator Model (2)  
8018 \* 6.2.3. The authenticatorGetAssertion operation (2) (3)  
8019  
8020 #authenticator-dataReferenced in:  
8021 \* 5.1.4.1. PublicKeyCredential's  
8022 [[DiscoverFromExternalSource]](origin, options,  
8023 sameOriginWithAncestors) method  
8024 \* 5.2.1. Information about Public Key Credential (interface  
8025 AuthenticatorAttestationResponse) (2)  
8026 \* 5.2.2. Web Authentication Assertion (interface  
8027 AuthenticatorAssertionResponse)  
8028 \* 6. WebAuthn Authenticator Model (2)  
8029 \* 6.1. Authenticator data (2) (3) (4) (5) (6) (7) (8) (9)  
8030 \* 6.1.1. Signature Counter Considerations (2)  
8031 \* 6.2.2. The authenticatorMakeCredential operation  
8032 \* 6.2.3. The authenticatorGetAssertion operation  
8033 \* 6.3. Attestation (2)  
8034 \* 6.3.1. Attested credential data  
8035 \* 6.3.2. Attestation Statement Formats (2)  
8036 \* 6.3.4. Generating an Attestation Object  
  
8037 \* 7.1. Registering a new credential  
8038 \* 8.5. Android SafetyNet Attestation Statement Format  
8039 \* 9.5. Authenticator extension processing (2)  
  
804c  
804c \* 10.6. User Verification Index Extension (uvi)  
  
8041 \* 10.8. User Verification Method Extension (uvm)  
8042 \* 13.2.1. Attestation Certificate Hierarchy  
8043 \* 13.3. credentialId Unsigned  
8044  
804e #rpidhashReferenced in:  
804e \* 7.2. Verifying an authentication assertion  
804f  
804e #flagsReferenced in:  
804e \* 5.10.6. User Verification Requirement enumeration (enum  
804e UserVerificationRequirement) (2)  
8051 \* 6.1. Authenticator data  
8052 \* 7.1. Registering a new credential (2)  
8053 \* 7.2. Verifying an authentication assertion (2)  
8054  
805e #signcountReferenced in:  
805e \* 6.1.1. Signature Counter Considerations (2)  
8057 \* 7.2. Verifying an authentication assertion (2) (3)  
8058  
805e #attestedcredentialdataReferenced in:  
805e \* 5.1.3. Create a new credential - PublicKeyCredential's  
8060 [[Create]](origin, options, sameOriginWithAncestors) method  
8061 \* 6.1. Authenticator data (2)  
8062 \* 6.2.2. The authenticatorMakeCredential operation  
8063 \* 6.2.3. The authenticatorGetAssertion operation  
8064 \* 7.1. Registering a new credential (2)  
8065 \* 8.3. TPM Attestation Statement Format  
8066 \* 8.4. Android Key Attestation Statement Format  
8067 \* 8.6. FIDO U2F Attestation Statement Format  
8068  
806e #authdataextensionsReferenced in:  
8071 \* 6.1. Authenticator data  
8072 \* 6.2.2. The authenticatorMakeCredential operation  
8073 \* 6.2.3. The authenticatorGetAssertion operation  
8074 \* 7.1. Registering a new credential (2)  
8075 \* 7.2. Verifying an authentication assertion (2)  
8076 \* 9.5. Authenticator extension processing (2)  
8077  
807e #signature-counterReferenced in:  
807e \* 6.1. Authenticator data

7061 \* 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6) (7) (8)  
7062 (9) (10)  
7063 \* 6.2.1. The authenticatorMakeCredential operation (2) (3) (4)  
7064 \* 6.2.2. The authenticatorGetAssertion operation (2)  
7065 \* 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6)  
7066

7067 #authenticator-sessionReferenced in:  
7068 \* 5.6. Abort operations with AbortSignal (2)  
7069 \* 6.2.1. The authenticatorMakeCredential operation  
7070 \* 6.2.2. The authenticatorGetAssertion operation  
7071 \* 6.2.3. The authenticatorCancel operation (2)

7072 #authenticatormakecredentialReferenced in:  
7073 \* 4. Terminology (2) (3) (4)  
7074 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7075 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
7076 \* 6. WebAuthn Authenticator model  
7077 \* 6.2.3. The authenticatorCancel operation (2)  
7078 \* 9. WebAuthn Extensions  
7079 \* 9.2. Defining extensions  
7080

7081 #authenticatorgetassertionReferenced in:  
7082 \* 4. Terminology (2) (3)  
7083 \* 5.1.4.1. PublicKeyCredential's  
7084 [[DiscoverFromExternalSource]](origin, options,  
7085 sameOriginWithAncestors) method (2) (3) (4)  
7086 \* 6. WebAuthn Authenticator model  
7087 \* 6.1. Authenticator data  
7088 \* 6.1.1. Signature Counter Considerations (2) (3)  
7089 \* 6.2.3. The authenticatorCancel operation (2)  
7090 \* 9. WebAuthn Extensions  
7091 \* 9.2. Defining extensions  
7092

7093 #authenticatorcancelReferenced in:  
7094 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7095 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
7096 (4)  
7097 \* 5.1.4.1. PublicKeyCredential's  
7098 [[DiscoverFromExternalSource]](origin, options,  
7099 sameOriginWithAncestors) method (2) (3) (4)  
7100 \* 6.2.1. The authenticatorMakeCredential operation  
7101 \* 6.2.2. The authenticatorGetAssertion operation  
7102

7103 #attestation-objectReferenced in:  
7104 \* 4. Terminology (2) (3)  
7105 \* 5. Web Authentication API  
7106 \* 5.2.1. Information about Public Key Credential (interface  
7107 AuthenticatorAttestationResponse) (2)  
7108 \* 5.4. Options for Credential Creation (dictionary  
7109 MakePublicKeyCredentialOptions) (2)  
7110 \* 6.2.1. The authenticatorMakeCredential operation (2)  
7111 \* 6.3. Attestation (2) (3)  
7112 \* 6.3.1. Attested credential data  
7113 \* 6.3.4. Generating an Attestation Object (2)  
7114 \* 7.1. Registering a new credential  
7115

7116 #attestation-statementReferenced in:  
7117 \* 4. Terminology (2)  
7118 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7119 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
7120

8080 \* 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6) (7) (8)  
8081 (9) (10)  
8082 \* 6.2.2. The authenticatorMakeCredential operation (2) (3) (4)  
8083 \* 6.2.3. The authenticatorGetAssertion operation (2)  
8084 \* 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6)  
8085

8086 #authenticator-operationsReferenced in:  
8087 \* 4. Terminology  
8088

8089 #authenticator-sessionReferenced in:  
8090 \* 5.6. Abort operations with AbortSignal (2)  
8091 \* 6.2.2. The authenticatorMakeCredential operation  
8092 \* 6.2.3. The authenticatorGetAssertion operation  
8093 \* 6.2.4. The authenticatorCancel operation (2)  
8094

8095 #credential-id-looking-upReferenced in:  
8096 \* 6.2.2. The authenticatorMakeCredential operation  
8097 \* 6.2.3. The authenticatorGetAssertion operation  
8098

8099 #authenticatormakecredentialReferenced in:  
8100 \* 4. Terminology (2) (3) (4)  
8101 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8102 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
8103 \* 6. WebAuthn Authenticator Model  
8104 \* 6.2.4. The authenticatorCancel operation (2)  
8105 \* 9. WebAuthn Extensions  
8106 \* 9.2. Defining extensions  
8107 \* 9.5. Authenticator extension processing  
8108

8109 #authenticatorgetassertionReferenced in:  
8110 \* 4. Terminology (2) (3)  
8111 \* 5.1.4.1. PublicKeyCredential's  
8112 [[DiscoverFromExternalSource]](origin, options,  
8113 sameOriginWithAncestors) method (2) (3) (4)  
8114 \* 6. WebAuthn Authenticator Model  
8115 \* 6.1. Authenticator data  
8116 \* 6.1.1. Signature Counter Considerations (2) (3)  
8117 \* 6.2.4. The authenticatorCancel operation (2)  
8118 \* 9. WebAuthn Extensions  
8119 \* 9.2. Defining extensions  
8120 \* 9.5. Authenticator extension processing  
8121 \* 10.1. FIDO AppID Extension (appid)  
8122

8123 #authenticatorcancelReferenced in:  
8124 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8125 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
8126 (4) (5)  
8127 \* 5.1.4.1. PublicKeyCredential's  
8128 [[DiscoverFromExternalSource]](origin, options,  
8129 sameOriginWithAncestors) method (2) (3) (4)  
8130 \* 6.2.2. The authenticatorMakeCredential operation  
8131 \* 6.2.3. The authenticatorGetAssertion operation  
8132

8133 #attestation-objectReferenced in:  
8134 \* 4. Terminology (2) (3)  
8135 \* 5. Web Authentication API  
8136 \* 5.2.1. Information about Public Key Credential (interface  
8137 AuthenticatorAttestationResponse) (2)  
8138 \* 5.4. Options for Credential Creation (dictionary  
8139 PublicKeyCredentialCreationOptions) (2)  
8140 \* 6.2.2. The authenticatorMakeCredential operation (2)  
8141 \* 6.3. Attestation (2) (3)  
8142 \* 6.3.1. Attested credential data  
8143 \* 6.3.4. Generating an Attestation Object (2)  
8144 \* 7.1. Registering a new credential  
8145

8146 #attestation-statementReferenced in:  
8147 \* 4. Terminology (2)  
8148 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8149 [[Create]](origin, options, sameOriginWithAncestors) method (2)

7121 \* 5.2.1. Information about Public Key Credential (interface  
7122 AuthenticatorAttestationResponse) (2) (3)  
7123 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
7124 AttestationConveyancePreference) (2) (3) (4) (5) (6) (7)  
7125 \* 6.3. Attestation (2) (3) (4) (5) (6) (7) (8)  
7126 \* 6.3.2. Attestation Statement Formats (2) (3) (4)  
7127 \* 7.1. Registering a new credential

7128  
7129 #attestation-statement-formatReferenced in:  
7130 \* 5.2.1. Information about Public Key Credential (interface  
7131 AuthenticatorAttestationResponse)  
7132 \* 5.8.4. Authenticator Transport enumeration (enum  
7133 AuthenticatorTransport)  
7134 \* 6.2.1. The authenticatorMakeCredential operation  
7135 \* 6.3. Attestation (2) (3) (4) (5) (6) (7)  
7136 \* 6.3.2. Attestation Statement Formats (2) (3) (4)  
7137 \* 6.3.4. Generating an Attestation Object  
7138 \* 7.1. Registering a new credential (2)

7139  
7140 #attestation-typeReferenced in:  
7141 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7142 [[Create]](origin, options, sameOriginWithAncestors) method  
7143 \* 6.3. Attestation (2) (3) (4) (5) (6)  
7144 \* 6.3.2. Attestation Statement Formats (2)  
7145

7146 #attested-credential-dataReferenced in:  
7147 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7148 [[Create]](origin, options, sameOriginWithAncestors) method  
7149 \* 6.1. Authenticator data (2) (3) (4) (5)  
7150 \* 6.2.1. The authenticatorMakeCredential operation  
7151 \* 6.3. Attestation (2)  
7152 \* 6.3.1. Attested credential data  
7153 \* 6.3.3. Attestation Types

7154  
7155 #aaguidReferenced in:  
7156 \* 4. Terminology  
7157 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7158 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
7159 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
7160 AttestationConveyancePreference)  
7161 \* 7.1. Registering a new credential  
7162 \* 8.2. Packed Attestation Statement Format  
7163 \* 8.3. TPM Attestation Statement Format  
7164

7165 #credentialidlengthReferenced in:  
7166 \* 6.1. Authenticator data  
7167

7168 #credentialidReferenced in:  
7169 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7170 [[Create]](origin, options, sameOriginWithAncestors) method  
7171 \* 6.1. Authenticator data  
7172 \* 7.1. Registering a new credential  
7173

7174 #credentialpublickeyReferenced in:  
7175 \* 6.1. Authenticator data

7176  
7177 \* 7.1. Registering a new credential  
7178 \* 8.2. Packed Attestation Statement Format  
7179 \* 8.3. TPM Attestation Statement Format  
7180 \* 8.4. Android Key Attestation Statement Format  
7181

7182 #signing-procedureReferenced in:  
7183 \* 6.3.2. Attestation Statement Formats  
7184 \* 6.3.4. Generating an Attestation Object

7185 #authenticator-data-for-the-attestationReferenced in:  
7186 \* 8.2. Packed Attestation Statement Format  
7187 \* 8.3. TPM Attestation Statement Format

8150 \* 5.2.1. Information about Public Key Credential (interface  
8151 AuthenticatorAttestationResponse) (2) (3)  
8152 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
8153 AttestationConveyancePreference) (2) (3) (4) (5) (6)  
8154 \* 6.3. Attestation (2) (3) (4) (5) (6) (7) (8)  
8155 \* 6.3.2. Attestation Statement Formats (2) (3) (4)  
8156 \* 7.1. Registering a new credential  
8157 \* 8.7. None Attestation Statement Format  
8158

8159 #attestation-statement-formatReferenced in:  
8160 \* 5.2.1. Information about Public Key Credential (interface  
8161 AuthenticatorAttestationResponse)  
8162 \* 5.10.4. Authenticator Transport enumeration (enum  
8163 AuthenticatorTransport)  
8164 \* 6.2.2. The authenticatorMakeCredential operation  
8165 \* 6.3. Attestation (2) (3) (4) (5) (6) (7)  
8166 \* 6.3.2. Attestation Statement Formats (2) (3) (4)  
8167 \* 6.3.4. Generating an Attestation Object  
8168 \* 7.1. Registering a new credential (2)

8169  
8170 #attestation-typeReferenced in:  
8171 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8172 [[Create]](origin, options, sameOriginWithAncestors) method  
8173 \* 6.3. Attestation (2) (3) (4) (5) (6)  
8174 \* 6.3.2. Attestation Statement Formats (2)  
8175

8176 #attested-credential-dataReferenced in:  
8177 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8178 [[Create]](origin, options, sameOriginWithAncestors) method (2)  
8179 \* 6.1. Authenticator data (2) (3) (4) (5)  
8180 \* 6.2.2. The authenticatorMakeCredential operation  
8181 \* 6.3. Attestation (2)  
8182 \* 6.3.1. Attested credential data  
8183 \* 6.3.3. Attestation Types

8184  
8185 #aaguidReferenced in:  
8186 \* 4. Terminology  
8187 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8188 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)  
8189 (4)

8190 \* 7.1. Registering a new credential  
8191 \* 8.2. Packed Attestation Statement Format  
8192 \* 8.3. TPM Attestation Statement Format  
8193

8194 #credentialidlengthReferenced in:  
8195 \* 6.1. Authenticator data  
8196

8197 #credentialidReferenced in:  
8198 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8199 [[Create]](origin, options, sameOriginWithAncestors) method  
8200 \* 6.1. Authenticator data  
8201 \* 7.1. Registering a new credential (2)  
8202

8203 #credentialpublickeyReferenced in:  
8204 \* 6.1. Authenticator data  
8205 \* 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE\_Key  
8206 format  
8207 \* 7.1. Registering a new credential  
8208 \* 8.2. Packed Attestation Statement Format  
8209 \* 8.3. TPM Attestation Statement Format  
8210 \* 8.4. Android Key Attestation Statement Format  
8211

8212 #signing-procedureReferenced in:  
8213 \* 6.3.2. Attestation Statement Formats  
8214 \* 6.3.4. Generating an Attestation Object

8215 #authenticator-data-for-the-attestationReferenced in:  
8216 \* 8.2. Packed Attestation Statement Format  
8217 \* 8.3. TPM Attestation Statement Format  
8218

718f \* 8.4. Android Key Attestation Statement Format (2)  
7190 \* 8.5. Android SafetyNet Attestation Statement Format  
719c \* 8.6. FIDO U2F Attestation Statement Format  
7191  
7192 #verification-procedure-inputsReferenced in:  
7193 \* 8.2. Packed Attestation Statement Format  
7194 \* 8.3. TPM Attestation Statement Format  
7195 \* 8.4. Android Key Attestation Statement Format  
7196 \* 8.5. Android SafetyNet Attestation Statement Format  
7197 \* 8.6. FIDO U2F Attestation Statement Format  
7198  
7199 #authenticator-data-claimed-to-have-been-used-for-the-attestationReferenced in:  
7200 nced in:  
7201 \* 8.4. Android Key Attestation Statement Format  
7202  
7203 #attestation-trust-pathReferenced in:  
7204 \* 6.3.2. Attestation Statement Formats  
7205 \* 8.2. Packed Attestation Statement Format (2) (3)  
7206 \* 8.3. TPM Attestation Statement Format  
7207 \* 8.4. Android Key Attestation Statement Format  
7208 \* 8.5. Android SafetyNet Attestation Statement Format  
7209 \* 8.6. FIDO U2F Attestation Statement Format  
7210  
7211 #basic-attestationReferenced in:  
7212 \* 6.3.5.1. Privacy  
7213 \* 8.4. Android Key Attestation Statement Format  
7214 \* 8.5. Android SafetyNet Attestation Statement Format  
7215 \* 8.6. FIDO U2F Attestation Statement Format  
  
7216  
7217 #self-attestationReferenced in:  
7218 \* 4. Terminology (2) (3) (4)  
  
7219 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
7220 AttestationConveyancePreference)  
7221 \* 6.3. Attestation (2)  
7222 \* 6.3.2. Attestation Statement Formats  
7223 \* 6.3.3. Attestation Types  
7224 \* 6.3.5.2. Attestation Certificate and Attestation Certificate CA  
7225 Compromise  
7226 \* 7.1. Registering a new credential (2) (3)  
7227 \* 8.2. Packed Attestation Statement Format (2)  
7228 \* 8.6. FIDO U2F Attestation Statement Format  
  
7229  
7230 #privacy-caReferenced in:  
7231 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7232 [[Create]](origin, options, sameOriginWithAncestors) method  
  
7233 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
7234 AttestationConveyancePreference)  
7235 \* 6.3.5.1. Privacy  
7236 \* 8.3. TPM Attestation Statement Format  
  
7237 \* 8.6. FIDO U2F Attestation Statement Format  
7238  
7239 #elliptic-curve-based-direct-anonymous-attestationReferenced in:  
7240 \* 6.3.5.1. Privacy  
7241  
7242 #ecdaaReferenced in:  
7243 \* 6.3.2. Attestation Statement Formats  
7244 \* 6.3.3. Attestation Types  
7245 \* 6.3.5.2. Attestation Certificate and Attestation Certificate CA  
7246 Compromise

821f \* 8.4. Android Key Attestation Statement Format (2)  
8220 \* 8.5. Android SafetyNet Attestation Statement Format  
8221 \* 8.6. FIDO U2F Attestation Statement Format  
8222  
8223 #verification-procedure-inputsReferenced in:  
8224 \* 8.2. Packed Attestation Statement Format  
8225 \* 8.3. TPM Attestation Statement Format  
8226 \* 8.4. Android Key Attestation Statement Format  
8227 \* 8.5. Android SafetyNet Attestation Statement Format  
8228 \* 8.6. FIDO U2F Attestation Statement Format  
8229  
8230 #authenticator-data-claimed-to-have-been-used-for-the-attestationReferenced in:  
8231 nced in:  
8232 \* 8.4. Android Key Attestation Statement Format  
8233  
8234 #attestation-trust-pathReferenced in:  
8235 \* 6.3.2. Attestation Statement Formats  
8236 \* 8.2. Packed Attestation Statement Format (2) (3)  
8237 \* 8.3. TPM Attestation Statement Format  
8238 \* 8.4. Android Key Attestation Statement Format  
8239 \* 8.5. Android SafetyNet Attestation Statement Format  
8240 \* 8.6. FIDO U2F Attestation Statement Format  
8241  
8242 #basic-attestationReferenced in:  
8243 \* 14.1. Attestation Privacy  
8244  
8245 #basicReferenced in:  
8246 \* 8.2. Packed Attestation Statement Format (2)  
8247 \* 8.4. Android Key Attestation Statement Format (2)  
8248 \* 8.5. Android SafetyNet Attestation Statement Format (2)  
8249 \* 8.6. FIDO U2F Attestation Statement Format (2)  
8250  
8251 #self-attestationReferenced in:  
8252 \* 4. Terminology (2) (3) (4)  
8253 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8254 [[Create]](origin, options, sameOriginWithAncestors) method  
8255 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
8256 AttestationConveyancePreference)  
8257 \* 6.3. Attestation (2)  
8258 \* 6.3.2. Attestation Statement Formats  
8259 \* 6.3.3. Attestation Types  
  
8260 \* 7.1. Registering a new credential (2) (3)  
8261 \* 8.2. Packed Attestation Statement Format (2)  
8262 \* 13.2.2. Attestation Certificate and Attestation Certificate CA  
8263 Compromise  
8264  
8265 #selfReferenced in:  
8266 \* 8.2. Packed Attestation Statement Format  
8267  
8268 #attestation-caReferenced in:  
8269 \* 5.4.6. Attestation Conveyance Preference enumeration (enum  
8270 AttestationConveyancePreference)  
8271 \* 6.3.3. Attestation Types (2)  
8272 \* 14.1. Attestation Privacy (2)  
8273  
8274 #attcaReferenced in:  
8275 \* 8.2. Packed Attestation Statement Format  
8276 \* 8.3. TPM Attestation Statement Format (2)  
8277 \* 8.6. FIDO U2F Attestation Statement Format  
8278  
8279 #elliptic-curve-based-direct-anonymous-attestationReferenced in:  
8280 \* 14.1. Attestation Privacy  
8281  
8282 #ecdaaReferenced in:  
8283 \* 6.3.2. Attestation Statement Formats  
8284 \* 6.3.3. Attestation Types

7247 \* 7.1. Registering a new credential  
7248 \* 8.2. Packed Attestation Statement Format (2)  
7249 \* 8.3. TPM Attestation Statement Format (2) (3)

7250  
7251 #attestation-statement-format-identifierReferenced in:  
7252 \* 6.3.2. Attestation Statement Formats  
7253 \* 6.3.4. Generating an Attestation Object

7254  
7255 #identifier-of-the-eccdaa-issuer-public-keyReferenced in:  
7256 \* 7.1. Registering a new credential  
7257 \* 8.2. Packed Attestation Statement Format  
7258 \* 8.3. TPM Attestation Statement Format (2)

7259  
7260 #eccdaa-issuer-public-keyReferenced in:  
7261 \* 6.3.2. Attestation Statement Formats  
7262 \* 6.3.5.1. Privacy  
7263 \* 7.1. Registering a new credential  
7264 \* 8.2. Packed Attestation Statement Format (2) (3)

7265  
7266 #registration-extensionReferenced in:  
7267 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7268 [[Create]](origin, options, sameOriginWithAncestors) method  
7269 \* 9. WebAuthn Extensions (2) (3) (4) (5) (6)  
7270 \* 9.6. Example Extension  
7271 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
7272 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
7273 \* 10.4. Authenticator Selection Extension (authnSel)  
7274 \* 10.5. Supported Extensions Extension (exts)  
7275 \* 10.6. User Verification Index Extension (uvi)  
7276 \* 10.7. Location Extension (loc)  
7277 \* 10.8. User Verification Method Extension (uvm)

7278 \* 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)  
7279 (6) (7)

7280  
7281 #authentication-extensionReferenced in:  
7282 \* 5.1.4.1. PublicKeyCredential's  
7283 [[DiscoverFromExternalSource]](origin, options,  
7284 sameOriginWithAncestors) method  
7285 \* 9. WebAuthn Extensions (2) (3) (4) (5) (6)  
7286 \* 9.6. Example Extension  
7287 \* 10.1. FIDO AppID Extension (appid)  
7288 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
7289 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
7290 \* 10.6. User Verification Index Extension (uvi)  
7291 \* 10.7. Location Extension (loc)  
7292 \* 10.8. User Verification Method Extension (uvm)  
7293 \* 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)  
7294 (6)

7295  
7296 #client-extensionReferenced in:  
7297 \* 5.1.3. Create a new credential - PublicKeyCredential's  
7298 [[Create]](origin, options, sameOriginWithAncestors) method  
7299 \* 5.1.4.1. PublicKeyCredential's  
7300 [[DiscoverFromExternalSource]](origin, options,  
7301 sameOriginWithAncestors) method  
7302 \* 5.7. Authentication Extensions (typedef AuthenticationExtensions)  
7303 \* 9. WebAuthn Extensions  
7304 \* 9.2. Defining extensions  
7305 \* 9.4. Client extension processing

7306  
7307 #authenticator-extensionReferenced in:

8285 \* 7.1. Registering a new credential  
8286 \* 8.2. Packed Attestation Statement Format (2) (3) (4)  
8287 \* 8.3. TPM Attestation Statement Format (2) (3) (4)  
8288 \* 13.2.2. Attestation Certificate and Attestation Certificate CA  
8289 Compromise

8290  
8291 #noneReferenced in:  
8292 \* 8.7. None Attestation Statement Format (2)

8293  
8294 #attestation-statement-format-identifierReferenced in:  
8295 \* 6.3.2. Attestation Statement Formats  
8296 \* 6.3.4. Generating an Attestation Object

8297  
8298 #identifier-of-the-eccdaa-issuer-public-keyReferenced in:  
8299 \* 7.1. Registering a new credential  
8300 \* 8.2. Packed Attestation Statement Format  
8301 \* 8.3. TPM Attestation Statement Format (2)

8302  
8303 #eccdaa-issuer-public-keyReferenced in:  
8304 \* 6.3.2. Attestation Statement Formats

8305 \* 7.1. Registering a new credential  
8306 \* 8.2. Packed Attestation Statement Format (2) (3)  
8307 \* 14.1. Attestation Privacy

8308  
8309 #registration-extensionReferenced in:  
8310 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8311 [[Create]](origin, options, sameOriginWithAncestors) method  
8312 \* 9. WebAuthn Extensions (2) (3) (4) (5) (6)

8313 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
8314 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
8315 \* 10.4. Authenticator Selection Extension (authnSel)  
8316 \* 10.5. Supported Extensions Extension (exts)  
8317 \* 10.6. User Verification Index Extension (uvi)  
8318 \* 10.7. Location Extension (loc)  
8319 \* 10.8. User Verification Method Extension (uvm)  
8320 \* 10.9. Biometric Authenticator Performance Bounds Extension  
8321 (biometricPerfBounds)  
8322 \* 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)  
8323 (6) (7)

8324  
8325 #authentication-extensionReferenced in:  
8326 \* 5.1.4.1. PublicKeyCredential's  
8327 [[DiscoverFromExternalSource]](origin, options,  
8328 sameOriginWithAncestors) method  
8329 \* 9. WebAuthn Extensions (2) (3) (4) (5) (6)

8330 \* 10.2. Simple Transaction Authorization Extension (txAuthSimple)  
8331 \* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)  
8332 \* 10.6. User Verification Index Extension (uvi)  
8333 \* 10.7. Location Extension (loc)  
8334 \* 10.8. User Verification Method Extension (uvm)  
8335 \* 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)  
8336 (6)

8337  
8338 #client-extensionReferenced in:  
8339 \* 5.1.3. Create a new credential - PublicKeyCredential's  
8340 [[Create]](origin, options, sameOriginWithAncestors) method  
8341 \* 5.1.4.1. PublicKeyCredential's  
8342 [[DiscoverFromExternalSource]](origin, options,  
8343 sameOriginWithAncestors) method

8344 \* 9. WebAuthn Extensions  
8345 \* 9.2. Defining extensions  
8346 \* 9.4. Client extension processing  
8347 \* 10.1. FIDO AppID Extension (appid)

8348  
8349 #authenticator-extensionReferenced in:

```

7308 * 5.1.3. Create a new credential - PublicKeyCredential's
7309 [[Create]](origin, options, sameOriginWithAncestors) method
7310 * 5.1.4.1. PublicKeyCredential's
7311 [[DiscoverFromExternalSource]](origin, options,
7312 sameOriginWithAncestors) method
7313 * 5.7. Authentication Extensions (typedef AuthenticationExtensions)
7314 * 9. WebAuthn Extensions (2) (3)
7315 * 9.2. Defining extensions (2)
7316 * 9.3. Extending request parameters
7317 * 9.5. Authenticator extension processing
7318
7319 #extension-identifierReferenced in:
7320 * 5.1. PublicKeyCredential Interface
7321 * 5.1.3. Create a new credential - PublicKeyCredential's
7322 [[Create]](origin, options, sameOriginWithAncestors) method
7323 * 5.1.4.1. PublicKeyCredential's
7324 [[DiscoverFromExternalSource]](origin, options,
7325 sameOriginWithAncestors) method
7326 * 6.1. Authenticator data
7327 * 6.2.1. The authenticatorMakeCredential operation (2)
7328 * 6.2.2. The authenticatorGetAssertion operation (2)
7329
7330 * 9. WebAuthn Extensions (2)
7331 * 9.2. Defining extensions
7332 * 9.3. Extending request parameters
7333 * 9.4. Client extension processing (2)
7334 * 9.5. Authenticator extension processing (2)
7335 * 9.6. Example Extension
7336 * 10.5. Supported Extensions Extension (exts) (2)
7337 * 10.7. Location Extension (loc)
7338 * 11.2. WebAuthn Extension Identifier Registrations
7339
7340 #client-extension-inputReferenced in:
7341 * 9. WebAuthn Extensions (2) (3)
7342
7343 * 9.2. Defining extensions
7344 * 9.3. Extending request parameters (2) (3) (4) (5) (6)
7345 * 9.4. Client extension processing (2) (3) (4)
7346 * 9.6. Example Extension
7347
7348 #authenticator-extension-inputReferenced in:
7349 * 6.2.1. The authenticatorMakeCredential operation
7350 * 6.2.2. The authenticatorGetAssertion operation
7351 * 9. WebAuthn Extensions (2) (3) (4) (5)
7352
7353 * 9.2. Defining extensions
7354 * 9.3. Extending request parameters (2) (3)
7355 * 9.4. Client extension processing
7356 * 9.5. Authenticator extension processing (2) (3)
7357
7358 #client-extension-processingReferenced in:
7359 * 5.1. PublicKeyCredential Interface
7360 * 5.1.3. Create a new credential - PublicKeyCredential's
7361 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7362 * 5.1.4.1. PublicKeyCredential's
7363 [[DiscoverFromExternalSource]](origin, options,
7364 sameOriginWithAncestors) method (2)
7365 * 9. WebAuthn Extensions (2) (3) (4)
7366 * 9.2. Defining extensions
7367
7368 #client-extension-outputReferenced in:
7369 * 5.1. PublicKeyCredential Interface
7370 * 5.1.3. Create a new credential - PublicKeyCredential's
7371 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7372 * 5.1.4.1. PublicKeyCredential's

```

```

8350 * 5.1.3. Create a new credential - PublicKeyCredential's
8351 [[Create]](origin, options, sameOriginWithAncestors) method
8352 * 5.1.4.1. PublicKeyCredential's
8353 [[DiscoverFromExternalSource]](origin, options,
8354 sameOriginWithAncestors) method
8355
8356 * 9. WebAuthn Extensions (2) (3)
8357 * 9.2. Defining extensions (2)
8358 * 9.3. Extending request parameters
8359 * 9.5. Authenticator extension processing
8360
8361 #extension-identifierReferenced in:
8362 * 5.1. PublicKeyCredential Interface
8363 * 5.1.3. Create a new credential - PublicKeyCredential's
8364 [[Create]](origin, options, sameOriginWithAncestors) method
8365 * 5.1.4.1. PublicKeyCredential's
8366 [[DiscoverFromExternalSource]](origin, options,
8367 sameOriginWithAncestors) method
8368 * 6.1. Authenticator data
8369 * 6.2.2. The authenticatorMakeCredential operation (2)
8370 * 6.2.3. The authenticatorGetAssertion operation (2)
8371 * 7.1. Registering a new credential (2)
8372 * 7.2. Verifying an authentication assertion (2)
8373 * 9. WebAuthn Extensions (2)
8374 * 9.2. Defining extensions
8375 * 9.3. Extending request parameters
8376 * 9.4. Client extension processing (2)
8377 * 9.5. Authenticator extension processing (2)
8378
8379 * 10.5. Supported Extensions Extension (exts) (2)
8380
8381 * 11.2. WebAuthn Extension Identifier Registrations
8382
8383 #client-extension-inputReferenced in:
8384 * 5.7. Authentication Extensions Client Inputs (typedef
8385 AuthenticationExtensionsClientInputs)
8386 * 7.1. Registering a new credential
8387 * 7.2. Verifying an authentication assertion
8388 * 9. WebAuthn Extensions (2) (3) (4)
8389 * 9.2. Defining extensions
8390 * 9.3. Extending request parameters (2) (3) (4) (5) (6)
8391 * 9.4. Client extension processing (2) (3) (4)
8392
8393 #authenticator-extension-inputReferenced in:
8394 * 5.9. Authentication Extensions Authenticator Inputs (typedef
8395 AuthenticationExtensionsAuthenticatorInputs)
8396 * 6.2.2. The authenticatorMakeCredential operation (2)
8397 * 6.2.3. The authenticatorGetAssertion operation (2)
8398 * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
8399 * 9.2. Defining extensions
8400 * 9.3. Extending request parameters (2) (3)
8401 * 9.4. Client extension processing
8402 * 9.5. Authenticator extension processing (2) (3)
8403
8404 #client-extension-processingReferenced in:
8405 * 5.1. PublicKeyCredential Interface
8406 * 5.1.3. Create a new credential - PublicKeyCredential's
8407 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8408 * 5.1.4.1. PublicKeyCredential's
8409 [[DiscoverFromExternalSource]](origin, options,
8410 sameOriginWithAncestors) method (2)
8411 * 9. WebAuthn Extensions (2) (3) (4)
8412 * 9.2. Defining extensions
8413
8414 #client-extension-outputReferenced in:
8415 * 5.1. PublicKeyCredential Interface
8416 * 5.1.3. Create a new credential - PublicKeyCredential's
8417 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8418 * 5.1.4.1. PublicKeyCredential's

```

```

7370 [[DiscoverFromExternalSource]](origin, options,
7371 sameOriginWithAncestors) method (2)
7372 * 9. WebAuthn Extensions (2) (3)

7373
7374 * 9.2. Defining extensions (2) (3)
7375 * 9.4. Client extension processing (2) (3)
7376 * 9.6. Example Extension
7377
7378 #authenticator-extension-processingReferenced in:
7379 * 6.2.1. The authenticatorMakeCredential operation
7380 * 6.2.2. The authenticatorGetAssertion operation
7381 * 9. WebAuthn Extensions
7382 * 9.2. Defining extensions
7383 * 9.5. Authenticator extension processing
7384
7385 #authenticator-extension-outputReferenced in:
7386 * 6.1. Authenticator data
7387 * 9. WebAuthn Extensions (2) (3)

7388
7389 * 9.2. Defining extensions (2) (3)
7390 * 9.4. Client extension processing
7391 * 9.5. Authenticator extension processing
7392 * 9.6. Example Extension
7393 * 10.5. Supported Extensions Extension (exts)
7394 * 10.6. User Verification Index Extension (uvi)
7395 * 10.7. Location Extension (loc)
7396 * 10.8. User Verification Method Extension (uvm)

7397
7398 #typedefdef-authenticatorselectionlistReferenced in:
7399 * 10.4. Authenticator Selection Extension (authnSel)
7400
7401 #typedefdef-aaguidReferenced in:
7402 * 10.4. Authenticator Selection Extension (authnSel)

```

7401

```

8416 [[DiscoverFromExternalSource]](origin, options,
8417 sameOriginWithAncestors) method (2)
8418 * 5.8. Authentication Extensions Client Outputs (typedef
8419 AuthenticationExtensionsClientOutputs)
8420 * 7.1. Registering a new credential
8421 * 7.2. Verifying an authentication assertion
8422 * 9. WebAuthn Extensions (2) (3) (4)
8423 * 9.2. Defining extensions (2) (3)
8424 * 9.4. Client extension processing (2) (3)

8425
8426 #authenticator-extension-processingReferenced in:
8427 * 6.2.2. The authenticatorMakeCredential operation
8428 * 6.2.3. The authenticatorGetAssertion operation
8429 * 9. WebAuthn Extensions
8430 * 9.2. Defining extensions
8431 * 9.5. Authenticator extension processing
8432
8433 #authenticator-extension-outputReferenced in:
8434 * 6.1. Authenticator data
8435 * 7.1. Registering a new credential
8436 * 7.2. Verifying an authentication assertion
8437 * 9. WebAuthn Extensions (2) (3) (4)
8438 * 9.2. Defining extensions (2) (3)
8439 * 9.4. Client extension processing
8440 * 9.5. Authenticator extension processing

8441
8442 * 10.5. Supported Extensions Extension (exts)
8443 * 10.6. User Verification Index Extension (uvi)

8444
8445 * 10.8. User Verification Method Extension (uvm)

8446 #dictdef-txauthgenericargReferenced in:
8447 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
8448
8449 #typedefdef-authenticatorselectionlistReferenced in:
8450 * 10.4. Authenticator Selection Extension (authnSel) (2)
8451
8452 #typedefdef-aaguidReferenced in:
8453 * 10.4. Authenticator Selection Extension (authnSel)
8454
8455 #typedefdef-authenticationextensionssupportedReferenced in:
8456 * 10.5. Supported Extensions Extension (exts)
8457
8458 #typedefdef-uvmentryReferenced in:
8459 * 10.8. User Verification Method Extension (uvm)
8460
8461 #typedefdef-uvmentriesReferenced in:
8462 * 10.8. User Verification Method Extension (uvm)
8463
8464 #anonymization-caReferenced in:
8465 * 5.1.3. Create a new credential - PublicKeyCredential's
8466 [[Create]](origin, options, sameOriginWithAncestors) method
8467 * 5.4.6. Attestation Conveyance Preference enumeration (enum
8468 AttestationConveyancePreference)
8469 * 14.1. Attestation Privacy (2) (3)

```