

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

0005 W3C

0006

0007 Web Authentication: An API for accessing Public Key Credentials - Level 1

0008

0009 W3C Working Draft, 5 December 2017

0010

0011 This version:

0012 <https://www.w3.org/TR/2017/WD-webauthn-20171205/>

0013

0014 Latest published Version:

0015 <https://www.w3.org/TR/webauthn/>

0016

0017 Editor's Draft:

0018 <https://w3c.github.io/webauthn/>

0019

0020 Previous versions:

0021 <https://www.w3.org/TR/2017/WD-webauthn-20170811/>

0022 <https://www.w3.org/TR/2017/WD-webauthn-20170505/>

0023 <https://www.w3.org/TR/2017/WD-webauthn-20170216/>

0024 <https://www.w3.org/TR/2016/WD-webauthn-20161207/>

0025 <https://www.w3.org/TR/2016/WD-webauthn-20160928/>

0026 <https://www.w3.org/TR/2016/WD-webauthn-20160902/>

0027 <https://www.w3.org/TR/2016/WD-webauthn-20160531/>

0028

0029 Issue Tracking:

0030 [Github](#)

0031

0032 Editors:

0033 [Vijay Bharadwaj \(Microsoft\)](#)

0034 [Hubert Le Van Gong \(PayPal\)](#)

0035 [Dirk Balfanz \(Google\)](#)

0036 [Alexei Czeskis \(Google\)](#)

0037 [Arnar Birgisson \(Google\)](#)

0038 [Jeff Hodges \(PayPal\)](#)

0039

0040 [Michael B. Jones \(Microsoft\)](#)

0041

0042 [Rolf Lindemann \(Nok Nok Labs\)](#)

0043 [J.C. Jones \(Mozilla\)](#)

0044

0045

0046

0047

0048

0049

0050

0051

0052

0053

0054

0055

0056

0057

0058

0059

0060

0061

0062

0063

0064

0065

0066

0042 Tests:

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

0044

0045

0046 Copyright 2017 W3C[^] (MIT, ERCIM, Keio, Beihang). W3C liability, trademark and document use rules apply.

0047

0048

0049

0050

0051

0052

0053

0054

0055

0056

0057

0058

0059

0060

0061

0062

0063

0064

0065

0066

Abstract

This specification defines an API enabling the creation and use of

0001 THE_URL:file://localhost/Users/jehodges/documents/work/standards/W3C/webauthn/index-master-tr-716a169-WD-08a.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

0005 W3C

0006

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

0008

0009 W3C Working Draft, 6 March 2018

0010

0011 This version:

0012 <https://www.w3.org/TR/2018/WD-webauthn-20180306/>

0013

0014 Latest published version:

0015 <https://www.w3.org/TR/webauthn/>

0016

0017 Editor's Draft:

0018 <https://w3c.github.io/webauthn/>

0019

0020 Previous Versions:

0021 <https://www.w3.org/TR/2017/WD-webauthn-20171205/>

0022 <https://www.w3.org/TR/2017/WD-webauthn-20170811/>

0023 <https://www.w3.org/TR/2017/WD-webauthn-20170505/>

0024 <https://www.w3.org/TR/2017/WD-webauthn-20170216/>

0025 <https://www.w3.org/TR/2016/WD-webauthn-20161207/>

0026 <https://www.w3.org/TR/2016/WD-webauthn-20160928/>

0027 <https://www.w3.org/TR/2016/WD-webauthn-20160902/>

0028 <https://www.w3.org/TR/2016/WD-webauthn-20160531/>

0029

0030 Issue Tracking:

0031 [GitHub](#)

0032

0033 Editors:

0034 [Dirk Balfanz \(Google\)](#)

0035 [Alexei Czeskis \(Google\)](#)

0036

0037 [Jeff Hodges \(PayPal\)](#)

0038 [J.C. Jones \(Mozilla\)](#)

0039 [Michael B. Jones \(Microsoft\)](#)

0040 [Akshay Kumar \(Microsoft\)](#)

0041 [Angelo Liao \(Microsoft\)](#)

0042 [Rolf Lindemann \(Nok Nok Labs\)](#)

0043 [Emil Lundberg \(Yubico\)](#)

0044

0045 Former Editors:

0046 [Vijay Bharadwaj \(Microsoft\)](#)

0047 [Arnar Birgisson \(Google\)](#)

0048 [Hubert Le Van Gong \(PayPal\)](#)

0049

0050 Contributors:

0051 [Christiaan Brand \(Google\)](#)

0052 [Adam Langley \(Google\)](#)

0053 [Giridhar Mandyam \(Qualcomm\)](#)

0054 [Mike West \(Google\)](#)

0055 [Johan Verrept \(VASCO Data Security\)](#)

0056 [Jeffrey Yasskin \(Google\)](#)

0057

0058 Tests:

0059 [web-platform-tests webauthn/](#) (ongoing work)

0060

0061

0062

0063

0064

0065

0066

0060 Copyright 2018 W3C[^] (MIT, ERCIM, Keio, Beihang). W3C liability, trademark and document use rules apply.

0061

0062

0063

0064

0065

0066

Abstract

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/>.

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

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

0080 This document was produced by a group operating under the W3C Patent
0081 Policy. W3C maintains a public list of any patent disclosures made in
0082 connection with the deliverables of the group; that page also includes
0083 instructions for disclosing a patent. An individual who has actual
0084 knowledge of a patent which the individual believes contains Essential
0085 Claim(s) must disclose the information in accordance with section 6 of
0086 the W3C Patent Policy.

0087 This document is governed by the 1 [March 2017](#) W3C Process Document.

0088 Table of Contents

0089 1. 1 Introduction
0090 1. 1.1 Use Cases
0091 1. 1.1.1 Registration
0092 2. 1.1.2 Authentication
0093 3. 1.1.3 Other use cases and configurations
0094 2. 2 Conformance
0095 1. 2.1 User Agents
0096 2. 2.2 Authenticators
0097 3. 2.3 Relying Parties

0098 3. 3 Dependencies
0099 4. 4 Terminology
0100 5. 5 Web Authentication API
0101 1. 5.1 PublicKeyCredential Interface
0102 1. 5.1.1 CredentialCreationOptions Extension
0103 2. 5.1.2 CredentialRequestOptions Extension
0104 3. 5.1.3 Create a new credential - PublicKeyCredential's
0105 `[[Create]](origin, options, sameOriginWithAncestors)`
0106 method
0107 4. 5.1.4 Use an existing credential to make an assertion -
0108 PublicKeyCredential's `[[Get]](options)` method
0109 1. 5.1.4.1 PublicKeyCredential's
0110 `[[DiscoverFromExternalSource]](origin, options,`
0111 `sameOriginWithAncestors)` method
0112 5. 5.1.5 Store an existing credential -

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

0079 Status of this document

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

0085 This document was published by the Web Authentication Working Group as
0086 a Working Draft. This document is intended to become a W3C
0087 Recommendation. Feedback and comments on this specification are
0088 welcome. Please use Github issues. Discussions may also be found in the
0089 public-webauthn@w3.org archives.

0090 Publication as a Working Draft does not imply endorsement by the W3C
0091 Membership. This is a draft document and may be updated, replaced or
0092 obsoleted by other documents at any time. It is inappropriate to cite
0093 this document as other than work in progress.

0094 This document was produced by a group operating under the W3C Patent
0095 Policy. W3C maintains a public list of any patent disclosures made in
0096 connection with the deliverables of the group; that page also includes
0097 instructions for disclosing a patent. An individual who has actual
0098 knowledge of a patent which the individual believes contains Essential
0099 Claim(s) must disclose the information in accordance with section 6 of
0100 the W3C Patent Policy.

0101 This document is governed by the 1 [February 2018](#) W3C Process Document.

0102 Table of Contents

0103 1. 1 Introduction
0104 1. 1.1 Use Cases
0105 1. 1.1.1 Registration
0106 2. 1.1.2 Authentication
0107 3. 1.1.3 Other use cases and configurations
0108 2. 2 Conformance
0109 1. 2.1 User Agents
0110 2. 2.2 Authenticators
0111 1. 2.2.1 Backwards Compatibility with FIDO U2F
0112 3. 2.3 Relying Parties
0113 4. 2.4 All Conformance Classes
0114 3. 3 Dependencies
0115 4. 4 Terminology
0116 5. 5 Web Authentication API
0117 1. 5.1 PublicKeyCredential Interface
0118 1. 5.1.1 CredentialCreationOptions [Dictionary](#) Extension
0119 2. 5.1.2 CredentialRequestOptions [Dictionary](#) Extension
0120 3. 5.1.3 Create a new credential - PublicKeyCredential's
0121 `[[Create]](origin, options, sameOriginWithAncestors)`
0122 method
0123 4. 5.1.4 Use an existing credential to make an assertion -
0124 PublicKeyCredential's `[[Get]](options)` method
0125 1. 5.1.4.1 PublicKeyCredential's
0126 `[[DiscoverFromExternalSource]](origin, options,`
0127 `sameOriginWithAncestors)` method
0128 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
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

0174 2. 6.3.2 Attestation Statement Formats
0175 3. 6.3.3 Attestation Types
0176 4. 6.3.4 Generating an Attestation Object
0177 5. 6.3.5 Security Considerations
0178 1. 6.3.5.1 Privacy
0179 2. 6.3.5.2 Attestation Certificate and Attestation

0137 PublicKeyCredential's `[[Store]](credential,`
0138 `sameOriginWithAncestors)` method
0139 6. 5.1.6 Preventing silent access to an existing credential
0140 - PublicKeyCredential's
0141 `[[preventSilentAccess]](credential,`
0142 `sameOriginWithAncestors)` method
0143 7. 5.1.7 Availability of User-Verifying Platform
0144 Authenticator - PublicKeyCredential's
0145 `isUserVerifyingPlatformAuthenticatorAvailable()` method
0146 2. 5.2 Authenticator Responses (interface AuthenticatorResponse)
0147 1. 5.2.1 Information about Public Key Credential (interface
0148 AuthenticatorAttestationResponse)
0149 2. 5.2.2 Web Authentication Assertion (interface
0150 AuthenticatorAssertionResponse)
0151 3. 5.3 Parameters for Credential Generation (dictionary
0152 PublicKeyCredentialParameters)
0153 4. 5.4 Options for Credential Creation (dictionary
0154 PublicKeyCredentialCreationOptions)
0155 1. 5.4.1 Public Key Entity Description (dictionary
0156 PublicKeyCredentialEntity)
0157 2. 5.4.2 RP Parameters for Credential Generation (dictionary
0158 PublicKeyCredentialRpEntity)
0159 3. 5.4.3 User Account Parameters for Credential Generation
0160 (dictionary PublicKeyCredentialUserEntity)
0161 4. 5.4.4 Authenticator Selection Criteria (dictionary
0162 AuthenticatorSelectionCriteria)
0163 5. 5.4.5 Authenticator Attachment enumeration (enum
0164 AuthenticatorAttachment)
0165 6. 5.4.6 Attestation Conveyance Preference enumeration (enum
0166 AttestationConveyancePreference)
0167 5. 5.5 Options for Assertion Generation (dictionary
0168 PublicKeyCredentialRequestOptions)
0169 6. 5.6 Abort operations with AbortSignal
0170 7. 5.7 Authentication Extensions Client Inputs (typedef
0171 AuthenticationExtensionsClientInputs)
0172 8. 5.8 Authentication Extensions Client Outputs (typedef
0173 AuthenticationExtensionsClientOutputs)
0174 9. 5.9 Authentication Extensions Authenticator Inputs (typedef
0175 AuthenticationExtensionsAuthenticatorInputs)
0176 10. 5.10 Supporting Data Structures
0177 1. 5.10.1 Client data used in WebAuthn signatures
0178 (dictionary CollectedClientData)
0179 2. 5.10.2 Credential Type enumeration (enum
0180 PublicKeyCredentialType)
0181 3. 5.10.3 Credential Descriptor (dictionary
0182 PublicKeyCredentialDescriptor)
0183 4. 5.10.4 Authenticator Transport enumeration (enum
0184 AuthenticatorTransport)
0185 5. 5.10.5 Cryptographic Algorithm Identifier (typedef
0186 COSEAlgorithmIdentifier)
0187 6. 5.10.6 User Verification Requirement enumeration (enum
0188 UserVerificationRequirement)
0189 6. 6 WebAuthn Authenticator Model
0190 1. 6.1 Authenticator data
0191 1. 6.1.1 Signature Counter Considerations
0192 2. 6.2 Authenticator operations
0193 1. 6.2.1 Lookup Credential Source by Credential ID algorithm
0194 2. 6.2.2 The authenticatorMakeCredential operation
0195 3. 6.2.3 The authenticatorGetAssertion operation
0196 4. 6.2.4 The authenticatorCancel operation
0197 3. 6.3 Attestation
0198 1. 6.3.1 Attested credential data
0199 1. 6.3.1.1 Examples of credentialPublicKey Values
0200 encoded in COSE_Key format
0201 2. 6.3.2 Attestation Statement Formats
0202 3. 6.3.3 Attestation Types
0203 4. 6.3.4 Generating an Attestation Object
0204 5. 6.3.5 Signature Formats for Packed Attestation, FIDO U2F
0205 Attestation, and Assertion Signatures

018c **Certificate CA Compromise**
0181 **3. 6.3.5.3 Attestation Certificate Hierarchy**
0182 7. 7 Relying Party Operations
0183 1. 7.1 Registering a new credential
0184 2. 7.2 Verifying an authentication assertion
0185 8. 8 Defined Attestation Statement Formats
0186 1. 8.1 Attestation Statement Format Identifiers
0187 2. 8.2 Packed Attestation Statement Format
0188 1. 8.2.1 Packed attestation statement certificate
0185 requirements
019c 3. 8.3 TPM Attestation Statement Format
0191 1. 8.3.1 TPM attestation statement certificate requirements
0192 4. 8.4 Android Key Attestation Statement Format
0193 5. 8.5 Android SafetyNet Attestation Statement Format
0194 6. 8.6 FIDO U2F Attestation Statement Format

0195 9. 9 WebAuthn Extensions
0196 1. 9.1 Extension Identifiers
0197 2. 9.2 Defining extensions
0198 3. 9.3 Extending request parameters
0199 4. 9.4 Client extension processing
0200 5. 9.5 Authenticator extension processing
0201 **6. 9.6 Example Extension**
0202 10. 10 Defined Extensions
0203 1. 10.1 FIDO AppID Extension (appid)
0204 2. 10.2 Simple Transaction Authorization Extension (txAuthSimple)
0205 3. 10.3 Generic Transaction Authorization Extension
0206 (txAuthGeneric)
0207 4. 10.4 Authenticator Selection Extension (authnSel)
0208 5. 10.5 Supported Extensions Extension (exts)
0209 6. 10.6 User Verification Index Extension (uvi)
0210 7. 10.7 Location Extension (loc)
0211 8. 10.8 User Verification Method Extension (uvm)

0212 11. 11 IANA Considerations
0213 1. 11.1 WebAuthn Attestation Statement Format Identifier
0214 Registrations
0215 2. 11.2 WebAuthn Extension Identifier Registrations
0216 3. 11.3 COSE Algorithm Registrations
0217 12. 12 Sample scenarios
0218 1. 12.1 Registration
0219 2. 12.2 Registration Specifically with User Verifying Platform
0220 Authenticator
0221 3. 12.3 Authentication
0222 4. 12.4 Aborting Authentication Operations
0223 5. 12.5 Decommissioning
0224 13. 13 Security Considerations
0225 1. 13.1 Cryptographic Challenges
0226 **14. 14 Acknowledgements**
0227 **15. Index**

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

0206 7. 7 Relying Party Operations
0207 1. 7.1 Registering a new credential
0208 2. 7.2 Verifying an authentication assertion
0209 8. 8 Defined Attestation Statement Formats
0210 1. 8.1 Attestation Statement Format Identifiers
0211 2. 8.2 Packed Attestation Statement Format
0212 1. 8.2.1 Packed attestation statement certificate
0213 requirements
0214 3. 8.3 TPM Attestation Statement Format
0215 1. 8.3.1 TPM attestation statement certificate requirements
0216 4. 8.4 Android Key Attestation Statement Format
0217 5. 8.5 Android SafetyNet Attestation Statement Format
0218 6. 8.6 FIDO U2F Attestation Statement Format
0219 **7. 8.7 None Attestation Statement Format**
0220 9. 9 WebAuthn Extensions
0221 1. 9.1 Extension Identifiers
0222 2. 9.2 Defining extensions
0223 3. 9.3 Extending request parameters
0224 4. 9.4 Client extension processing
0225 5. 9.5 Authenticator extension processing

0226 10. 10 Defined Extensions
0227 1. 10.1 FIDO AppID Extension (appid)
0228 2. 10.2 Simple Transaction Authorization Extension (txAuthSimple)
0229 3. 10.3 Generic Transaction Authorization Extension
0230 (txAuthGeneric)
0231 4. 10.4 Authenticator Selection Extension (authnSel)
0232 5. 10.5 Supported Extensions Extension (exts)
0233 6. 10.6 User Verification Index Extension (uvi)
0234 7. 10.7 Location Extension (loc)
0235 8. 10.8 User Verification Method Extension (uvm)
0236 **9. 10.9 Biometric Authenticator Performance Bounds Extension**
0237 **(biometricPerfBounds)**
0238 11. 11 IANA Considerations
0239 1. 11.1 WebAuthn Attestation Statement Format Identifier
0240 Registrations
0241 2. 11.2 WebAuthn Extension Identifier Registrations
0242 3. 11.3 COSE Algorithm Registrations
0243 12. 12 Sample scenarios
0244 1. 12.1 Registration
0245 2. 12.2 Registration Specifically with User Verifying Platform
0246 Authenticator
0247 3. 12.3 Authentication
0248 4. 12.4 Aborting Authentication Operations
0249 5. 12.5 Decommissioning
0250 13. 13 Security Considerations
0251 1. 13.1 Cryptographic Challenges
0252 **2. 13.2 Attestation Security Considerations**
0253 1. 13.2.1 Attestation Certificate Hierarchy
0254 2. 13.2.2 Attestation Certificate and Attestation
0255 Certificate CA Compromise
0256 3. 13.3 credentialId Unsigned
0257 4. 13.4 Browser Permissions Framework and Extensions
0258 **14. 14 Privacy Considerations**
0259 1. 14.1 Attestation Privacy
0260 2. 14.2 Registration Ceremony Privacy
0261 3. 14.3 Authentication Ceremony Privacy
0262 **15. 15 Acknowledgements**
0263 **16. Index**
0264 1. Terms defined by this specification
0265 2. Terms defined by reference
0266 **17. References**
0267 1. Normative References
0268 2. Informative References
0269 **18. IDL Index**
0270 **19. Issues Index**
0271
0272 1. Introduction

0237
0238
0239
0240
0241
0242
0243
0244
0245
0246
0247
0248
0249
0250
0251
0252
0253
0254
0255
0256
0257
0258
0259
0260
0261
0262
0263
0264
0265
0266
0267
0268
0269
0270
0271
0272
0273
0274
0275
0276
0277
0278
0279
0280
0281
0282
0283
0284
0285
0286
0287
0288
0289
0290
0291
0292
0293
0294
0295
0296
0297
0298
0299
0300
0301
0302
0303
0304
0305
0306
0307
0308
0309
0310
0311
0312
0313
0314
0315
0316
0317
0318
0319
0320
0321
0322
0323
0324
0325
0326
0327
0328
0329
0330
0331
0332
0333
0334
0335
0336
0337
0338
0339
0340
0341
0342

This section is not normative.

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

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

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

1.1. Use Cases

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

1.1.1. Registration

* On a phone:

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

1.1.2. Authentication

* On a laptop or desktop:

- + User navigates to example.com in a browser, sees an option to "Sign in with your phone."
- + User chooses this option and gets a message from the browser,

0273
0274
0275
0276
0277
0278
0279
0280
0281
0282
0283
0284
0285
0286
0287
0288
0289
0290
0291
0292
0293
0294
0295
0296
0297
0298
0299
0300
0301
0302
0303
0304
0305
0306
0307
0308
0309
0310
0311
0312
0313
0314
0315
0316
0317
0318
0319
0320
0321
0322
0323
0324
0325
0326
0327
0328
0329
0330
0331
0332
0333
0334
0335
0336
0337
0338
0339
0340
0341
0342

This section is not normative.

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

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

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

1.1. Use Cases

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

1.1.1. Registration

* On a phone:

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

1.1.2. Authentication

* On a laptop or desktop:

- + User navigates to example.com in a browser, sees an option to "Sign in with your phone."
- + User chooses this option and gets a message from the browser,

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.
 0319
 0320 1.1.3. Other use cases and configurations
 0321
 0322 A variety of additional use cases and configurations are also possible,
 0323 including (but not limited to):
 0324 * A user navigates to example.com on their laptop, is guided through
 0325 a flow to create and register a credential on their phone.
 0326 * A user obtains an discrete, roaming authenticator, such as a "fob"
 0327 with USB or USB+NFC/BLE connectivity options, loads example.com in
 0328 their browser on a laptop or phone, and is guided through a flow to
 0329 create and register a credential on the fob.
 0330 * A Relying Party prompts the user for their authorization gesture in
 0331 order to authorize a single transaction, such as a payment or other
 0332 financial transaction.
 0333
 0334 2. Conformance
 0335
 0336 This specification defines three conformance classes. Each of these
 0337 classes is specified so that conforming members of the class are secure
 0338 against non-conforming or hostile members of the other classes.
 0339
 0340 2.1. User Agents
 0341
 0342 A User Agent MUST behave as described by 5 Web Authentication API in
 0343 order to be considered conformant. Conforming User Agents MAY implement
 0344 algorithms given in this specification in any way desired, so long as
 0345 the end result is indistinguishable from the result that would be
 0346 obtained by the specification's algorithms.
 0347
 0348 A conforming User Agent MUST also be a conforming implementation of the
 0349 IDL fragments of this specification, as described in the "Web IDL"
 0350 specification. [WebIDL-1]
 0351
 0352 2.2. Authenticators
 0353
 0354 An authenticator MUST provide the operations defined by 6 WebAuthn
 0355 Authenticator model, and those operations MUST behave as described
 0356 there. This is a set of functional and security requirements for an
 0357 authenticator to be usable by a Conforming User Agent.
 0358
 0359 As described in 1.1 Use Cases, an authenticator may be implemented in
 0360 the operating system underlying the User Agent, or in external
 0361 hardware, or a combination of both.
 0362
 0363
 0364
 0365 2.3. Relying Parties
 0366 A Relying Party MUST behave as described in 7 Relying Party Operations
 0367 to **get** the security benefits offered by this specification.

0343 "Please complete this action on your phone."
 0344 * Next, on their phone:
 0345 + User sees a discrete prompt or notification, "Sign in to
 0346 example.com."
 0347 + User selects this prompt / notification.
 0348 + User is shown a list of their example.com identities, e.g.,
 0349 "Sign in as Alice / Sign in as Bob."
 0350 + User picks an identity, is prompted for an authorization
 0351 gesture (PIN, biometric, etc.) and provides this.
 0352 * Now, back on the laptop:
 0353 + Web page shows that the selected user is signed in, and
 0354 navigates to the signed-in page.
 0355
 0356 1.1.3. Other use cases and configurations
 0357
 0358 A variety of additional use cases and configurations are also possible,
 0359 including (but not limited to):
 0360 * A user navigates to example.com on their laptop, is guided through
 0361 a flow to create and register a credential on their phone.
 0362 * A user obtains a discrete, roaming authenticator, such as a "fob"
 0363 with USB or USB+NFC/BLE connectivity options, loads example.com in
 0364 their browser on a laptop or phone, and is guided through a flow to
 0365 create and register a credential on the fob.
 0366 * A Relying Party prompts the user for their authorization gesture in
 0367 order to authorize a single transaction, such as a payment or other
 0368 financial transaction.
 0369
 0370 2. Conformance
 0371
 0372 This specification defines three conformance classes. Each of these
 0373 classes is specified so that conforming members of the class are secure
 0374 against non-conforming or hostile members of the other classes.
 0375
 0376 2.1. User Agents
 0377
 0378 A User Agent MUST behave as described by 5 Web Authentication API in
 0379 order to be considered conformant. Conforming User Agents MAY implement
 0380 algorithms given in this specification in any way desired, so long as
 0381 the end result is indistinguishable from the result that would be
 0382 obtained by the specification's algorithms.
 0383
 0384 A conforming User Agent MUST also be a conforming implementation of the
 0385 IDL fragments of this specification, as described in the "Web IDL"
 0386 specification. [WebIDL-1]
 0387
 0388 2.2. Authenticators
 0389
 0390 An authenticator MUST provide the operations defined by 6 WebAuthn
 0391 Authenticator Model, and those operations MUST behave as described
 0392 there. This is a set of functional and security requirements for an
 0393 authenticator to be usable by a Conforming User Agent.
 0394
 0395 As described in 1.1 Use Cases, an authenticator may be implemented in
 0396 the operating system underlying the User Agent, or in external
 0397 hardware, or a combination of both.
 0398
 0399 2.2.1. Backwards Compatibility with FIDO U2F
 0400
 0401 Authenticators that only support the 8.6 FIDO U2F Attestation
 0402 Statement Format have no mechanism to store a user handle, so the
 0403 returned userHandle will always be null.
 0404
 0405 2.3. Relying Parties
 0406
 0407 A Relying Party MUST behave as described in 7 Relying Party Operations
 0408 to **obtain** the security benefits offered by this specification.
 0409
 0410 2.4. All Conformance Classes
 0411
 0412 All CBOR encoding performed by the members of the above conformance

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
0414
0415
0416

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

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

4. Terminology

Assertion

See Authentication Assertion.

0417
0418
0419
0420
0421
0422
0423
0424
0425

0413
0414
0415
0416
0417
0418
0419
0420
0421
0422
0423
0424
0425
0426
0427
0428
0429
0430
0431
0432
0433
0434
0435
0436
0437
0438
0439
0440
0441
0442
0443
0444
0445
0446
0447
0448
0449
0450
0451
0452
0453
0454
0455
0456
0457
0458
0459
0460
0461
0462
0463
0464
0465
0466
0467
0468
0469
0470
0471
0472
0473
0474
0475
0476
0477
0478
0479
0480
0481
0482

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

3. Dependencies

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

Base64url encoding

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

CBOR

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

CDDL

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

COSE

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

Credential Management

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

DOM

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

ECMAScript

%ArrayBuffer% is defined in [ECMAScript].

HTML

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

Web IDL

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

FIDO AppID

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

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

4. Terminology

Assertion

See Authentication Assertion.

0426 **Attestation**
 0427 Generally, attestation is a statement serving to bear witness,
 0428 confirm, or authenticate. In the WebAuthn context, attestation
 0429 is employed to attest to the provenance of an authenticator and
 0430 the data it emits; including, for example: credential IDs,
 0431 credential key pairs, signature counters, etc. An attestation
 0432 statement is conveyed in an attestation object during
 0433 registration. See also 6.3 Attestation and Figure 3. Whether or
 0434 how the client platform conveys the attestation statement and
 0435 AAGUID portions of the attestation object to the Relying Party
 0436 is described by attestation conveyance.
 0437
 0438 **Attestation Certificate**
 0439 A X.509 Certificate for the attestation key pair used by an
 0440 authenticator to attest to its manufacture and capabilities. At
 0441 registration time, the authenticator uses the attestation
 0442 private key to sign the Relying Party-specific credential public
 0443 key (and additional data) that it generates and returns via the
 0444 authenticatorMakeCredential operation. Relying Parties use the
 0445 attestation public key conveyed in the attestation certificate
 0446 to verify the attestation signature. Note that in the case of
 0447 self attestation, the authenticator has no distinct attestation
 0448 key pair nor attestation certificate, see self attestation for
 0449 details.
 0450
 0451 **Authentication**
 0452 The ceremony where a user, and the user's computing device(s)
 0453 (containing at least one authenticator) work in concert to
 0454 cryptographically prove to an Relying Party that the user
 0455 controls the credential private key associated with a
 0456 previously-registered public key credential (see Registration).
 0457 Note that this includes a test of user presence or user
 0458 verification.
 0459
 0460 **Authentication Assertion**
 0461 The cryptographically signed AuthenticatorAssertionResponse
 0462 object returned by an authenticator as the result of a
 0463 authenticatorGetAssertion operation.
 0464
 0465 This corresponds to the [CREDENTIAL-MANAGEMENT-1]
 0466 specification's single-use credentials.
 0467
 0468 **Authenticator**
 0469 A cryptographic entity used by a WebAuthn Client to (i) generate
 0470 a public key credential and register it with a Relying Party,
 0471 and (ii) authenticate by potentially verifying the user, and
 0472 then cryptographically signing and returning, in the form of an
 0473 Authentication Assertion, a challenge and other data presented
 0474 by a Relying Party (in concert with the WebAuthn Client).
 0475
 0476 **Authorization Gesture**
 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
 0485 **Biometric Recognition**
 0486 The automated recognition of individuals based on their
 0487 biological and behavioral characteristics
 0488 [ISOBiometricVocabulary].
 0489
 0490 **Ceremony**
 0491 The concept of a ceremony [Ceremony] is an extension of the
 0492 concept of a network protocol, with human nodes alongside

0483 **Attestation**
 0484 Generally, attestation is a statement serving to bear witness,
 0485 confirm, or authenticate. In the WebAuthn context, attestation
 0486 is employed to attest to the provenance of an authenticator and
 0487 the data it emits; including, for example: credential IDs,
 0488 credential key pairs, signature counters, etc. An attestation
 0489 statement is conveyed in an attestation object during
 0490 registration. See also 6.3 Attestation and Figure 3. Whether or
 0491 how the client platform conveys the attestation statement and
 0492 AAGUID portions of the attestation object to the Relying Party
 0493 is described by attestation conveyance.
 0494
 0495 **Attestation Certificate**
 0496 A X.509 Certificate for the attestation key pair used by an
 0497 authenticator to attest to its manufacture and capabilities. At
 0498 registration time, the authenticator uses the attestation
 0499 private key to sign the Relying Party-specific credential public
 0500 key (and additional data) that it generates and returns via the
 0501 authenticatorMakeCredential operation. Relying Parties use the
 0502 attestation public key conveyed in the attestation certificate
 0503 to verify the attestation signature. Note that in the case of
 0504 self attestation, the authenticator has no distinct attestation
 0505 key pair nor attestation certificate, see self attestation for
 0506 details.
 0507
 0508 **Authentication**
 0509 The ceremony where a user, and the user's computing device(s)
 0510 (containing at least one authenticator) work in concert to
 0511 cryptographically prove to a Relying Party that the user
 0512 controls the credential private key associated with a
 0513 previously-registered public key credential (see Registration).
 0514 Note that this includes a test of user presence or user
 0515 verification.
 0516
 0517 **Authentication Assertion**
 0518 The cryptographically signed AuthenticatorAssertionResponse
 0519 object returned by an authenticator as the result of an
 0520 authenticatorGetAssertion operation.
 0521
 0522 This corresponds to the [CREDENTIAL-MANAGEMENT-1]
 0523 specification's single-use credentials.
 0524
 0525 **Authenticator**
 0526 A cryptographic entity used by a WebAuthn Client to (i) generate
 0527 a public key credential and register it with a Relying Party,
 0528 and (ii) authenticate by potentially verifying the user, and
 0529 then cryptographically signing and returning, in the form of an
 0530 Authentication Assertion, a challenge and other data presented
 0531 by a Relying Party (in concert with the WebAuthn Client).
 0532
 0533 **Authorization Gesture**
 0534 An authorization gesture is a physical interaction performed by
 0535 a user with an authenticator as part of a ceremony, such as
 0536 registration or authentication. By making such an authorization
 0537 gesture, a user provides consent for (i.e., authorizes) a
 0538 ceremony to proceed. This **MAY** involve user verification if the
 0539 employed authenticator is capable, or it **MAY** involve a simple
 0540 test of user presence.
 0541
 0542 **Biometric Recognition**
 0543 The automated recognition of individuals based on their
 0544 biological and behavioral characteristics
 0545 [ISOBiometricVocabulary].
 0546
 0547 **Biometric Authenticator**
 0548 Any authenticator that implements biometric recognition.
 0549
 0550 **Ceremony**
 0551 The concept of a ceremony [Ceremony] is an extension of the
 0552 concept of a network protocol, with human nodes alongside

0493 computer nodes and with communication links that include user
 0494 interface(s), human-to-human communication, and transfers of
 0495 physical objects that carry data. What is out-of-band to a
 0496 protocol is in-band to a ceremony. In this specification,
 0497 Registration and Authentication are ceremonies, and an
 0498 authorization gesture is often a component of those ceremonies.
 0499

0500 **Client**
 0501 See Conforming User Agent.

0502 **Client-Side**
 0503 This refers in general to the combination of the user's platform
 0504 device, user agent, authenticators, and everything gluing it all
 0505 together.
 0506

0507 **Client-side-resident Credential Private Key**
 0508 A Client-side-resident Credential Private Key is stored either
 0509 on the client platform, or in some cases on the authenticator
 0510 itself, e.g., in the case of a discrete first-factor roaming
 0511 authenticator. Such client-side credential private key storage
 0512 has the property that the authenticator is able to select the
 0513 credential private key given only an RP ID, possibly with user
 0514 assistance (e.g., by providing the user a pick list of
 0515 credentials associated with the RP ID). By definition, the
 0516 private key is always exclusively controlled by the
 0517 Authenticator. In the case of a Client-side-resident Credential
 0518 Private Key, the Authenticator might offload storage of wrapped
 0519 key material to the client platform, but the client platform is
 0520 not expected to offload the key storage to remote entities (e.g.
 0521 RP Server).
 0522

0523 **Conforming User Agent**
 0524 A user agent implementing, in conjunction with the underlying
 0525 platform, the Web Authentication API and algorithms given in
 0526 this specification, and handling communication between
 0527 authenticators and Relying Parties.
 0528

0529 **Credential ID**
 0530 A probabilistically-unique byte sequence identifying a public
 0531 key credential source and its authentication assertions.
 0532
 0533 Credential IDs are generated by authenticators in two forms:
 0534
 0535 1. At least 16 bytes that include at least 100 bits of entropy,
 0536 or
 0537 2. The public key credential source, without its Credential ID,
 0538 encrypted so only its managing authenticator can decrypt it.
 0539 This form allows the authenticator to be nearly stateless, by
 0540 having the Relying Party store any necessary state.
 0541 Note: [FIDO-UAF-AUTHNR-CMDS] includes guidance on encryption
 0542 techniques under "Security Guidelines".
 0543

0544 Relying Parties do not need to distinguish these two Credential
 0545 ID forms.
 0546

0547 **Credential Public Key**
 0548 The public key portion of an Relying Party-specific credential
 0549 key pair, generated by an authenticator and returned to an
 0550 Relying Party at registration time (see also public key
 0551 credential). The private key portion of the credential key pair
 0552 is known as the credential private key. Note that in the case of
 0553 self attestation, the credential key pair is also used as the
 0554 attestation key pair, see self attestation for details.
 0555

0556

0553 computer nodes and with communication links that include user
 0554 interface(s), human-to-human communication, and transfers of
 0555 physical objects that carry data. What is out-of-band to a
 0556 protocol is in-band to a ceremony. In this specification,
 0557 Registration and Authentication are ceremonies, and an
 0558 authorization gesture is often a component of those ceremonies.
 0559

0560 **Client**
 0561 See [WebAuthn Client](#), Conforming User Agent.

0562 **Client-Side**
 0563 This refers in general to the combination of the user's platform
 0564 device, user agent, authenticators, and everything gluing it all
 0565 together.
 0566

0567 **Client-side-resident Credential Private Key**
 0568 A Client-side-resident Credential Private Key is stored either
 0569 on the client platform, or in some cases on the authenticator
 0570 itself, e.g., in the case of a discrete first-factor roaming
 0571 authenticator. Such client-side credential private key storage
 0572 has the property that the authenticator is able to select the
 0573 credential private key given only an RP ID, possibly with user
 0574 assistance (e.g., by providing the user a pick list of
 0575 credentials associated with the RP ID). By definition, the
 0576 private key is always exclusively controlled by the
 0577 Authenticator. In the case of a Client-side-resident Credential
 0578 Private Key, the Authenticator might offload storage of wrapped
 0579 key material to the client platform, but the client platform is
 0580 not expected to offload the key storage to remote entities (e.g.
 0581 RP Server).
 0582

0583 **Conforming User Agent**
 0584 A user agent implementing, in conjunction with the underlying
 0585 platform, the Web Authentication API and algorithms given in
 0586 this specification, and handling communication between
 0587 authenticators and Relying Parties.
 0588

0589 **Credential ID**
 0590 A probabilistically-unique byte sequence identifying a public
 0591 key credential source and its authentication assertions.
 0592
 0593 Credential IDs are generated by authenticators in two forms:
 0594
 0595 1. At least 16 bytes that include at least 100 bits of entropy,
 0596 or
 0597 2. The public key credential source, without its Credential ID,
 0598 encrypted so only its managing authenticator can decrypt it.
 0599 This form allows the authenticator to be nearly stateless, by
 0600 having the Relying Party store any necessary state.
 0601 Note: [FIDO-UAF-AUTHNR-CMDS] includes guidance on encryption
 0602 techniques under "Security Guidelines".
 0603

0604 Relying Parties do not need to distinguish these two Credential
 0605 ID forms.
 0606

0607 **Credential Public Key**
 0608 The public key portion of a Relying Party-specific credential
 0609 key pair, generated by an authenticator and returned to a
 0610 Relying Party at registration time (see also public key
 0611 credential). The private key portion of the credential key pair
 0612 is known as the credential private key. Note that in the case of
 0613 self attestation, the credential key pair is also used as the
 0614 attestation key pair, see self attestation for details.
 0615

0616 **Human Palatability**
 0617 An identifier that is human-palatable is intended to be
 0618 rememberable and reproducible by typical human users, in
 0619 contrast to identifiers that are, for example, randomly
 0620 generated sequences of bits [EduPersonObjectClassSpec].
 0621

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**:

- 0561 + A Credential ID.
- 0562 + A credential private key.
- 0563 + The Relying Party Identifier for the Relying Party that
- 0564 created this credential source.
- 0565 + An optional user handle for the person who created this
- 0566 credential source.
- 0567 + Optional other information used by the authenticator to inform
- 0568 its UI. For example, this might include the user's
- 0569

0570 displayName.

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

0577
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),
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 "Authentication information" (possibly including a private key)
0598 Public key credential source

0600 "Signed value"
0601 Authentication assertion

0602 [RFC4949] "credential"
0603 Credential public key or attestation object

0604 At registration time, the authenticator creates an asymmetric
0605 key pair, and stores its private key portion and information
0606 from the Relying Party into a public key credential source. The
0607 public key portion is returned to the Relying Party, who then
0608 stores it in conjunction with the present user's account.
0609 Subsequently, only that Relying Party, as identified by its RP
0610
0611
0612

0623 Public Key Credential Source
0624 A credential source ([CREDENTIAL-MANAGEMENT-1]) used by an
0625 authenticator to generate authentication assertions. A public
0626 key credential source **consists of a struct with the following**
0627 **items:**

0628
0629 **type**
0630 whose value is of PublicKeyCredentialType, defaulting to
0631 public-key.

0632
0633 **id**
0634 A Credential ID.

0635
0636 **privateKey**
0637 The credential private key.

0638
0639 **rpld**
0640 The Relying Party Identifier, for the Relying Party this
0641 public key credential source is associated with.

0642
0643 **userHandle**
0644 The user handle associated when this public key credential
0645 source was created. This item is nullable.

0646
0647 **otherUI**
0648 Optional other information used by the authenticator to
0649 inform its UI. For example, this might include the user's
0650 displayName.

0651 The authenticatorMakeCredential operation creates a public key
0652 credential source bound to a managing authenticator and returns
0653 the credential public key associated with its credential private
0654 key. The Relying Party can use this credential public key to
0655 verify the authentication assertions created by this public key
0656 credential source.

0657
0658
0659 **Public Key Credential**

0660 Generically, a credential is data one entity presents to another
0661 in order to authenticate the former to the latter [RFC4949]. The
0662 term public key credential refers to one of: a public key
0663 credential source, the possibly-attested credential public key
0664 corresponding to a public key credential source, or an
0665 authentication assertion. Which one is generally determined by
0666 context.

0667
0668 Note: This is a willful violation of [RFC4949]. In English, a
0669 "credential" is both a) the thing presented to prove a statement
0670 and b) intended to be used multiple times. It's impossible to
0671 achieve both criteria securely with a single piece of data in a
0672 public key system. [RFC4949] chooses to define a credential as
0673 the thing that can be used multiple times (the public key),
0674 while this specification gives "credential" the English term's
0675 flexibility. This specification uses more specific terms to
0676 identify the data related to an [RFC4949] credential:

0677 "Authentication information" (possibly including a private key)
0678 Public key credential source

0679 "Signed value"
0680 Authentication assertion

0681 [RFC4949] "credential"
0682 Credential public key or attestation object

0683 At registration time, the authenticator creates an asymmetric
0684 key pair, and stores its private key portion and information
0685 from the Relying Party into a public key credential source. The
0686 public key portion is returned to the Relying Party, who then
0687 stores it in conjunction with the present user's account.
0688 Subsequently, only that Relying Party, as identified by its RP
0689
0690
0691
0692

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:
 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
 0677 **Test of User Presence**
 0678 A test of user presence is a simple form of authorization
 0679 gesture and technical process where a user interacts with an
 0680 authenticator by (typically) simply touching it (other
 0681 modalities may also exist), yielding a boolean result. Note that
 0682 this does not constitute user verification because a user

0693 ID, is able to employ the public key credential in
 0694 authentication ceremonies, via the get() method. The Relying
 0695 Party uses its stored copy of the credential public key to
 0696 verify the resultant authentication assertion.
 0697
 0698 **Rate Limiting**
 0699 The process (also known as throttling) by which an authenticator
 0700 implements controls against brute force attacks by limiting the
 0701 number of consecutive failed authentication attempts within a
 0702 given period of time. If the limit is reached, the authenticator
 0703 should impose a delay that increases exponentially with each
 0704 successive attempt, or disable the current authentication
 0705 modality and offer a different authentication factor if
 0706 available. Rate limiting is often implemented as an aspect of
 0707 user verification.
 0708
 0709 **Registration**
 0710 The ceremony where a user, a Relying Party, and the user's
 0711 computing device(s) (containing at least one authenticator) work
 0712 in concert to create a public key credential and associate it
 0713 with the user's Relying Party account. Note that this includes
 0714 employing a test of user presence or user verification.
 0715
 0716 **Relying Party**
 0717 The entity whose web application utilizes the Web Authentication
 0718 API to register and authenticate users. See Registration and
 0719 Authentication, respectively.
 0720
 0721 Note: While the term Relying Party is used in other contexts
 0722 (e.g., X.509 and OAuth), an entity acting as a Relying Party in
 0723 one context is not necessarily a Relying Party in other
 0724 contexts.
 0725
 0726 **Relying Party Identifier**
 0727 **RP ID**
 0728 A valid domain string that identifies the Relying Party on whose
 0729 behalf a given registration or authentication ceremony is being
 0730 performed. A public key credential can only be used for
 0731 authentication with the same entity (as identified by RP ID) it
 0732 was registered with. By default, the RP ID for a WebAuthn
 0733 operation is set to the caller's origin's effective domain. This
 0734 default MAY be overridden by the caller, as long as the
 0735 caller-specified RP ID value is a registrable domain suffix of
 0736 or is equal to the caller's origin's effective domain. See also
 0737 5.1.3 Create a new credential - PublicKeyCredential's
 0738 [[Create]](origin, options, sameOriginWithAncestors) method and
 0739 5.1.4 Use an existing credential to make an assertion -
 0740 PublicKeyCredential's [[Get]](options) method.
 0741
 0742 Note: A Public key credential's scope is for a Relying Party's
 0743 origin, with the following restrictions and relaxations:
 0744
 0745 + The scheme is always https (i.e., a restriction), and,
 0746 + the host may be equal to the Relying Party's origin's
 0747 effective domain, or it may be equal to a registrable domain
 0748 suffix of the Relying Party's origin's effective domain (i.e.,
 0749 an available relaxation), and,
 0750 + all (TCP) ports on that host (i.e., a relaxation).
 0751
 0752 This is done in order to match the behavior of pervasively
 0753 deployed ambient credentials (e.g., cookies, [RFC6265]). Please
 0754 note that this is a greater relaxation of "same-origin"
 0755 restrictions than what document.domain's setter provides.
 0756
 0757 **Test of User Presence**
 0758 A test of user presence is a simple form of authorization
 0759 gesture and technical process where a user interacts with an
 0760 authenticator by (typically) simply touching it (other
 0761 modalities may also exist), yielding a boolean result. Note that
 0762 this does not constitute user verification because a user

0683 presence test, by definition, is not capable of biometric
0684 recognition, nor does it involve the presentation of a shared
0685 secret such as a password or PIN.

0686 **User Consent**
0687 User consent means the user agrees with what they are being
0688 asked, i.e., it encompasses reading and understanding prompts.
0689 An authorization gesture is a ceremony component often employed
0690 to indicate user consent.

0691 **User Handle**
0692 The user handle is specified by a Relying Party and is a unique
0693 identifier for a user account with that Relying Party. A user
0694 handle is an opaque byte sequence with a maximum size of 64
0695 bytes.

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 -
0700 an authenticator will never contain more than one credential for
0701 a given Relying Party under the same user handle.

0702 **User Verification**
0703 The technical process by which an authenticator locally
0704 authorizes the invocation of the authenticatorMakeCredential and
0705 authenticatorGetAssertion operations. User verification **may** be
0706 instigated through various authorization gesture modalities; for
0707 example, through a touch plus pin code, password entry, or
0708 biometric recognition (e.g., presenting a fingerprint)
0709 [ISOBiometricVocabulary]. The intent is to be able to
0710 distinguish individual users. Note that invocation of the
0711 authenticatorMakeCredential and authenticatorGetAssertion
0712 operations implies use of key material managed by the
0713 authenticator. Note that for security, user verification and use
0714 of credential private keys must occur within a single logical
0715 security boundary defining the authenticator.

0716 **User Present**
0717 **UP**
0718 Upon successful completion of a user presence test, the user is
0719 said to be "present".

0720 **User Verified**
0721 **UV**
0722 Upon successful completion of a user verification process, the
0723 user is said to be "verified".

0724 **WebAuthn Client**
0725 Also referred to herein as simply a client. See also Conforming
0726 User Agent.

0732 5. Web Authentication API

0733 This section normatively specifies the API for creating and using
0734 public key credentials. The basic idea is that the credentials belong
0735 to the user and are managed by an authenticator, with which the Relying
0736 Party interacts through the client (consisting of the browser and
0737 underlying OS platform). Scripts can (with the user's consent) request
0738 the browser to create a new credential for future use by the Relying
0739 Party. Scripts can also request the user's permission to perform
0740 authentication operations with an existing credential. All such
0741 operations are performed in the authenticator and are mediated by the

0763 presence test, by definition, is not capable of biometric
0764 recognition, nor does it involve the presentation of a shared
0765 secret such as a password or PIN.

0766 **User Consent**
0767 User consent means the user agrees with what they are being
0768 asked, i.e., it encompasses reading and understanding prompts.
0769 An authorization gesture is a ceremony component often employed
0770 to indicate user consent.

0771 **User Handle**
0772 The user handle is specified by a Relying Party and is a unique
0773 identifier for a user account with that Relying Party. A user
0774 handle is an opaque byte sequence with a maximum size of 64
0775 bytes.

0776 The user handle is not meant to be displayed to the user, but is
0777 used by the Relying Party to control the number of credentials -
0780 an authenticator will never contain more than one credential for
0781 a given Relying Party under the same user handle.

0782 **User Verification**
0783 The technical process by which an authenticator locally
0784 authorizes the invocation of the authenticatorMakeCredential and
0785 authenticatorGetAssertion operations. User verification **MAY** be
0786 instigated through various authorization gesture modalities; for
0787 example, through a touch plus pin code, password entry, or
0788 biometric recognition (e.g., presenting a fingerprint)
0789 [ISOBiometricVocabulary]. The intent is to be able to
0790 distinguish individual users. Note that invocation of the
0791 authenticatorMakeCredential and authenticatorGetAssertion
0792 operations implies use of key material managed by the
0793 authenticator. Note that for security, user verification and use
0794 of credential private keys must occur within a single logical
0795 security boundary defining the authenticator.

0796 **User Present**
0797 **UP**
0798 Upon successful completion of a user presence test, the user is
0799 said to be "present".

0800 **User Verified**
0801 **UV**
0802 Upon successful completion of a user verification process, the
0803 user is said to be "verified".

0804 **WebAuthn Client**
0805 Also referred to herein as simply a client. See also Conforming
0806 User Agent. **A WebAuthn Client is an intermediary entity**
0807 **typically implemented in the user agent (in whole, or in part).**
0808 **Conceptually, it underlies the Web Authentication API and**
0809 **embodies the implementation of the [[Create]](origin, options,**
0810 **sameOriginWithAncestors) and**
0811 **[[DiscoverFromExternalSource]](origin, options,**
0812 **sameOriginWithAncestors) internal methods. It is responsible for**
0813 **both marshalling the inputs for the underlying authenticator**
0814 **operations, and for returning the results of the latter**
0815 **operations to the Web Authentication API's callers.**

0821 5. Web Authentication API

0822 This section normatively specifies the API for creating and using
0823 public key credentials. The basic idea is that the credentials belong
0824 to the user and are managed by an authenticator, with which the Relying
0825 Party interacts through the client (consisting of the browser and
0826 underlying OS platform). Scripts can (with the user's consent) request
0827 the browser to create a new credential for future use by the Relying
0828 Party. Scripts can also request the user's permission to perform
0829 authentication operations with an existing credential. All such
0830 operations are performed in the authenticator and are mediated by the

0744 browser and/or platform on the user's behalf. At no point does the
 0745 script get access to the credentials themselves; it only gets
 0746 information about the credentials in the form of objects.
 0747
 0748 In addition to the above script interface, the authenticator **may**
 0749 implement (or come with client software that implements) a user
 0750 interface for management. Such an interface **may** be used, for example,
 0751 to reset the authenticator to a clean state or to inspect the current
 0752 state of the authenticator. In other words, such an interface is
 0753 similar to the user interfaces provided by browsers for managing user
 0754 state such as history, saved passwords and cookies. Authenticator
 0755 management actions such as credential deletion are considered to be the
 0756 responsibility of such a user interface and are deliberately omitted
 0757 from the API exposed to scripts.
 0758
 0759 The security properties of this API are provided by the client and the
 0760 authenticator working together. The authenticator, which holds and
 0761 manages credentials, ensures that all operations are scoped to a
 0762 particular origin, and cannot be replayed against a different origin,
 0763 by incorporating the origin in its responses. Specifically, as defined
 0764 in 6.2 Authenticator operations, the full origin of the requester is
 0765 included, and signed over, in the attestation object produced when a
 0766 new credential is created as well as in all assertions produced by
 0767 WebAuthn credentials.
 0768
 0769 Additionally, to maintain user privacy and prevent malicious Relying
 0770 Parties from probing for the presence of public key credentials
 0771 belonging to other Relying Parties, each credential is also associated
 0772 with a Relying Party Identifier, or RP ID. This RP ID is provided by
 0773 the client to the authenticator for all operations, and the
 0774 authenticator ensures that credentials created by a Relying Party can
 0775 only be used in operations requested by the same RP ID. Separating the
 0776 origin from the RP ID in this way allows the API to be used in cases
 0777 where a single Relying Party maintains multiple origins.
 0778
 0779 The client facilitates these security measures by providing the Relying
 0780 Party's origin and RP ID to the authenticator for each operation. Since
 0781 this is an integral part of the WebAuthn security model, user agents
 0782 only expose this API to callers in secure contexts.
 0783
 0784 The Web Authentication API is defined by the union of the Web IDL
 0785 fragments presented in the following sections. A combined IDL listing
 0786 is given in the IDL Index.
 0787
 0788 **5.1. PublicKeyCredential Interface**
 0789
 0790 The PublicKeyCredential interface inherits from Credential
 0791 [CREDENTIAL-MANAGEMENT-1], and contains the attributes that are
 0792 returned to the caller when a new credential is created, or a new
 0793 assertion is requested.
 0794 [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

0833 browser and/or platform on the user's behalf. At no point does the
 0834 script get access to the credentials themselves; it only gets
 0835 information about the credentials in the form of objects.
 0836
 0837 In addition to the above script interface, the authenticator **MAY**
 0838 implement (or come with client software that implements) a user
 0839 interface for management. Such an interface **MAY** be used, for example,
 0840 to reset the authenticator to a clean state or to inspect the current
 0841 state of the authenticator. In other words, such an interface is
 0842 similar to the user interfaces provided by browsers for managing user
 0843 state such as history, saved passwords, and cookies. Authenticator
 0844 management actions such as credential deletion are considered to be the
 0845 responsibility of such a user interface and are deliberately omitted
 0846 from the API exposed to scripts.
 0847
 0848 The security properties of this API are provided by the client and the
 0849 authenticator working together. The authenticator, which holds and
 0850 manages credentials, ensures that all operations are scoped to a
 0851 particular origin, and cannot be replayed against a different origin,
 0852 by incorporating the origin in its responses. Specifically, as defined
 0853 in 6.2 Authenticator operations, the full origin of the requester is
 0854 included, and signed over, in the attestation object produced when a
 0855 new credential is created as well as in all assertions produced by
 0856 WebAuthn credentials.
 0857
 0858 Additionally, to maintain user privacy and prevent malicious Relying
 0859 Parties from probing for the presence of public key credentials
 0860 belonging to other Relying Parties, each credential is also associated
 0861 with a Relying Party Identifier, or RP ID. This RP ID is provided by
 0862 the client to the authenticator for all operations, and the
 0863 authenticator ensures that credentials created by a Relying Party can
 0864 only be used in operations requested by the same RP ID. Separating the
 0865 origin from the RP ID in this way allows the API to be used in cases
 0866 where a single Relying Party maintains multiple origins.
 0867
 0868 The client facilitates these security measures by providing the Relying
 0869 Party's origin and RP ID to the authenticator for each operation. Since
 0870 this is an integral part of the WebAuthn security model, user agents
 0871 only expose this API to callers in secure contexts.
 0872
 0873 The Web Authentication API is defined by the union of the Web IDL
 0874 fragments presented in the following sections. A combined IDL listing
 0875 is given in the IDL Index.
 0876
 0877 **5.1. PublicKeyCredential Interface**
 0878
 0879 The PublicKeyCredential interface inherits from Credential
 0880 [CREDENTIAL-MANAGEMENT-1], and contains the attributes that are
 0881 returned to the caller when a new credential is created, or a new
 0882 assertion is requested.
 0883 [SecureContext, Exposed=Window]
 0884 interface PublicKeyCredential : Credential {
 0885 [SameObject] readonly attribute ArrayBuffer rawId;
 0886 [SameObject] readonly attribute AuthenticatorResponse response;
 0887 AuthenticationExtensionsClientOutputs getClientExtensionResults();
 0888 };
 0889
 0890 id
 0891 This attribute is inherited from Credential, though
 0892 PublicKeyCredential overrides Credential's getter, instead
 0893 returning the base64url encoding of the data contained in the
 0894 object's [[identifier]] internal slot.
 0895
 0896 rawId
 0897 This attribute returns the ArrayBuffer contained in the
 0898 [[identifier]] internal slot.
 0899
 0900 response, of type AuthenticatorResponse, readonly
 0901 This attribute contains the authenticator's response to the
 0902 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
 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

0903 generate an authentication assertion. If the PublicKeyCredential
 0904 is created in response to create(), this attribute's value will
 0905 be an AuthenticatorAttestationResponse, otherwise, the
 0906 PublicKeyCredential was created in response to get(), and this
 0907 attribute's value will be an AuthenticatorAssertionResponse.
 0908
 0909 getClientExtensionResults()
 0910 This operation returns the value of [[clientExtensionsResults]],
 0911 which is a map containing extension identifier -> client
 0912 extension output entries produced by the extension's client
 0913 extension processing.
 0914
 0915 [[type]]
 0916 The PublicKeyCredential interface object's [[type]] internal
 0917 slot's value is the string "public-key".
 0918
 0919 Note: This is reflected via the type attribute getter inherited
 0920 from Credential.
 0921
 0922 [[discovery]]
 0923 The PublicKeyCredential interface object's [[discovery]]
 0924 internal slot's value is "remote".
 0925
 0926 [[identifier]]
 0927 This internal slot contains **the credential ID, chosen by the**
 0928 **platform with help from the authenticator. The credential ID is**
 0929 **used to look up credentials for use, and is therefore expected**
 0930 **to be globally unique with high probability across all**
 0931 **credentials of the same type, across all authenticators.**
 0932
 0933 Note: This API does not constrain the format or length of this
 0934 identifier, except that it **MUST** be sufficient for the platform
 0935 to uniquely select a key. For example, an authenticator without
 0936 on-board storage may create identifiers containing a credential
 0937 private key wrapped with a symmetric key that is burned into the
 0938 authenticator.
 0939
 0940 [[clientExtensionsResults]]
 0941 This internal slot contains the results of processing client
 0942 extensions requested by the Relying Party upon the Relying
 0943 Party's invocation of either navigator.credentials.create() or
 0944 navigator.credentials.get().
 0945
 0946 PublicKeyCredential's interface object inherits Credential's
 0947 implementation of [[CollectFromCredentialStore]](origin, options,
 0948 sameOriginWithAncestors), and defines its own implementation of
 0949 [[Create]](origin, options, sameOriginWithAncestors),
 0950 [[DiscoverFromExternalSource]](origin, options,
 0951 sameOriginWithAncestors), and [[Store]](credential,
 0952 sameOriginWithAncestors).
 0953
 0954 5.1.1. CredentialCreationOptions Dictionary Extension
 0955
 0956 To support registration via navigator.credentials.create(), this
 0957 document extends the CredentialCreationOptions dictionary as follows:
 0958 partial dictionary CredentialCreationOptions {
 0959 PublicKeyCredentialCreationOptions publicKey;
 0960 };
 0961
 0962 5.1.2. CredentialRequestOptions Dictionary Extension
 0963
 0964 To support obtaining assertions via navigator.credentials.get(), this
 0965 document extends the CredentialRequestOptions dictionary as follows:
 0966 partial dictionary CredentialRequestOptions {
 0967 PublicKeyCredentialRequestOptions publicKey;
 0968 };
 0969
 0970 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin,
 0971 options, sameOriginWithAncestors) method
 0972

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
 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

0973 PublicKeyCredential's interface object's implementation of the
 0974
 0975 [[Create]](origin, options, sameOriginWithAncestors) internal method
 0976 [CREDENTIAL-MANAGEMENT-1] allows Relying Party scripts to call
 0977 navigator.credentials.create() to request the creation of a new public
 0978 key credential source, bound to an authenticator. This
 0979 navigator.credentials.create() operation can be aborted by leveraging
 0980 the AbortController; see DOM 3.3 Using AbortController and AbortSignal
 0981 objects in APIs for detailed instructions.
 0982
 0983 This internal method accepts three arguments:
 0984
 0985 origin
 0986 This argument is the relevant settings object's origin, as
 0987 determined by the calling create() implementation.
 0988
 0989 options
 0990 This argument is a CredentialCreationOptions object whose
 0991 options.publicKey member contains a
 0992 PublicKeyCredentialCreationOptions object specifying the desired
 0993 attributes of the to-be-created public key credential.
 0994
 0995 sameOriginWithAncestors
 0996 This argument is a boolean which is true if and only if the
 0997 caller's environment settings object is same-origin with its
 0998 ancestors.
 0999
 1000 Note: This algorithm is synchronous: the Promise resolution/rejection
 1001 is handled by navigator.credentials.create().
 1002
 1003 When this method is invoked, the user agent MUST execute the following
 1004 algorithm:
 1005 1. Assert: options.publicKey is present.
 1006 2. If sameOriginWithAncestors is false, return a "NotAllowedError"
 1007 DOMException.
 1008 Note: This "sameOriginWithAncestors" restriction aims to address
 1009 the concern raised in the Origin Confusion section of
 1010 [CREDENTIAL-MANAGEMENT-1], while allowing Relying Party script
 1011 access to Web Authentication functionality, e.g., when running in a
 1012 secure context framed document that is same-origin with its
 1013 ancestors. However, in the future, this specification (in
 1014 conjunction with [CREDENTIAL-MANAGEMENT-1]) may provide Relying
 1015 Parties with more fine-grained control--e.g., ranging from allowing
 1016 only top-level access to Web Authentication functionality, to
 1017 allowing cross-origin embedded cases--by leveraging
 1018 [Feature-Policy] once the latter specification becomes stably
 1019 implemented in user agents.
 1020 3. Let options be the value of options.publicKey.
 1021 4. If the timeout member of options is present, check if its value
 1022 lies within a reasonable range as defined by the platform and if
 1023 not, correct it to the closest value lying within that range. Set a
 1024 timer lifetimeTimer to this adjusted value. If the timeout member
 1025 of options is not present, then set lifetimeTimer to a
 1026 platform-specific default.
 1027 5. Let callerOrigin be origin. If callerOrigin is an opaque origin,
 1028 return a DOMException whose name is "NotAllowedError", and
 1029 terminate this algorithm.
 1030 6. Let effectiveDomain be the callerOrigin's effective domain. If
 1031 effective domain is not a valid domain, then return a DOMException
 1032 whose name is "SecurityError" and terminate this algorithm.
 1033 Note: An effective domain may resolve to a host, which can be
 1034 represented in various manners, such as domain, ipv4 address, ipv6
 1035 address, opaque host, or empty host. Only the domain format of host
 1036 is allowed here.
 1037 7. If options.rp.id
 1038
 1039 is present
 1040 If options.rp.id is not a registrable domain suffix of and
 1041 is not equal to effectiveDomain, return a DOMException
 1042 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 type
 0990 The string "webauthn.create".
 0991 challenge
 0992 The base64url encoding of options.challenge.
 0993 origin
 0994 The serialization of callerOrigin.
 0995 hashAlgorithm
 0996 The recognized algorithm name of the hash algorithm
 0997 selected by the client for generating the hash of the
 0998 serialized client data.
 1003 tokenBindingId
 1004 The Token Binding ID associated with callerOrigin, if one
 1005 is available.
 1006 clientExtensions
 1007 clientExtensions
 1010 authenticatorExtensions
 1011 authenticatorExtensions
 1012
 1013 14. Let clientDataJSON be the JSON-serialized client data constructed
 1014 from collectedClientData.
 1015 15. Let clientDataHash be the hash of the serialized client data
 1016 represented by clientDataJSON.
 1017 16. If the options.signal is present and its aborted flag is set to
 1018 true, return a DOMException whose name is "AbortError" and
 1019 terminate this algorithm.
 1020 17. Start lifetimeTimer.
 1021 18. Let issuedRequests be a new ordered set.
 1022

1043 algorithm.
 1044
 1045 Is not present
 1046 Set options rp.id to effectiveDomain.
 1047
 1048 Note: options rp.id represents the caller's RP ID. The RP ID
 1049 defaults to being the caller's origin's effective domain unless the
 1050 caller has explicitly set options rp.id when calling create().
 1051 8. Let credTypesAndPubKeyAlgs be a new list whose items are pairs of
 1052 PublicKeyCredentialType and a COSEAlgorithmIdentifier.
 1053 9. For each current of options pubKeyCredParams:
 1054 1. If current.type does not contain a PublicKeyCredentialType
 1055 supported by this implementation, then continue.
 1056 2. Let alg be current.alg.
 1057 3. Append the pair of current.type and alg to
 1058 credTypesAndPubKeyAlgs.
 1059 10. If credTypesAndPubKeyAlgs is empty and options pubKeyCredParams is
 1060 not empty, return a DOMException whose name is "NotSupportedError",
 1061 and terminate this algorithm.
 1062 11. Let clientExtensions be a new map and let authenticatorExtensions
 1063 be a new map.
 1064 12. If the extensions member of options is present, then for each
 1065 extensionId -> clientExtensionInput of options.extensions:
 1066 1. If extensionId is not supported by this client platform or is
 1067 not a registration extension, then continue.
 1068 2. Set clientExtensions[extensionId] to clientExtensionInput.
 1069 3. If extensionId is not an authenticator extension, then
 1070 continue.
 1071 4. Let authenticatorExtensionInput be the (CBOR) result of
 1072 running extensionId's client extension processing algorithm on
 1073 clientExtensionInput. If the algorithm returned an error,
 1074 continue.
 1075 5. Set authenticatorExtensions[extensionId] to the base64url
 1076 encoding of authenticatorExtensionInput.
 1077 13. Let collectedClientData be a new CollectedClientData instance whose
 1078 fields are:
 1079 type
 1080 The string "webauthn.create".
 1081 challenge
 1082 The base64url encoding of options.challenge.
 1083 origin
 1084 The serialization of callerOrigin.
 1085 tokenBinding
 1086 The status of Token Binding between the client and the
 1087 callerOrigin, as well as the Token Binding ID associated
 1088 with callerOrigin, if one is available.
 1093
 1094 14. Let clientDataJSON be the JSON-serialized client data constructed
 1095 from collectedClientData.
 1096 15. Let clientDataHash be the hash of the serialized client data
 1097 represented by clientDataJSON.
 1098 16. If the options.signal is present and its aborted flag is set to
 1099 true, return a DOMException whose name is "AbortError" and
 1100 terminate this algorithm.
 1101 17. Start lifetimeTimer.
 1102 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 is set to preferred
 1047 If the authenticator
 1048 is capable of user verification
 1049 Let `userVerification` be true.
 1050 is not capable of user verification
 1051 Let `userVerification` be false.
 1052 is set to discouraged
 1053 Let `userVerification` be false.
 1054 3. Let `userPresence` be a Boolean value set to the inverse of
 1055 `userVerification`.
 1056 4. Let `excludeCredentialDescriptorList` be a new list.
 1057 5. For each credential descriptor C in
 1058 `options.excludeCredentials`:
 1059 1. If `C.transports` is not empty, and authenticator is
 1060 connected over a transport not mentioned in `C.transports`,
 1061 the client MAY continue.
 1062 2. Otherwise, Append C to `excludeCredentialDescriptorList`.
 1063 6. Invoke the `authenticatorMakeCredential` operation on
 1064 authenticator with `clientDataHash`, `options rp`, `options.user`,
 1065 `options.authenticatorSelection.requireResidentKey`,
 1066 `userPresence`, `userVerification`, `credTypesAndPubKeyAlgs`,
 1067 `excludeCredentialDescriptorList`, and `authenticatorExtensions`
 1068 as parameters.
 1069 7. Append authenticator to `issuedRequests`.
 1070 20. While `issuedRequests` is not empty, perform the following actions
 1071 depending upon `lifetimeTimer` and responses from the authenticators:
 1072 If `lifetimeTimer` expires,
 1073 For each authenticator in `issuedRequests` invoke the
 1074 `authenticatorCancel` operation on authenticator and remove
 1075 authenticator from `issuedRequests`.
 1076 If the `options.signal` is present and its aborted flag is set to
 1077 true,
 1078 For each authenticator in `issuedRequests` invoke the
 1079 `authenticatorCancel` operation on authenticator and remove
 1080 authenticator from `issuedRequests`. Then return a
 1081 `DOMException` whose name is "AbortError" and terminate this
 1082 algorithm.
 1083 If any authenticator returns a status indicating that the user

1103 19. For each authenticator that becomes available on this platform
 1104 during the lifetime of `lifetimeTimer`, do the following:
 1105 The definitions of "lifetime of" and "becomes available" are
 1106 intended to represent how devices are hot-plugged into (USB) or
 1107 discovered by (NFC) browsers, and are underspecified. Resolving
 1108 this with good definitions or some other means will be addressed by
 1109 resolving Issue #613.
 1110 1. If `options.authenticatorSelection` is present:
 1111 1. If `options.authenticatorSelection.authenticatorAttachment`
 1112 is present and its value is not equal to authenticator's
 1113 attachment modality, continue.
 1114 2. If `options.authenticatorSelection.requireResidentKey` is
 1115 set to true and the authenticator is not capable of
 1116 storing a Client-Side-Resident Credential Private Key,
 1117 continue.
 1118 3. If `options.authenticatorSelection.userVerification` is set
 1119 to required and the authenticator is not capable of
 1120 performing user verification, continue.
 1121 2. Let `userVerification` be the effective user verification
 1122 requirement for credential creation, a Boolean value, as
 1123 follows. If `options.authenticatorSelection.userVerification`
 1124 is set to required
 1125 Let `userVerification` be true.
 1126 is set to preferred
 1127 If the authenticator
 1128 is capable of user verification
 1129 Let `userVerification` be true.
 1130 is not capable of user verification
 1131 Let `userVerification` be false.
 1132 is set to discouraged
 1133 Let `userVerification` be false.
 1134 3. Let `userPresence` be a Boolean value set to the inverse of
 1135 `userVerification`.
 1136 4. Let `excludeCredentialDescriptorList` be a new list.
 1137 5. For each credential descriptor C in
 1138 `options.excludeCredentials`:
 1139 1. If `C.transports` is not empty, and authenticator is
 1140 connected over a transport not mentioned in `C.transports`,
 1141 the client MAY continue.
 1142 2. Otherwise, Append C to `excludeCredentialDescriptorList`.
 1143 6. Invoke the `authenticatorMakeCredential` operation on
 1144 authenticator with `clientDataHash`, `options rp`, `options.user`,
 1145 `options.authenticatorSelection.requireResidentKey`,
 1146 `userPresence`, `userVerification`, `credTypesAndPubKeyAlgs`,
 1147 `excludeCredentialDescriptorList`, and `authenticatorExtensions`
 1148 as parameters.
 1149 7. Append authenticator to `issuedRequests`.
 1150 20. While `lifetimeTimer` has not expired, perform the following actions
 1151 depending upon `lifetimeTimer` and responses from the authenticators:
 1152 If `lifetimeTimer` expires,
 1153 For each authenticator in `issuedRequests` invoke the
 1154 `authenticatorCancel` operation on authenticator and remove
 1155 authenticator from `issuedRequests`.
 1156 If the `options.signal` is present and its aborted flag is set to
 1157 true,
 1158 For each authenticator in `issuedRequests` invoke the
 1159 `authenticatorCancel` operation on authenticator and remove
 1160 authenticator from `issuedRequests`. Then return a
 1161 `DOMException` whose name is "AbortError" and terminate this
 1162 algorithm.
 1163 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 If any authenticator returns an error status,

1101 Remove authenticator from issuedRequests.
1102

1103 If any authenticator indicates success,

- 1104
1105 1. Remove authenticator from issuedRequests.
1106 2. Let credentialCreationData be a struct whose items are:

1107 attestationObjectResult
1108 whose value is the bytes returned from the
1109 successful authenticatorMakeCredential
1110 operation.
1111
1112

1113 Note: this value is attObj, as defined in
1114 6.3.4 Generating an Attestation Object.
1115

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 extension processing algorithm to create the
1129 client extension outputs, for each client
1130 extension in clientDataJSON.clientExtensions.
1131

- 1132 3. Let constructCredentialAlg be an algorithm that takes a
1133 global object global, and whose steps are:

1173 cancelled the operation,
1174
1175 1. Remove authenticator from issuedRequests.
1176 2. For each remaining authenticator in issuedRequests invoke
1177 the authenticatorCancel operation on authenticator and
1178 remove it from issuedRequests.

1179 Note: Authenticators may return an indication of "the
1180 user cancelled the entire operation". How a user agent
1181 manifests this state to users is unspecified.

1182
1183 If any authenticator returns an error status equivalent to
1184 "InvalidStateError",

- 1185
1186 1. Remove authenticator from issuedRequests.
1187 2. For each remaining authenticator in issuedRequests invoke
1188 the authenticatorCancel operation on authenticator and
1189 remove it from issuedRequests.
1190 3. Return a DOMException whose name is "InvalidStateError"
1191 and terminate this algorithm.
1192

1193 Note: This error status is handled separately because the
1194 authenticator returns it only if
1195 excludeCredentialDescriptorList identifies a credential
1196 bound to the authenticator and the user has consented to
1197 the operation. Given this explicit consent, it is
1198 acceptable for this case to be distinguishable to the
1199 Relying Party.

1200
1201 If any authenticator returns an error status not equivalent to
1202 "InvalidStateError",
1203 Remove authenticator from issuedRequests.
1204

1205 Note: This case does not imply user consent for the
1206 operation, so details about the error must be hidden from
1207 the Relying Party in order to prevent leak of potentially
1208 identifying information. See 14.2 Registration Ceremony
1209 Privacy for details.
1210

1211 If any authenticator indicates success,

- 1212
1213 1. Remove authenticator from issuedRequests.
1214 2. Let credentialCreationData be a struct whose items are:

1215 attestationObjectResult
1216 whose value is the bytes returned from the
1217 successful authenticatorMakeCredential
1218 operation.
1219
1220

1221 Note: this value is attObj, as defined in
1222 6.3.4 Generating an Attestation Object.
1223

1224 clientDataJSONResult
1225 whose value is the bytes of clientDataJSON.
1226

1227 attestationConveyancePreferenceOption
1228 whose value is the value of
1229 options.attestation.
1230

1231 clientExtensionResults
1232 whose value is an AuthenticationExtensionsClientOutputs object
1233 containing extension identifier -> client
1234 extension output entries. The entries are
1235 created by running each extension's client
1236 extension processing algorithm to create the
1237 client extension outputs, for each client
1238 extension in clientDataJSON.clientExtensions.
1239

- 1240
1241 3. Let constructCredentialAlg be an algorithm that takes a
1242 global object global, and whose steps are:

1134 1. Let attestationObject be a new ArrayBuffer, created
 1135 using global's %ArrayBuffer%, containing the bytes
 1136 of credentialCreationData.attestationObjectResult's
 1137 value.
 1138 2. If
 1139 credentialCreationData.attestationConveyancePreferen
 1140 ceOption's value is
 1141 "none"
 1142 Replace potentially uniquely identifying
 1143 information (such as AAGUID and
 1144 attestation certificates) in the
 1145 attested credential data and attestation
 1146 statement, respectively, with blinded
 1147 versions of the same data.
 1148
 1149 need to define "blinding". See also
 1150 #462.
 1151 <[https://github.com/w3c/webauthn/issues/
 1152 694](https://github.com/w3c/webauthn/issues/694)>
 1153
 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 case: If the authenticator violates the
 1168 privacy requirements of the attestation
 1169 type it is using, the client SHOULD
 1170 terminate this algorithm with a
 1171 "AttestationNotPrivateError".
 1172
 1173
 1174 3. Let id be
 1175 attestationObject.authData.attestedCredentialData.cr
 1176 edentialId.
 1177 4. Let pubKeyCred be a new PublicKeyCredential object
 1178 associated with global whose fields are:
 1179
 1180 [[identifier]]
 1181 id
 1182

1243 1. If
 1244 credentialCreationData.attestationConveyancePreferen
 1245 ceOption's value is
 1246 "none"
 1247 Replace potentially uniquely identifying
 1248 information with non-identifying
 1249 versions of the same:
 1250
 1251 1. If the AAGUID in the attested credential
 1252 data is 16 zero bytes,
 1253 credentialCreationData.attestationObjectRe
 1254 sult.fmt is "packed", and "x5c" &
 1255 "ecdaaKeyld" are both absent from
 1256 credentialCreationData.attestationObjectRe
 1257 sult, then self attestation is being used
 1258 and no further action is needed.
 1259 2. Otherwise
 1260 1. Replace the AAGUID in the attested
 1261 credential data with 16 zero bytes.
 1262 2. Set the value of
 1263 credentialCreationData.attestationObj
 1264 ectResult.fmt to "none", and set the
 1265 value of
 1266 credentialCreationData.attestationObj
 1267 ectResult.attStmt to be an empty CBOR
 1268 map. (See 8.7 None Attestation
 1269 Statement Format and 6.3.4
 1270 Generating an Attestation Object).
 1271
 1272 "indirect"
 1273 The client MAY replace the AAGUID and
 1274 attestation statement with a more
 1275 privacy-friendly and/or more easily
 1276 verifiable version of the same data (for
 1277 example, by employing an Anonymization
 1278 CA).
 1279
 1280 "direct"
 1281 Convey the authenticator's AAGUID and
 1282 attestation statement, unaltered, to the
 1283 RP.
 1284
 1285 @balfanz wishes to add to the "direct"
 1286 case: If the authenticator violates the
 1287 privacy requirements of the attestation
 1288 type it is using, the client SHOULD
 1289 terminate this algorithm with an
 1290 "AttestationNotPrivateError".
 1291
 1292
 1293 2. Let attestationObject be a new ArrayBuffer, created
 1294 using global's %ArrayBuffer%, containing the bytes
 1295 of credentialCreationData.attestationObjectResult's
 1296 value.
 1297 3. Let id be
 1298 attestationObject.authData.attestedCredentialData.cr
 1299 edentialId.
 1300 4. Let pubKeyCred be a new PublicKeyCredential object
 1301 associated with global whose fields are:
 1302
 1303 [[identifier]]
 1304 id
 1305

1183 response
 1184 A new AuthenticatorAttestationResponse
 1185 object associated with global whose
 1186 fields are:
 1187
 1188 clientDataJSON
 1189 A new ArrayBuffer, created using
 1190 global's %ArrayBuffer%, containing
 1191 the bytes of
 1192 credentialCreationData.clientDataJ
 1193 SONResult.
 1194
 1195 attestationObject
 1196 attestationObject
 1197
 1198 [[clientExtensionsResults]]
 1199 A new ArrayBuffer, created using
 1200 global's %ArrayBuffer%, containing the
 1201 bytes of
 1202 credentialCreationData.clientExtensionRe
 1203 sults.
 1204
 1205 5. Return pubKeyCred.
 1206 4. For each remaining authenticator in issuedRequests invoke
 1207 the authenticatorCancel operation on authenticator and
 1208 remove it from issuedRequests.
 1209 5. Return constructCredentialAlg and terminate this
 1210 algorithm.
 1211
 1212 21. Return a DOMException whose name is "NotAllowedError".

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 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,

1249

1306 response
 1307 A new AuthenticatorAttestationResponse
 1308 object associated with global whose
 1309 fields are:
 1310
 1311 clientDataJSON
 1312 A new ArrayBuffer, created using
 1313 global's %ArrayBuffer%, containing
 1314 the bytes of
 1315 credentialCreationData.clientDataJ
 1316 SONResult.
 1317
 1318 attestationObject
 1319 attestationObject
 1320
 1321 [[clientExtensionsResults]]
 1322 A new ArrayBuffer, created using
 1323 global's %ArrayBuffer%, containing the
 1324 bytes of
 1325 credentialCreationData.clientExtensionRe
 1326 sults.
 1327
 1328 5. Return pubKeyCred.
 1329 4. For each remaining authenticator in issuedRequests invoke
 1330 the authenticatorCancel operation on authenticator and
 1331 remove it from issuedRequests.
 1332 5. Return constructCredentialAlg and terminate this
 1333 algorithm.
 1334

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

1335
 1336 During the above process, the user agent SHOULD show some UI to the
 1337 user to guide them in the process of selecting and authorizing an
 1338 authenticator.
 1339

1340 5.1.4. Use an existing credential to make an assertion -
 1341 PublicKeyCredential's [[Get]](options) method
 1342

1343 Relying Parties call navigator.credentials.get({publicKey:..., ...}) to
 1344 discover and use an existing public key credential, with the user's
 1345 consent. Relying Party script optionally specifies some criteria to
 1346 indicate what credential sources are acceptable to it. The user agent
 1347 and/or platform locates credential sources matching the specified
 1348 criteria, and guides the user to pick one that the script will be
 1349 allowed to use. The user may choose to decline the entire interaction
 1350 even if a credential source is present, for example to maintain
 1351 privacy. If the user picks a credential source, the user agent then
 1352 uses 6.2.3 The authenticatorGetAssertion operation to sign a Relying
 1353 Party-provided challenge and other collected data into an assertion,
 1354 which is used as a credential.
 1355

1356 The get() implementation [CREDENTIAL-MANAGEMENT-1] calls
 1357 PublicKeyCredential.[[CollectFromCredentialStore]]() to collect any
 1358 credentials that should be available without user mediation (roughly,
 1359 this specification's authorization gesture), and if it does not find
 1360 exactly one of those, it then calls
 1361 PublicKeyCredential.[[DiscoverFromExternalSource]]() to have the user
 1362 select a credential source.
 1363

1364 Since this specification requires an authorization gesture to create
 1365 any credentials, the
 1366 PublicKeyCredential.[[CollectFromCredentialStore]](origin, options,
 1367 sameOriginWithAncestors) internal method inherits the default behavior
 1368 of Credential.[[CollectFromCredentialStore]](), of returning an empty
 1369 set.
 1370
 1371 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin,

1372

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
 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:

1376 options, sameOriginWithAncestors) method
 1377
 1378 This internal method accepts three arguments:
 1379
 1380 origin
 1381 This argument is the relevant settings object's origin, as
 1382 determined by the calling get() implementation, i.e.,
 1383 CredentialsContainer's Request a Credential abstract operation.
 1384
 1385 options
 1386 This argument is a CredentialRequestOptions object whose
 1387 options.publicKey member contains a
 1388 PublicKeyCredentialRequestOptions object specifying the desired
 1389 attributes of the public key credential to discover.
 1390
 1391 sameOriginWithAncestors
 1392 This argument is a boolean which is true if and only if the
 1393 caller's environment settings object is same-origin with its
 1394 ancestors.
 1395
 1396 Note: This algorithm is synchronous: the Promise resolution/rejection
 1397 is handled by navigator.credentials.get().
 1398
 1399 When this method is invoked, the user agent MUST execute the following
 1400 algorithm:
 1401 1. Assert: options.publicKey is present.
 1402 2. If sameOriginWithAncestors is false, return a "NotAllowedError"
 1403 DOMException.
 1404 Note: This "sameOriginWithAncestors" restriction aims to address
 1405 the concern raised in the Origin Confusion section of
 1406 [CREDENTIAL-MANAGEMENT-1], while allowing Relying Party script
 1407 access to Web Authentication functionality, e.g., when running in a
 1408 secure context framed document that is same-origin with its
 1409 ancestors. However, in the future, this specification (in
 1410 conjunction with [CREDENTIAL-MANAGEMENT-1]) may provide Relying
 1411 Parties with more fine-grained control--e.g., ranging from allowing
 1412 only top-level access to Web Authentication functionality, to
 1413 allowing cross-origin embedded cases--by leveraging
 1414 [Feature-Policy] once the latter specification becomes stably
 1415 implemented in user agents.
 1416 3. Let options be the value of options.publicKey.
 1417 4. If the timeout member of options is present, check if its value
 1418 lies within a reasonable range as defined by the platform and if
 1419 not, correct it to the closest value lying within that range. Set a
 1420 timer lifetimeTimer to this adjusted value. If the timeout member
 1421 of options is not present, then set lifetimeTimer to a
 1422 platform-specific default.
 1423 5. Let callerOrigin be origin. If callerOrigin is an opaque origin,
 1424 return a DOMException whose name is "NotAllowedError", and
 1425 terminate this algorithm.
 1426 6. Let effectiveDomain be the callerOrigin's effective domain. If
 1427 effective domain is not a valid domain, then return a DOMException
 1428 whose name is "SecurityError" and terminate this algorithm.
 1429 Note: An effective domain may resolve to a host, which can be
 1430 represented in various manners, such as domain, ipv4 address, ipv6
 1431 address, opaque host, or empty host. Only the domain format of host
 1432 is allowed here.
 1433 7. If options.rpld is not present, then set rpld to effectiveDomain.
 1434 Otherwise:
 1435 1. If options.rpld is not a registrable domain suffix of and is
 1436 not equal to effectiveDomain, return a DOMException whose name
 1437 is "SecurityError", and terminate this algorithm.
 1438 2. Set rpld to options.rpld.
 1439 Note: rpld represents the caller's RP ID. The RP ID defaults
 1440 to being the caller's origin's effective domain unless the
 1441 caller has explicitly set options.rpld when calling get().
 1442 8. Let clientExtensions be a new map and let authenticatorExtensions
 1443 be a new map.
 1444 9. If the extensions member of options is present, then for each
 1445 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:
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

1446 1. If extensionId is not supported by this client platform or is
1447 not an authentication extension, then continue.
1448 2. Set clientExtensions[extensionId] to clientExtensionInput.
1449 3. If extensionId is not an authenticator extension, then
1450 continue.
1451 4. Let authenticatorExtensionInput be the (CBOR) result of
1452 running extensionId's client extension processing algorithm on
1453 clientExtensionInput. If the algorithm returned an error,
1454 continue.
1455 5. Set authenticatorExtensions[extensionId] to the base64url
1456 encoding of authenticatorExtensionInput.
1457 10. Let collectedClientData be a new CollectedClientData instance whose
1458 fields are:
1459
1460 type
1461 The string "webauthn.get".
1462
1463 challenge
1464 The base64url encoding of options.challenge
1465
1466 origin
1467 The serialization of callerOrigin.
1468
1469 tokenBinding
1470 The status of Token Binding between the client and the
1471 callerOrigin, as well as the Token Binding ID associated
1472 with callerOrigin, if one is available.
1473
1474 11. Let clientDataJSON be the JSON-serialized client data constructed
1475 from collectedClientData.
1476 12. Let clientDataHash be the hash of the serialized client data
1477 represented by clientDataJSON.
1478 13. If the options.signal is present and its aborted flag is set to
1479 true, return a DOMException whose name is "AbortError" and
1480 terminate this algorithm.
1481 14. Let issuedRequests be a new ordered set.
1482 15. Let authenticator be a platform-specific handle whose value
1483 identifies an authenticator.
1484 16. Start lifetimeTimer.
1485 17. For each authenticator that becomes available on this platform
1486 during the lifetime of lifetimeTimer, perform the following steps:
1487 The definitions of "lifetime of" and "becomes available" are
1488 intended to represent how devices are hot-plugged into (USB) or
1489 discovered by (NFC) browsers, and are underspecified. Resolving
1490 this with good definitions or some other means will be addressed by
1491 resolving Issue #613.
1492 1. If options.userVerification is set to required and the
1493 authenticator is not capable of performing user verification,
1494 continue.
1495 2. Let userVerification be the effective user verification
1496 requirement for assertion, a Boolean value, as follows. If
1497 options.userVerification
1498
1499 is set to required
1500 Let userVerification be true.
1501
1502 is set to preferred
1503 If the authenticator
1504
1505 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 value, let savedCredentialId be a new
1415 PublicKeyCredentialDescriptor.id and set its value
1416 to allowCredentialDescriptorList[0].id's value (see
1417 here in 6.2.2 The authenticatorGetAssertion
1418 operation for more information).
1419
1420 The foregoing step `_may_` be incorrect, in that we
1421 are attempting to create `savedCredentialId` here and
1422 use it later below, and we do not have a global in
1423 which to allocate a place for it. Perhaps this is
1424 good enough? addendum: @jcjones feels the above step
1425 is likely good enough.
1426

1. For each credential descriptor C in
allowCredentialDescriptorList, append each value, if
any, of C.transports to distinctTransports.
Note: This will aggregate only distinct values of
transports (for this authenticator) in
distinctTransports due to the properties of ordered
sets.
2. If distinctTransports

is not empty
The client selects one transport value
from distinctTransports, possibly
incorporating local configuration
knowledge of the appropriate transport
to use with authenticator in making its
selection.

Then, using transport, invoke the
authenticatorGetAssertion operation on
authenticator, with rpId,
clientDataHash,
allowCredentialDescriptorList,
userPresence, userVerification, and
authenticatorExtensions as parameters.

1506 Let userVerification be true.
1507
1508 is not capable of user verification
1509 Let userVerification be false.
1510
1511 is set to discouraged
1512 Let userVerification be false.
1513
1514 3. Let userPresence be a Boolean value set to the inverse of
1515 userVerification.
1516 4. If options.allowCredentials

is not empty
1. Let allowCredentialDescriptorList be a new list.
2. Execute a platform-specific procedure to determine
which, if any, public key credentials described by
options.allowCredentials are bound to this
authenticator, by matching with rpId,
options.allowCredentials.id, and
options.allowCredentials.type. Set
allowCredentialDescriptorList to this filtered list.
3. If allowCredentialDescriptorList is empty, continue.
4. Let distinctTransports be a new ordered set.
5. If allowCredentialDescriptorList has exactly one

value, let savedCredentialId be a new
PublicKeyCredentialDescriptor.id and set its value
to allowCredentialDescriptorList[0].id's value (see
here in 6.2.3 The authenticatorGetAssertion
operation for more information).

The foregoing step `_may_` be incorrect, in that we
are attempting to create `savedCredentialId` here and
use it later below, and we do not have a global in
which to allocate a place for it. Perhaps this is
good enough? addendum: @jcjones feels the above step
is likely good enough.

1. For each credential descriptor C in
allowCredentialDescriptorList, append each value, if
any, of C.transports to distinctTransports.
Note: This will aggregate only distinct values of
transports (for this authenticator) in
distinctTransports due to the properties of ordered
sets.
2. If distinctTransports

is not empty
The client selects one transport value
from distinctTransports, possibly
incorporating local configuration
knowledge of the appropriate transport
to use with authenticator in making its
selection.

Then, using transport, invoke the
authenticatorGetAssertion operation on
authenticator, with rpId,
clientDataHash,
allowCredentialDescriptorList,
userPresence, userVerification, and
authenticatorExtensions as parameters.

1451 is empty
 1452 Using local configuration knowledge of
 1453 the appropriate transport to use with
 1454 authenticator, invoke the
 1455 authenticatorGetAssertion operation on
 1456 authenticator with rpId, clientDataHash,
 1457 allowCredentialDescriptorList,
 1458 userPresence, userVerification, and
 1459 clientExtensions as parameters.
 1460
 1461 is empty
 1462 Using local configuration knowledge of the
 1463 appropriate transport to use with authenticator,
 1464 invoke the authenticatorGetAssertion operation on
 1465 authenticator with rpId, clientDataHash,
 1466 userPresence, userVerification and clientExtensions
 1467 as parameters.
 1468
 1469 Note: In this case, the Relying Party did not supply
 1470 a list of acceptable credential descriptors. Thus
 1471 the authenticator is being asked to exercise any
 1472 credential it may possess that is bound to the
 1473 Relying Party, as identified by rpId.
 1474
 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.
 1500
 1501 If any authenticator returns an error status,
 1502 Remove authenticator from issuedRequests.
 1503
 1504 If any authenticator indicates success,
 1505
 1506 1. Remove authenticator from issuedRequests.
 1507 2. Let assertionCreationData be a struct whose items are:
 1508
 1509 credentialIdResult
 1510 If savedCredentialId exists, set the value of
 1511 credentialIdResult to be the bytes of
 1512 savedCredentialId. Otherwise, set the value of
 1513 credentialIdResult to be the bytes of the
 1514 credential ID returned from the successful
 1515 authenticatorGetAssertion operation, as
 1516 defined in 6.2.2 The
 1517 authenticatorGetAssertion operation.

1568 is empty
 1569 Using local configuration knowledge of
 1570 the appropriate transport to use with
 1571 authenticator, invoke the
 1572 authenticatorGetAssertion operation on
 1573 authenticator with rpId, clientDataHash,
 1574 allowCredentialDescriptorList,
 1575 userPresence, userVerification, and
 1576 clientExtensions as parameters.
 1577
 1578 is empty
 1579 Using local configuration knowledge of the
 1580 appropriate transport to use with authenticator,
 1581 invoke the authenticatorGetAssertion operation on
 1582 authenticator with rpId, clientDataHash,
 1583 userPresence, userVerification and clientExtensions
 1584 as parameters.
 1585
 1586 Note: In this case, the Relying Party did not supply
 1587 a list of acceptable credential descriptors. Thus,
 1588 the authenticator is being asked to exercise any
 1589 credential it may possess that is bound to the
 1590 Relying Party, as identified by rpId.
 1591
 1592
 1593 5. Append authenticator to issuedRequests.
 1594 18. While lifetimeTimer has not expired, perform the following actions
 1595 depending upon lifetimeTimer and responses from the authenticators:
 1596
 1597 If lifetimeTimer expires,
 1598 For each authenticator in issuedRequests invoke the
 1599 authenticatorCancel operation on authenticator and remove
 1600 authenticator from issuedRequests.
 1601
 1602 If the signal member is present and the aborted flag is set to
 1603 true,
 1604 For each authenticator in issuedRequests invoke the
 1605 authenticatorCancel operation on authenticator and remove
 1606 authenticator from issuedRequests. Then return a
 1607 DOMException whose name is "AbortError" and terminate this
 1608 algorithm.
 1609
 1610 If any authenticator returns a status indicating that the user
 1611 cancelled the operation,
 1612
 1613 1. Remove authenticator from issuedRequests.
 1614 2. For each remaining authenticator in issuedRequests invoke
 1615 the authenticatorCancel operation on authenticator and
 1616 remove it from issuedRequests.
 1617 Note: Authenticators may return an indication of "the
 1618 user cancelled the entire operation". How a user agent
 1619 manifests this state to users is unspecified.
 1620
 1621 If any authenticator returns an error status,
 1622 Remove authenticator from issuedRequests.
 1623
 1624 If any authenticator indicates success,
 1625
 1626 1. Remove authenticator from issuedRequests.
 1627 2. Let assertionCreationData be a struct whose items are:
 1628
 1629 credentialIdResult
 1630 If savedCredentialId exists, set the value of
 1631 credentialIdResult to be the bytes of
 1632 savedCredentialId. Otherwise, set the value of
 1633 credentialIdResult to be the bytes of the
 1634 credential ID returned from the successful
 1635 authenticatorGetAssertion operation, as
 1636 defined in 6.2.3 The
 1637 authenticatorGetAssertion operation.

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
 1540 extension processing algorithm to create the
 1541 client extension outputs, for each client
 1542 extension in clientDataJSON.clientExtensions.
 1543
 1544 3. Let constructAssertionAlg be an algorithm that takes a
 1545 global object global, and whose steps are:
 1546 1. Let pubKeyCred be a new PublicKeyCredential object
 1547 associated with global whose fields are:
 1548
 1549 [[identifier]]
 1550 A new ArrayBuffer, created using
 1551 global's %ArrayBuffer%, containing the
 1552 bytes of
 1553 assertionCreationData.credentialIdResult
 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.
 1566
 1567 authenticatorData
 1568 A new ArrayBuffer, created using
 1569 global's %ArrayBuffer%, containing
 1570 the bytes of
 1571 assertionCreationData.authenticato
 1572 rDataResult.
 1573
 1574 signature
 1575 A new ArrayBuffer, created using
 1576 global's %ArrayBuffer%, containing
 1577 the bytes of
 1578 assertionCreationData.signatureRes
 1579 ult.
 1580
 1581 userHandle
 1582 A new ArrayBuffer, created using

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

1583 global's %ArrayBuffer%, containing
 1584 the bytes of
 1585 assertionCreationData.userHandleRe
 1586 sult.

1587
 1588 [[clientExtensionsResults]]
 1589 A new ArrayBuffer, created using
 1590 global's %ArrayBuffer%, containing the
 1591 bytes of
 1592 assertionCreationData.clientExtensionRes
 1593 ults.

1594
 1595 2. Return pubKeyCred.
 1596 4. For each remaining authenticator in issuedRequests invoke
 1597 the authenticatorCancel operation on authenticator and
 1598 remove it from issuedRequests.
 1599 5. Return constructAssertionAlg and terminate this
 1600 algorithm.

1601
 1602 19. Return a DOMException whose name is "NotAllowedError".

1603
 1604 During the above process, the user agent SHOULD show some UI to the
 1605 user to guide them in the process of selecting and authorizing an
 1606 authenticator with which to complete the operation.

1607
 1608 5.1.5. Store an existing credential - PublicKeyCredential's
 1609 [[Store]](credential, sameOriginWithAncestors) method
 1610

1611 The [[Store]](credential, sameOriginWithAncestors) method is not
 1612 supported for Web Authentication's PublicKeyCredential type, so it
 1613 always returns an error.

1614
 1615 Note: This algorithm is synchronous; the Promise resolution/rejection
 1616 is handled by navigator.credentials.store().

1617
 1618 This internal method accepts two arguments:

1619
 1620 credential
 1621 This argument is a PublicKeyCredential object.

1622
 1623 sameOriginWithAncestors
 1624 This argument is a boolean which is true if and only if the
 1625 caller's environment settings object is same-origin with its
 1626 ancestors.

1627
 1628 When this method is invoked, the user agent MUST execute the following
 1629 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
 1635 PublicKeyCredential's isUserVerifyingPlatformAuthenticatorAvailable() method

1708 null. Otherwise, set this field to
 1709 a new ArrayBuffer, created using
 1710 global's %ArrayBuffer%, containing
 1711 the bytes of
 1712 assertionCreationData.userHandleRe
 1713 sult.

1714
 1715 [[clientExtensionsResults]]
 1716 A new ArrayBuffer, created using
 1717 global's %ArrayBuffer%, containing the
 1718 bytes of
 1719 assertionCreationData.clientExtensionRes
 1720 ults.

1721
 1722 2. Return pubKeyCred.
 1723 4. For each remaining authenticator in issuedRequests invoke
 1724 the authenticatorCancel operation on authenticator and
 1725 remove it from issuedRequests.
 1726 5. Return constructAssertionAlg and terminate this
 1727 algorithm.

1728
 1729 19. Return a DOMException whose name is "NotAllowedError". **In order to**
 1730 **prevent information leak that could identify the user without**
 1731 **consent, this step MUST NOT be executed before lifetimeTimer has**
 1732 **expired. See 14.3 Authentication Ceremony Privacy for details.**

1733
 1734 During the above process, the user agent SHOULD show some UI to the
 1735 user to guide them in the process of selecting and authorizing an
 1736 authenticator with which to complete the operation.

1737
 1738 5.1.5. Store an existing credential - PublicKeyCredential's
 1739 [[Store]](credential, sameOriginWithAncestors) method
 1740

1741 The [[Store]](credential, sameOriginWithAncestors) method is not
 1742 supported for Web Authentication's PublicKeyCredential type, so it
 1743 always returns an error.

1744
 1745 Note: This algorithm is synchronous; the Promise resolution/rejection
 1746 is handled by navigator.credentials.store().

1747
 1748 This internal method accepts two arguments:

1749
 1750 credential
 1751 This argument is a PublicKeyCredential object.

1752
 1753 sameOriginWithAncestors
 1754 This argument is a boolean which is true if and only if the
 1755 caller's environment settings object is same-origin with its
 1756 ancestors.

1757
 1758 When this method is invoked, the user agent MUST execute the following
 1759 algorithm:

1760 1. Return a DOMException whose name is "NotSupportedError", and
 1761 terminate this algorithm

1762
 1763 5.1.6. Preventing silent access to an existing credential -
 1764 PublicKeyCredential's [[preventSilentAccess]](credential,
 1765 sameOriginWithAncestors) method

1766
 1767 Calling the [[preventSilentAccess]](credential,
 1768 sameOriginWithAncestors) method will have no effect on authenticators
 1769 that require an authorization gesture, but setting that flag may
 1770 potentially exclude authenticators that can operate without user
 1771 intervention.

1772
 1773 This internal method accepts no arguments.

1774
 1775 5.1.7. Availability of User-Verifying Platform Authenticator -
 1776 PublicKeyCredential's isUserVerifyingPlatformAuthenticatorAvailable() method
 1777

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
 1652 If this assessment is affirmative, the promise is resolved with the
 1653 value of True. Otherwise, the promise is resolved with the value of
 1654 False. Based on the result, the Relying Party can take further actions
 1655 to guide the user to create a credential.
 1656

1657 This method has no arguments and returns a boolean value.
 1658

1659 If the promise will return False, the client SHOULD wait a fixed period
 1660 of time from the invocation of the method before returning False. This
 1661 is done so that callers can not distinguish between the case where the
 1662 user was unwilling to create a credential using one of the available
 1663 user-verifying platform authenticators and the case where no
 1664 user-verifying platform authenticator exists. Trying to make these
 1665 cases indistinguishable is done in an attempt to not provide additional
 1666 information that could be used for fingerprinting. A timeout value on
 1667 the order of 10 minutes is recommended; this is enough time for
 1668 successful user interactions to be performed but short enough that the
 1669 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().
 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

1778 Relying Parties use this method to determine whether they can create a
 1779 new credential using a user-verifying platform authenticator. Upon
 1780 invocation, the client employs a platform-specific procedure to
 1781 discover available user-verifying platform authenticators. If
 1782 successful, the client then assesses whether the user is willing to
 1783 create a credential using one of the available user-verifying platform
 1784 authenticators. This assessment may include various factors, such as:
 1785 * Whether the user is running in private or incognito mode.
 1786 * Whether the user has configured the client to not create such
 1787 credentials.
 1788 * Whether the user has previously expressed an unwillingness to
 1789 create a new credential for this Relying Party, either through
 1790 configuration or by declining a user interface prompt.
 1791 * The user's explicitly stated intentions, determined through user
 1792 interaction.

1793
 1794 If this assessment is affirmative, the promise is resolved with the
 1795 value of True. Otherwise, the promise is resolved with the value of
 1796 False. Based on the result, the Relying Party can take further actions
 1797 to guide the user to create a credential.
 1798

1799 This method has no arguments and returns a boolean value.
 1800

1801 If the promise will return False, the client SHOULD wait a fixed period
 1802 of time from the invocation of the method before returning False. This
 1803 is done so that callers cannot distinguish between the case where the
 1804 user was unwilling to create a credential using one of the available
 1805 user-verifying platform authenticators and the case where no
 1806 user-verifying platform authenticator exists. Trying to make these
 1807 cases indistinguishable is done in an attempt to not provide additional
 1808 information that could be used for fingerprinting. A timeout value on
 1809 the order of 10 minutes is recommended; this is enough time for
 1810 successful user interactions to be performed but short enough that the
 1811 dangling promise will still be resolved in a reasonably timely fashion.
 1812 partial interface PublicKeyCredential {
 1813 static Promise < boolean > isUserVerifyingPlatformAuthenticatorAvailable();
 1814 };
 1815

1816 **5.2. Authenticator Responses (interface AuthenticatorResponse)**
 1817

1818 Authenticators respond to Relying Party requests by returning an object
 1819 derived from the AuthenticatorResponse interface:
 1820 [SecureContext, Exposed=Window]
 1821 interface AuthenticatorResponse {
 1822 [SameObject] readonly attribute ArrayBuffer clientDataJSON;
 1823 };
 1824

1825 clientDataJSON, of type ArrayBuffer, readonly
 1826 This attribute contains a JSON serialization of the client data
 1827 passed to the authenticator by the client in its call to either
 1828 create() or get().
 1829

1830 **5.2.1. Information about Public Key Credential (interface**
 1831 **AuthenticatorAttestationResponse)**
 1832

1833 The AuthenticatorAttestationResponse interface represents the
 1834 authenticator's response to a client's request for the creation of a
 1835 new public key credential. It contains information about the new
 1836 credential that can be used to identify it for later use, and metadata
 1837 that can be used by the Relying Party to assess the characteristics of
 1838 the credential during registration.
 1839 [SecureContext, Exposed=Window]
 1840 interface AuthenticatorAttestationResponse : AuthenticatorResponse {
 1841 [SameObject] readonly attribute ArrayBuffer attestationObject;
 1842 };
 1843

1844 clientDataJSON
 1845 This attribute, inherited from AuthenticatorResponse, contains
 1846 the JSON-serialized client data (see 6.3 Attestation) passed to
 1847 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
 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 authenticatorGetAssertion
 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.

1848 credential. The exact JSON serialization must be preserved, as
 1849 the hash of the serialized client data has been computed over
 1850 it.
 1851
 1852 attestationObject, of type ArrayBuffer, readonly
 1853 This attribute contains an attestation object, which is opaque
 1854 to, and cryptographically protected against tampering by, the
 1855 client. The attestation object contains both authenticator data
 1856 and an attestation statement. The former contains the AAGUID, a
 1857 unique credential ID, and the credential public key. The
 1858 contents of the attestation statement are determined by the
 1859 attestation statement format used by the authenticator. It also
 1860 contains any additional information that the Relying Party's
 1861 server requires to validate the attestation statement, as well
 1862 as to decode and validate the authenticator data along with the
 1863 JSON-serialized client data. For more details, see 6.3
 1864 Attestation, 6.3.4 Generating an Attestation Object, and Figure
 1865 3.
 1866
 1867 5.2.2. Web Authentication Assertion (interface
 1868 AuthenticatorAssertionResponse)
 1869
 1870 The AuthenticatorAssertionResponse interface represents an
 1871 authenticator's response to a client's request for generation of a new
 1872 authentication assertion given the Relying Party's challenge and
 1873 optional list of credentials it is aware of. This response contains a
 1874 cryptographic signature proving possession of the credential private
 1875 key, and optionally evidence of user consent to a specific transaction.
 1876 [SecureContext, Exposed=Window]
 1877 interface AuthenticatorAssertionResponse : AuthenticatorResponse {
 1878 [SameObject] readonly attribute ArrayBuffer authenticatorData;
 1879 [SameObject] readonly attribute ArrayBuffer signature;
 1880 [SameObject] readonly attribute ArrayBuffer? userHandle;
 1881 };
 1882
 1883 clientDataJSON
 1884 This attribute, inherited from AuthenticatorResponse, contains
 1885 the JSON-serialized client data (see 5.10.1 Client data used in
 1886 WebAuthn signatures (dictionary CollectedClientData)) passed to
 1887 the authenticator by the client in order to generate this
 1888 assertion. The exact JSON serialization **MUST** be preserved, as
 1889 the hash of the serialized client data has been computed over
 1890 it.
 1891
 1892 authenticatorData, of type ArrayBuffer, readonly
 1893 This attribute contains the authenticator data returned by the
 1894 authenticator. See 6.1 Authenticator data.
 1895
 1896 signature, of type ArrayBuffer, readonly
 1897 This attribute contains the raw signature returned from the
 1898 authenticator. See 6.2.3 The authenticatorGetAssertion
 1899 operation.
 1900
 1901 userHandle, of type ArrayBuffer, readonly, **nullable**
 1902 This attribute contains the user handle returned from the
 1903 authenticator, **or null if the authenticator did not return a**
 1904 **user handle. See 6.2.3 The authenticatorGetAssertion operation.**
 1905
 1906 5.3. Parameters for Credential Generation (dictionary
 1907 PublicKeyCredentialParameters)
 1908
 1909 dictionary PublicKeyCredentialParameters {
 1910 required PublicKeyCredentialType type;
 1911 required COSEAlgorithmIdentifier alg;
 1912 };
 1913
 1914 This dictionary is used to supply additional parameters when creating a
 1915 new credential.
 1916
 1917 The type member specifies the type of credential to be created.

1776 The alg member specifies the cryptographic signature algorithm with
 1777 which the newly generated credential will be used, and thus also the
 1778 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,
 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

1918 The alg member specifies the cryptographic signature algorithm with
 1919 which the newly generated credential will be used, and thus also the
 1920 type of asymmetric key pair to be generated, e.g., RSA or Elliptic
 1921 Curve.
 1922
 1923 Note: we use "alg" as the latter member name, rather than spelling-out
 1924 "algorithm", because it will be serialized into a message to the
 1925 authenticator, which may be sent over a low-bandwidth link.
 1926
 1927 5.4. Options for Credential Creation (dictionary
 1928 **PublicKeyCredentialCreationOptions**)
 1929
 1930 dictionary **PublicKeyCredentialCreationOptions** {
 1931 required PublicKeyCredentialRpEntity rp;
 1932 required PublicKeyCredentialUserEntity user;
 1933
 1934 required BufferSource challenge;
 1935 required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
 1936
 1937 unsigned long timeout;
 1938 sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
 1939 AuthenticatorSelectionCriteria authenticatorSelection;
 1940 AttestationConveyancePreference attestation = "none";
 1941 AuthenticationExtensionsClientInputs extensions;
 1942 };
 1943
 1944 rp, of type PublicKeyCredentialRpEntity
 1945 This member contains data about the Relying Party responsible
 1946 for the request.
 1947
 1948 Its value's name member is required.
 1949
 1950 Its value's id member specifies the relying party identifier
 1951 with which the credential should be associated. If omitted, its
 1952 value will be the CredentialsContainer object's relevant
 1953 settings object's origin's effective domain.
 1954
 1955 user, of type PublicKeyCredentialUserEntity
 1956 This member contains data about the user account for which the
 1957 Relying Party is requesting attestation.
 1958
 1959 Its value's name, displayName and id members are required.
 1960
 1961 challenge, of type BufferSource
 1962 This member contains a challenge intended to be used for
 1963 generating the newly created credential's attestation object.
 1964
 1965 pubKeyCredParams, of type sequence<PublicKeyCredentialParameters>
 1966 This member contains information about the desired properties of
 1967 the credential to be created. The sequence is ordered from most
 1968 preferred to least preferred. The platform makes a best-effort
 1969 to create the most preferred credential that it can.
 1970
 1971 timeout, of type unsigned long
 1972 This member specifies a time, in milliseconds, that the caller
 1973 is willing to wait for the call to complete. This is treated as
 1974 a hint, and MAY be overridden by the platform.
 1975
 1976 excludeCredentials, of type sequence<PublicKeyCredentialDescriptor>,
 1977 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 member's
 1889 value. Authenticators MAY truncate a name member's value to a
 1890 length equal to or greater than 64 bytes.
 1891
 1892 icon, of type USVString
 1893 A serialized URL which resolves to an image associated with the
 1894 entity. For example, this could be a user's avatar or a Relying
 1895 Party's logo. This URL MUST be an a priori authenticated URL.
 1896 Authenticators MUST accept and store a 128 byte minimum length
 1897 for a icon member's value. Authenticators MAY ignore a icon
 1898 member's value if its length is greater than 128 bytes.
 1899
 1900 5.4.2. RP Parameters for Credential Generation (dictionary

1979 This member is intended for use by Relying Parties that wish to
 1980 limit the creation of multiple credentials for the same account
 1981 on a single authenticator. The platform is requested to return
 1982 an error if the new credential would be created on an
 1983 authenticator that also contains one of the credentials
 1984 enumerated in this parameter.
 1985
 1986 authenticatorSelection, of type AuthenticatorSelectionCriteria
 1987 This member is intended for use by Relying Parties that wish to
 1988 select the appropriate authenticators to participate in the
 1989 create() operation.
 1990
 1991 attestation, of type AttestationConveyancePreference, defaulting to
 1992 "none"
 1993 This member is intended for use by Relying Parties that wish to
 1994 express their preference for attestation conveyance. The default
 1995 is none.
 1996
 1997 extensions, of type AuthenticationExtensions **ClientInputs**
 1998 This member contains additional parameters requesting additional
 1999 processing by the client and authenticator. For example, the
 2000 caller may request that only authenticators with certain
 2001 capabilities be used to create the credential, or that
 2002 **particular information be returned in the attestation object.**
 2003 **Some extensions are defined in 9 WebAuthn Extensions; consult**
 2004 **the IANA "WebAuthn Extension Identifier" registry established by**
 2005 **[WebAuthn-Registries] for an up-to-date list of registered**
 2006 **WebAuthn Extensions.**
 2007
 2008 5.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)
 2009
 2010 The PublicKeyCredentialEntity dictionary describes a user account, or a
 2011 Relying Party, with which a public key credential is associated.
 2012 dictionary PublicKeyCredentialEntity {
 2013 required DOMString name;
 2014 USVString icon;
 2015 };
 2016
 2017 name, of type DOMString
 2018 A human-readable name for the entity. Its function depends on
 2019 what the PublicKeyCredentialEntity represents:
 2020
 2021 + When inherited by PublicKeyCredentialRpEntity it is a
 2022 human-friendly identifier for the Relying Party, intended only
 2023 for display. For example, "ACME Corporation", "Wonderful
 2024 Widgets, Inc." or "Awesome Site".
 2025 + When inherited by PublicKeyCredentialUserEntity, it is a
 2026 human-palatable identifier for a user account. It is intended
 2027 only for display, and SHOULD allow the user to easily tell the
 2028 difference between user accounts with similar displayNames.
 2029 For example, "alexm", "alex.p.mueller@example.com" or
 2030 "+14255551234". The Relying Party MAY let the user choose
 2031 this, and MAY restrict the choice as needed or appropriate.
 2032 For example, a Relying Party might choose to map
 2033 human-palatable username account identifiers to the name
 2034 member of PublicKeyCredentialUserEntity.
 2035
 2036 Authenticators MUST accept and store a 64-byte minimum length
 2037 for a name member's value. Authenticators MAY truncate a name
 2038 member's value to a length equal to or greater than 64 bytes.
 2039
 2040 icon, of type USVString
 2041 A serialized URL which resolves to an image associated with the
 2042 entity. For example, this could be a user's avatar or a Relying
 2043 Party's logo. This URL MUST be an a priori authenticated URL.
 2044 Authenticators MUST accept and store a 128-byte minimum length
 2045 for an icon member's value. Authenticators MAY ignore an icon
 2046 member's value if its length is greater than 128 bytes.
 2047
 2048 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 member'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
1948   Authenticator Attachment enumeration (enum
1949   AuthenticatorAttachment).
1950
1951 requireResidentKey, of type boolean, defaulting to false
1952   This member describes the Relying Parties' requirements
1953   regarding availability of the Client-side-resident Credential
1954   Private Key. If the parameter is set to true, the authenticator
1955   MUST create a Client-side-resident Credential Private Key when
1956   creating a public key credential.
1957
1958 userVerification, of type UserVerificationRequirement, defaulting to
1959   "preferred"
1960   This member describes the Relying Party's requirements regarding
1961   user verification for the create() operation. Eligible
1962   authenticators are filtered to only those capable of satisfying
1963   this requirement.
1964
1965 5.4.5. Authenticator Attachment enumeration (enum AuthenticatorAttachment)
1966 enum AuthenticatorAttachment {

```

```

2049 PublicKeyCredentialRpEntity)
2050
2051 The PublicKeyCredentialRpEntity dictionary is used to supply additional
2052 Relying Party attributes when creating a new credential.
2053 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
2054   DOMString id;
2055 };
2056
2057 id, of type DOMString
2058   A unique identifier for the Relying Party entity, which sets the
2059   RP ID.
2060
2061 5.4.3. User Account Parameters for Credential Generation (dictionary
2062 PublicKeyCredentialUserEntity)
2063
2064 The PublicKeyCredentialUserEntity dictionary is used to supply
2065 additional user account attributes when creating a new credential.
2066 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
2067   required BufferSource id;
2068   required DOMString displayName;
2069 };
2070
2071 id, of type BufferSource
2072   The user handle of the user account entity.
2073
2074 displayName, of type DOMString
2075   A human-friendly name for the user account, intended only for
2076   display. For example, "Alex P. Miller" or ". The Relying
2077   Party SHOULD let the user choose this, and SHOULD NOT restrict
2078   the choice more than necessary.
2079
2080   Authenticators MUST accept and store a 64-byte minimum length
2081   for a displayName member's value. Authenticators MAY truncate a
2082   displayName member's value to a length equal to or greater than
2083   64 bytes.
2084
2085 5.4.4. Authenticator Selection Criteria (dictionary
2086 AuthenticatorSelectionCriteria)
2087
2088 Relying Parties may use the AuthenticatorSelectionCriteria dictionary
2089 to specify their requirements regarding authenticator attributes.
2090 dictionary AuthenticatorSelectionCriteria {
2091   AuthenticatorAttachment authenticatorAttachment;
2092   boolean requireResidentKey = false;
2093   UserVerificationRequirement userVerification = "preferred";
2094 };
2095
2096 authenticatorAttachment, of type AuthenticatorAttachment
2097   If this member is present, eligible authenticators are filtered
2098   to only authenticators attached with the specified 5.4.5
2099   Authenticator Attachment enumeration (enum
2100   AuthenticatorAttachment).
2101
2102 requireResidentKey, of type boolean, defaulting to false
2103   This member describes the Relying Parties' requirements
2104   regarding availability of the Client-side-resident Credential
2105   Private Key. If the parameter is set to true, the authenticator
2106   MUST create a Client-side-resident Credential Private Key when
2107   creating a public key credential.
2108
2109 userVerification, of type UserVerificationRequirement, defaulting to
2110   "preferred"
2111   This member describes the Relying Party's requirements regarding
2112   user verification for the create() operation. Eligible
2113   authenticators are filtered to only those capable of satisfying
2114   this requirement.
2115
2116 5.4.5. Authenticator Attachment enumeration (enum AuthenticatorAttachment)
2117 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
2002 originally registered with the Relying Party using a different client.
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022

```

```

2119 "platform", // Platform attachment
2120 "cross-platform" // Cross-platform attachment
2121 };
2122
2123 Clients can communicate with authenticators using a variety of
2124 mechanisms. For example, a client MAY use a platform-specific API to
2125 communicate with an authenticator which is physically bound to a
2126 platform. On the other hand, a client can use a variety of standardized
2127 cross-platform transport protocols such as Bluetooth (see 5.10.4
2128 Authenticator Transport enumeration (enum AuthenticatorTransport)) to
2129 discover and communicate with cross-platform attached authenticators.
2130 Therefore, we use AuthenticatorAttachment to describe an
2131 authenticator's attachment modality. We define authenticators that are
2132 part of the client's platform as having a platform attachment, and
2133 refer to them as platform authenticators. While those that are
2134 reachable via cross-platform transport protocols are defined as having
2135 cross-platform attachment, and refer to them as roaming authenticators.
2136 * platform attachment - the respective authenticator is attached
2137 using platform-specific transports. Usually, authenticators of this
2138 class are non-removable from the platform. A public key credential
2139 bound to a platform authenticator is called a platform credential.
2140 * cross-platform attachment - the respective authenticator is
2141 attached using cross-platform transports. Authenticators of this
2142 class are removable from, and can "roam" among, client platforms. A
2143 public key credential bound to a roaming authenticator is called a
2144 roaming credential.
2145
2146 This distinction is important because there are use-cases where only
2147 platform authenticators are acceptable to a Relying Party, and
2148 conversely ones where only roaming authenticators are employed. As a
2149 concrete example of the former, a platform credential may be used by
2150 Relying Parties to quickly and conveniently reauthenticate the user
2151 with a minimum of friction, e.g., the user will not have to dig around
2152 in their pocket for their key fob or phone. As a concrete example of
2153 the latter, when the user is accessing the Relying Party from a given
2154 client for the first time, they may be asked to use a roaming
2155 credential which was originally registered with the Relying Party using
2156 a different client.
2157
2158 Note: An attachment modality selection option is available only in the
2159 [[Create]](origin, options, sameOriginWithAncestors) operation. The
2160 Relying Party may use it to, for example, ensure the user has a roaming
2161 credential for authenticating using other clients; or to specifically
2162 register a platform credential for easier reauthentication using a
2163 particular client. The [[DiscoverFromExternalSource]](origin, options,
2164 sameOriginWithAncestors) operation has no attachment modality selection
2165 option, so the Relying Party should accept any of the user's registered
2166 credentials. The client and user will then use whichever is available
2167 and convenient at the time.
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186

```

2022 * indirect - indicates that the Relying Party prefers an attestation
 2023 conveyance yielding verifiable attestation statements, but allows
 2024 the client to decide how to obtain such attestation statements. The
 2025 client **may** replace the authenticator-generated attestation
 2026 statements with attestation statements generated by a **Privacy CA**,
 2027 in order to protect the user's privacy, or to **assist Relying**
 2028 Parties with attestation verification in a **heterogeneous ecosystem**.
 2029

2030 Note: There is no guarantee that the Relying Party will obtain a
 2031 verifiable attestation statement in this case. For example, in the
 2032 case that the authenticator employs self attestation.
 2033 * direct - indicates that the Relying Party wants to receive the
 2034 attestation statement as generated by the authenticator.
 2035

2036 5.5. Options for Assertion Generation (dictionary
 2037 PublicKeyCredentialRequestOptions)
 2038

2039 The PublicKeyCredentialRequestOptions dictionary supplies get() with
 2040 the data it needs to generate an assertion. Its challenge member **must**
 2041 be present, while its other members are **optional**.
 2042 dictionary PublicKeyCredentialRequestOptions {
 2043 required BufferSource challenge;
 2044 unsigned long timeout;
 2045 USVString rpld;
 2046 sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
 2047 UserVerificationRequirement userVerification = "preferred";
 2048 AuthenticationExtensions extensions;
 2049 };

2050 challenge, of type BufferSource
 2051 This member represents a challenge that the selected
 2052 authenticator signs, along with other data, when producing an
 2053 authentication assertion. See the 13.1 Cryptographic Challenges
 2054 security consideration.
 2055

2056 timeout, of type unsigned long
 2057 This **optional** member specifies a time, in milliseconds, that the
 2058 caller is willing to wait for the call to complete. The value is
 2059 treated as a hint, and **may** be overridden by the platform.
 2060

2061 rpld, of type USVString
 2062 This optional member specifies the relying party identifier
 2063 claimed by the caller. If omitted, its value will be the
 2064 CredentialsContainer object's relevant settings object's
 2065 origin's effective domain.
 2066

2067 allowCredentials, of type sequence<PublicKeyCredentialDescriptor>,
 2068 defaulting to None
 2069 This optional member contains a list of
 2070 PublicKeyCredentialDescriptor objects representing public key
 2071 credentials acceptable to the caller, in descending order of the
 2072 caller's preference (the first item in the list is the most
 2073 preferred credential, and so on down the list).
 2074

2075 userVerification, of type UserVerificationRequirement, defaulting to
 2076 "preferred"
 2077 This member describes the Relying Party's requirements regarding
 2078 user verification for the get() operation. Eligible
 2079 authenticators are filtered to only those capable of satisfying
 2080 this requirement.
 2081

2082 extensions, of type AuthenticationExtensions
 2083 This **optional** member contains additional parameters requesting
 2084 additional processing by the client and authenticator. For
 2085 example, if transaction confirmation is sought from the user,
 2086 then the prompt string might be included as an extension.
 2087

2088 5.6. Abort operations with AbortSignal
 2089
 2090 Developers are encouraged to leverage the AbortController to manage the
 2091

2187 * indirect - indicates that the Relying Party prefers an attestation
 2188 conveyance yielding verifiable attestation statements, but allows
 2189 the client to decide how to obtain such attestation statements. The
 2190 client **MAY** replace the authenticator-generated attestation
 2191 statements with attestation statements generated by an
 2192 **Anonymization CA**, in order to protect the user's privacy, or to
 2193 **assist Relying** Parties with attestation verification in a
 2194 **heterogeneous ecosystem**.
 2195 Note: There is no guarantee that the Relying Party will obtain a
 2196 verifiable attestation statement in this case. For example, in the
 2197 case that the authenticator employs self attestation.
 2198 * direct - indicates that the Relying Party wants to receive the
 2199 attestation statement as generated by the authenticator.
 2200

2201 5.5. Options for Assertion Generation (dictionary
 2202 PublicKeyCredentialRequestOptions)
 2203

2204 The PublicKeyCredentialRequestOptions dictionary supplies get() with
 2205 the data it needs to generate an assertion. Its challenge member **MUST**
 2206 be present, while its other members are **OPTIONAL**.
 2207 dictionary PublicKeyCredentialRequestOptions {
 2208 required BufferSource challenge;
 2209 unsigned long timeout;
 2210 USVString rpld;
 2211 sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
 2212 UserVerificationRequirement userVerification = "preferred";
 2213 AuthenticationExtensions **ClientInputs** extensions;
 2214 };

2215 challenge, of type BufferSource
 2216 This member represents a challenge that the selected
 2217 authenticator signs, along with other data, when producing an
 2218 authentication assertion. See the 13.1 Cryptographic Challenges
 2219 security consideration.
 2220

2221 timeout, of type unsigned long
 2222 This **OPTIONAL** member specifies a time, in milliseconds, that the
 2223 caller is willing to wait for the call to complete. The value is
 2224 treated as a hint, and **MAY** be overridden by the platform.
 2225

2226 rpld, of type USVString
 2227 This optional member specifies the relying party identifier
 2228 claimed by the caller. If omitted, its value will be the
 2229 CredentialsContainer object's relevant settings object's
 2230 origin's effective domain.
 2231

2232 allowCredentials, of type sequence<PublicKeyCredentialDescriptor>,
 2233 defaulting to None
 2234 This optional member contains a list of
 2235 PublicKeyCredentialDescriptor objects representing public key
 2236 credentials acceptable to the caller, in descending order of the
 2237 caller's preference (the first item in the list is the most
 2238 preferred credential, and so on down the list).
 2239

2240 userVerification, of type UserVerificationRequirement, defaulting to
 2241 "preferred"
 2242 This member describes the Relying Party's requirements regarding
 2243 user verification for the get() operation. Eligible
 2244 authenticators are filtered to only those capable of satisfying
 2245 this requirement.
 2246

2247 extensions, of type AuthenticationExtensions **ClientInputs**
 2248 This **OPTIONAL** member contains additional parameters requesting
 2249 additional processing by the client and authenticator. For
 2250 example, if transaction confirmation is sought from the user,
 2251 then the prompt string might be included as an extension.
 2252

2253 5.6. Abort operations with AbortSignal
 2254
 2255 Developers are encouraged to leverage the AbortController to manage the
 2256

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 typedef record<DOMString, any> AuthenticationExtensions;
```

2132 This is a dictionary containing zero or more WebAuthn extensions, as
 2133 defined in 9 WebAuthn Extensions. An AuthenticationExtensions instance
 2134 can contain either client extensions or authenticator extensions,
 2135 depending upon context.
 2136

2137 5.8. Supporting Data Structures

2139 The public key credential type uses certain data structures that are
 2140

2257 [[Create]](origin, options, sameOriginWithAncestors) and
 2258 [[DiscoverFromExternalSource]](origin, options,
 2259 sameOriginWithAncestors) operations. See DOM 3.3 Using AbortController
 2260 and AbortSignal objects in APIs section for detailed instructions.
 2261

2262 Note: DOM 3.3 Using AbortController and AbortSignal objects in APIs
 2263 section specifies that web platform APIs integrating with the
 2264 AbortController must reject the promise immediately once the aborted
 2265 flag is set. Given the complex inheritance and parallelization
 2266 structure of the [[Create]](origin, options, sameOriginWithAncestors)
 2267 and [[DiscoverFromExternalSource]](origin, options,
 2268 sameOriginWithAncestors) methods, the algorithms for the two APIs
 2269 fulfills this requirement by checking the aborted flag in three places.
 2270 In the case of [[Create]](origin, options, sameOriginWithAncestors),
 2271 the aborted flag is checked first in Credential Management 1 2.5.4
 2272 Create a Credential immediately before calling [[Create]](origin,
 2273 options, sameOriginWithAncestors), then in 5.1.3 Create a new
 2274 credential - PublicKeyCredential's [[Create]](origin, options,
 2275 sameOriginWithAncestors) method right before authenticator sessions
 2276 start, and finally during authenticator sessions. The same goes for
 2277 [[DiscoverFromExternalSource]](origin, options,
 2278 sameOriginWithAncestors).
 2279

2280 The visibility and focus state of the Window object determines whether
 2281 the [[Create]](origin, options, sameOriginWithAncestors) and
 2282 [[DiscoverFromExternalSource]](origin, options,
 2283 sameOriginWithAncestors) operations should continue. When the Window
 2284 object associated with the [Document loses focus, [[Create]](origin,
 2285 options, sameOriginWithAncestors) and
 2286 [[DiscoverFromExternalSource]](origin, options,
 2287 sameOriginWithAncestors) operations SHOULD be aborted.
 2288

2289 The WHATWG HTML WG is discussing whether to provide a hook when a
 2290 browsing context gains or loses focus. If a hook is provided, the
 2291 above paragraph will be updated to include the hook. See WHATWG HTML WG
 2292 Issue #2711 for more details.
 2293

2294 5.7. Authentication Extensions Client Inputs (typedef 2295 AuthenticationExtensionsClientInputs)

```
2296 dictionary AuthenticationExtensionsClientInputs {  

  2297 };  

  2298
```

2299 This is a dictionary containing the client extension input values for
 2300 zero or more WebAuthn extensions, as defined in 9 WebAuthn Extensions.
 2301

2302 5.8. Authentication Extensions Client Outputs (typedef 2303 AuthenticationExtensionsClientOutputs)

```
2304 dictionary AuthenticationExtensionsClientOutputs {  

  2305 };  

  2306
```

2307 This is a dictionary containing the client extension output values for
 2308 zero or more WebAuthn extensions, as defined in 9 WebAuthn Extensions.
 2309

2310 5.9. Authentication Extensions Authenticator Inputs (typedef 2311 AuthenticationExtensionsAuthenticatorInputs)

```
2312 typedef record<DOMString, DOMString> AuthenticationExtensionsAuthenticatorInputs  

  2313 ;  

  2314
```

2315 This is a dictionary containing the authenticator extension input
 2316 values for zero or more WebAuthn extensions, as defined in 9 WebAuthn
 2317 Extensions.
 2318

2319 5.10. Supporting Data Structures

2320 The public key credential type uses certain data structures that are
 2321
 2322
 2323
 2324

2141 specified in supporting specifications. These are as follows.
 2142
 2143 **5.8.1. Client data used in WebAuthn signatures (dictionary**
 2144 **CollectedClientData)**
 2145
 2146 The client data represents the contextual bindings of both the Relying
 2147 Party and the client platform. It is a key-value mapping with
 2148 string-valued keys. Values **may** be any type that has a valid encoding in
 2149 JSON. Its structure is defined by the following Web IDL.
 2150 dictionary CollectedClientData {
 2151 required DOMString type;
 2152 required DOMString challenge;
 2153 required DOMString origin;
 2154 required DOMString hashAlgorithm;
 2155 DOMString tokenBindingId;
 2156 AuthenticationExtensions clientExtensions;
 2157 AuthenticationExtensions authenticatorExtensions;
 2158 };
 2159

2160 The type member contains the string "webauthn.create" when creating new
 2161 credentials, and "webauthn.get" when getting an assertion from an
 2162 existing credential. The purpose of this member is to prevent certain
 2163 types of signature confusion attacks (where an attacker substitutes one
 2164 legitimate signature for another).
 2165

2166 The challenge member contains the base64url encoding of the challenge
 2167 provided by the RP. See the 13.1 Cryptographic Challenges security
 2168 consideration.
 2169

2170 The origin member contains the fully qualified origin of the requester,
 2171 as provided to the authenticator by the client, in the syntax defined
 2172 by [RFC6454].
 2173

2174 The **hashAlgorithm** member is a recognized algorithm name that supports
 2175 the "digest" operation, which specifies the algorithm used to compute
 2176 the hash of the serialized client data. This algorithm is chosen by the
 2177 client at its sole discretion.
 2178

2179 The **tokenBindingId** member contains the base64url encoding of the Token
 2180 Binding ID that this client uses for the Token Binding protocol when
 2181 communicating with the Relying Party. This can be omitted if no Token
 2182 Binding has been negotiated between the client and the Relying Party.
 2183

2184 The optional **clientExtensions** and **authenticatorExtensions** members
 2185 contain additional parameters generated by processing the extensions
 2186 passed in by the Relying Party. WebAuthn extensions are detailed in
 2187 Section 9 WebAuthn Extensions.
 2188

2189 This structure is used by the client to compute the following
 2190 quantities:

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

2200 **5.8.2. Credential Type enumeration (enum PublicKeyCredentialType)**

2201 enum PublicKeyCredentialType {
 2202 "public-key"
 2203

2325 specified in supporting specifications. These are as follows.

2326
 2327 **5.10.1. Client data used in WebAuthn signatures (dictionary**
 2328 **CollectedClientData)**

2329 The client data represents the contextual bindings of both the Relying
 2330 Party and the client platform. It is a key-value mapping with
 2331 string-valued keys. Values **can** be any type that has a valid encoding in
 2332 JSON. Its structure is defined by the following Web IDL.
 2333 dictionary CollectedClientData {

2334 required DOMString type;
 2335 required DOMString challenge;
 2336 required DOMString origin;
 2337 **TokenBinding tokenBinding;**
 2338

2339 };
 2340
 2341 dictionary TokenBinding {
 2342 required TokenBindingStatus status;
 2343 DOMString id;
 2344 };
 2345

2346 **enum TokenBindingStatus { "present", "supported", "not-supported" };**

2347 The type member contains the string "webauthn.create" when creating new
 2348 credentials, and "webauthn.get" when getting an assertion from an
 2349 existing credential. The purpose of this member is to prevent certain
 2350 types of signature confusion attacks (where an attacker substitutes one
 2351 legitimate signature for another).
 2352

2353 The challenge member contains the base64url encoding of the challenge
 2354 provided by the RP. See the 13.1 Cryptographic Challenges security
 2355 consideration.
 2356

2357 The origin member contains the fully qualified origin of the requester,
 2358 as provided to the authenticator by the client, in the syntax defined
 2359 by [RFC6454].
 2360

2361 The **tokenBinding** member contains information about the state of the
 2362 Token Binding protocol used when communicating with the Relying Party.
 2363 The status member is one of:
 2364 * **not-supported**: when the client does not support token binding.
 2365 * **supported**: the client supports token binding, but it was not
 2366 negotiated when communicating with the Relying Party.
 2367 * **present**: token binding was used when communicating with the Relying
 2368 Party. In this case, the id member **MUST** be present and **MUST** be a
 2369 base64url encoding of the Token Binding ID that was used.
 2370

2371 This structure is used by the client to compute the following
 2372 quantities:

2373 JSON-serialized client data
 2374 This is the UTF-8 encoding of the result of calling the initial
 2375 value of JSON.stringify on a CollectedClientData dictionary.
 2376

2377 Hash of the serialized client data
 2378 This is the hash (computed using **SHA-256**) of the **JSON-serialized**
 2379 client data, as constructed by the client.
 2380

2381 **5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)**

2382 enum PublicKeyCredentialType {
 2383 "public-key"
 2384

```

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;
2258
2259 A COSEAlgorithmIdentifier's value is a number identifying a
2260 cryptographic algorithm. The algorithm identifiers SHOULD be values
2261 registered in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG],
2262 for instance, -7 for "ES256" and -257 for "RS256".
2263
2264 5.8.6. User Verification Requirement enumeration (enum
2265 UserVerificationRequirement)
2266
2267 enum UserVerificationRequirement {
2268   "required",
2269   "preferred",
2270   "discouraged"
2271 };
2272
2273 A Relying Party may require user verification for some of its

```

```

2387 };
2388
2389 This enumeration defines the valid credential types. It is an extension
2390 point; values can be added to it in the future, as more credential
2391 types are defined. The values of this enumeration are used for
2392 versioning the Authentication Assertion and attestation structures
2393 according to the type of the authenticator.
2394
2395 Currently one credential type is defined, namely "public-key".
2396
2397 5.10.3. Credential Descriptor (dictionary PublicKeyCredentialDescriptor)
2398
2399 dictionary PublicKeyCredentialDescriptor {
2400   required PublicKeyCredentialType type;
2401   required BufferSource id;
2402   sequence<AuthenticatorTransport> transports;
2403 };
2404
2405 This dictionary contains the attributes that are specified by a caller
2406 when referring to a public key credential as an input parameter to the
2407 create() or get() methods. It mirrors the fields of the
2408 PublicKeyCredential object returned by the latter methods.
2409
2410 The type member contains the type of the public key credential the
2411 caller is referring to.
2412
2413 The id member contains the credential ID of the public key credential
2414 that the caller is referring to.
2415
2416 5.10.4. Authenticator Transport enumeration (enum AuthenticatorTransport)
2417
2418 enum AuthenticatorTransport {
2419   "usb",
2420   "nfc",
2421   "ble"
2422 };
2423
2424 Authenticators may communicate with clients using a variety of
2425 transports. This enumeration defines a hint as to how clients might
2426 communicate with a particular authenticator in order to obtain an
2427 assertion for a specific credential. Note that these hints represent
2428 the Relying Party's best belief as to how an authenticator may be
2429 reached. A Relying Party may obtain a list of transports hints from
2430 some attestation statement formats or via some out-of-band mechanism;
2431 it is outside the scope of this specification to define that mechanism.
2432 * usb - the respective authenticator can be contacted over USB.
2433 * nfc - the respective authenticator can be contacted over Near Field
2434 Communication (NFC).
2435 * ble - the respective authenticator can be contacted over Bluetooth
2436 Smart (Bluetooth Low Energy / BLE).
2437
2438 5.10.5. Cryptographic Algorithm Identifier (typedef COSEAlgorithmIdentifier)
2439
2440 typedef long COSEAlgorithmIdentifier;
2441
2442 A COSEAlgorithmIdentifier's value is a number identifying a
2443 cryptographic algorithm. The algorithm identifiers SHOULD be values
2444 registered in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG],
2445 for instance, -7 for "ES256" and -257 for "RS256".
2446
2447 5.10.6. User Verification Requirement enumeration (enum
2448 UserVerificationRequirement)
2449
2450 enum UserVerificationRequirement {
2451   "required",
2452   "preferred",
2453   "discouraged"
2454 };
2455
2456 A Relying Party may require user verification for some of its

```

2274 operations but not for others, and may use this type to express its
2275 needs.
2276

2277 The value required indicates that the Relying Party requires user
2278 verification for the operation and will fail the operation if the
2279 response does not have the UV flag set.
2280

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

2295 Client platforms **may** implement and expose this abstract model in any
2296 way desired. However, the behavior of the client's Web Authentication
2297 API implementation, when operating on the authenticators supported by
2298 that platform, **MUST** be indistinguishable from the behavior specified in
2299 5 Web Authentication API.
2300

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

2457 operations but not for others, and may use this type to express its
2458 needs.
2459

2460 The value required indicates that the Relying Party requires user
2461 verification for the operation and will fail the operation if the
2462 response does not have the UV flag set.
2463

2464 The value preferred indicates that the Relying Party prefers user
2465 verification for the operation if possible, but will not fail the
2466 operation if the response does not have the UV flag set.
2467

2468 The value discouraged indicates that the Relying Party does not want
2469 user verification employed during the operation (e.g., in the interest
2470 of minimizing disruption to the user interaction flow).
2471

2472 6. WebAuthn Authenticator Model

2473 The **Web Authentication API implies a specific abstract functional model
2474 for an authenticator**. This section describes that authenticator model.
2475

2476 Client platforms **MAY** implement and expose this abstract model in any
2477 way desired. However, the behavior of the client's Web Authentication
2478 API implementation, when operating on the authenticators supported by
2479 that platform, **MUST** be indistinguishable from the behavior specified in
2480 5 Web Authentication API.
2481

2482 For authenticators, this model defines the logical operations that they
2483 **MUST** support, and the data formats that they expose to the client and
2484 the Relying Party. However, it does not define the details of how
2485 authenticators communicate with the client platform, unless they are
2486 **necessary** for interoperability with Relying Parties. For instance, this
2487 abstract model does not define protocols for connecting authenticators
2488 to clients over transports such as USB or NFC. Similarly, this abstract
2489 model does not define specific error codes or methods of returning
2490 them; however, it does define error behavior in terms of the needs of
2491 the client. Therefore, specific error codes are mentioned as a means of
2492 showing which error conditions must be distinguishable (or not) from
2493 each other in order to enable a compliant and secure client
2494 implementation.
2495

2496 **Relying Parties may influence authenticator selection, if they deem
2497 necessary, by stipulating various authenticator characteristics when
2498 creating credentials and/or when generating assertions, through use of
2499 credential creation options or assertion generation options,
2500 respectively. The algorithms underlying the WebAuthn API marshal these
2501 options and pass them to the applicable authenticator operations
2502 defined below.**
2503

2504 In this abstract model, the authenticator provides key management and
2505 cryptographic signatures. It **can** be embedded in the WebAuthn client or
2506 housed in a separate device entirely. The authenticator itself **can**
2507 contain a cryptographic module which operates at a higher security
2508 level than the rest of the authenticator. This is particularly
2509 important for authenticators that are embedded in the WebAuthn client,
2510 as in those cases this cryptographic module (which may, for example, be
2511 a TPM) could be considered more trustworthy than the rest of the
2512 authenticator.
2513

2514 Each authenticator stores **a credentials map, a map from (rpId,
2515 userHandle) to public key credential source**.
2516

2517 **Additionally, each authenticator has an AAGUID, which is a 128-bit
2518 identifier indicating** the type (e.g. make and model) of the
2519 **authenticator**. The AAGUID **MUST** be chosen by the manufacturer to be
2520 **identical across all substantially identical authenticators made by
2521 that manufacturer, and different (with probability 1-2^-128 or greater)**

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:
2369 1. An attestation signature is produced when a new public key
2370 credential is created via an authenticatorMakeCredential operation.
2371 An attestation signature provides cryptographic proof of certain
2372 properties of the authenticator and the credential. For
2373 instance, an attestation signature asserts the authenticator type
2374 (as denoted by its AAGUID) and the credential public key. The
2375 attestation signature is signed by an attestation private key,
2376 which is chosen depending on the type of attestation desired. For
2377 more details on attestation, see 6.3 Attestation.
2378 2. An assertion signature is produced when the
2379 authenticatorGetAssertion method is invoked. It represents an
2380 assertion by the authenticator that the user has consented to a
2381 specific transaction, such as logging in, or completing a purchase.
2382 Thus, an assertion signature asserts that the authenticator
2383 possessing a particular credential private key has established, to
2384 the best of its ability, that the user requesting this transaction
2385 is the same user who consented to creating that particular public
2386 key credential. It also asserts additional information, termed
2387 client data, that may be useful to the caller, such as the means by
2388 which user consent was provided, and the prompt shown to the user
2389 by the authenticator. The assertion signature format is illustrated
2390 in Figure 2, below.
2391

2392 The formats of these signatures, as well as the procedures for
2393 generating them, are specified below.

2394 6.1. Authenticator data

2395 The authenticator data structure encodes contextual bindings made by
2396 the authenticator. These bindings are controlled by the authenticator
2397 itself, and derive their trust from the Relying Party's assessment of
2400 the security properties of the authenticator. In one extreme case, the
2401 authenticator may be embedded in the client, and its bindings may be no
2402 more trustworthy than the client data. At the other extreme, the
2403 authenticator may be a discrete entity with high-security hardware and
2404

2523 from the AAGUIDs of all other types of authenticators. The RP MAY use
2524 the AAGUID to infer certain properties of the authenticator, such as
2525 certification level and strength of key protection, using information
2526 from other sources.

2527
2528 The primary function of the authenticator is to provide WebAuthn
2529 signatures, which are bound to various contextual data. These data are
2530 observed and added at different levels of the stack as a signature
2531 request passes from the server to the authenticator. In verifying a
2532 signature, the server checks these bindings against expected values.
2533 These contextual bindings are divided in two: Those added by the RP or
2534 the client, referred to as client data; and those added by the
2535 authenticator, referred to as the authenticator data. The authenticator
2536 signs over the client data, but is otherwise not interested in its
2537 contents. To save bandwidth and processing requirements on the
2538 authenticator, the client hashes the client data and sends only the
2539 result to the authenticator. The authenticator signs over the
2540 combination of the hash of the serialized client data, and its own
2541 authenticator data.
2542

- 2543 The goals of this design can be summarized as follows.
- 2544 * The scheme for generating signatures should accommodate cases where
 - 2545 the link between the client platform and authenticator is very
 - 2546 limited, in bandwidth and/or latency. Examples include Bluetooth
 - 2547 Low Energy and Near-Field Communication.
 - 2548 * The data processed by the authenticator should be small and easy to
 - 2549 interpret in low-level code. In particular, authenticators should
 - 2550 not have to parse high-level encodings such as JSON.
 - 2551 * Both the client platform and the authenticator should have the
 - 2552 flexibility to add contextual bindings as needed.
 - 2553 * The design aims to reuse as much as possible of existing encoding
 - 2554 formats in order to aid adoption and implementation.

2555 Authenticators produce cryptographic signatures for two distinct
2556 purposes:

- 2557 1. An attestation signature is produced when a new public key
2558 credential is created via an authenticatorMakeCredential operation.
2559 An attestation signature provides cryptographic proof of certain
2560 properties of the authenticator and the credential. For instance,
2561 an attestation signature asserts the authenticator type (as denoted
2562 by its AAGUID) and the credential public key. The attestation
2563 signature is signed by an attestation private key, which is chosen
2564 depending on the type of attestation desired. For more details on
2565 attestation, see 6.3 Attestation.
- 2566 2. An assertion signature is produced when the
2567 authenticatorGetAssertion method is invoked. It represents an
2568 assertion by the authenticator that the user has consented to a
2569 specific transaction, such as logging in, or completing a purchase.
2570 Thus, an assertion signature asserts that the authenticator
2571 possessing a particular credential private key has established, to
2572 the best of its ability, that the user requesting this transaction
2573 is the same user who consented to creating that particular public
2574 key credential. It also asserts additional information, termed
2575 client data, that may be useful to the caller, such as the means by
2576 which user consent was provided, and the prompt shown to the user
2577 by the authenticator. The assertion signature format is illustrated
2578 in Figure 2, below.

2579 The formats of these signatures, as well as the procedures for
2580 generating them, are specified below.

2581 6.1. Authenticator data

2582 The authenticator data structure encodes contextual bindings made by
2583 the authenticator. These bindings are controlled by the authenticator
2584 itself, and derive their trust from the Relying Party's assessment of
2585 the security properties of the authenticator. In one extreme case, the
2586 authenticator may be embedded in the client, and its bindings may be no
2587 more trustworthy than the client data. At the other extreme, the
2588 authenticator may be a discrete entity with high-security hardware and
2589

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.
 2459
 2460 The UP flag SHALL be set if and only if the authenticator detected a
 2461 user through an authenticator specific gesture. The RFU bits SHALL be
 2462 set to zero.
 2463
 2464 For attestation signatures, the authenticator MUST set the AT flag and
 2465 include the attestedCredentialData. For authentication signatures, the
 2466 AT flag MUST NOT be set and the attestedCredentialData MUST NOT be
 2467 included.
 2468
 2469 If the authenticator does not include any extension data, it MUST set
 2470 the ED flag to zero, and to one if extension data is included.
 2471
 2472 The figure below shows a visual representation of the authenticator
 2473 data structure.
 2474 [Authenticator data layout](#) Authenticator data layout.

2593 software, connected to the client over a secure channel. In both cases,
 2594 the Relying Party receives the authenticator data in the same format,
 2595 and uses its knowledge of the authenticator to make trust decisions.
 2596
 2597 The authenticator data has a compact but extensible encoding. This is
 2598 desired since authenticators can be devices with limited capabilities
 2599 and low power requirements, with much simpler software stacks than the
 2600 client platform components.
 2601
 2602 The authenticator data structure is a byte array of 37 bytes or more,
 2603 as follows.
 2604
 2605 Name Length (in bytes) Description
 2606 rpIdHash 32 SHA-256 hash of the RP ID associated with the credential.
 2607 flags 1 Flags (bit 0 is the least significant bit):
 2608 * Bit 0: User Present (UP) result.
 2609 + 1 means the user is present.
 2610 + 0 means the user is not present.
 2611 * Bit 1: Reserved for future use (RFU1).
 2612 * Bit 2: User Verified (UV) result.
 2613 + 1 means the user is verified.
 2614 + 0 means the user is not verified.
 2615 * Bits 3-5: Reserved for future use (RFU2).
 2616 * Bit 6: Attested credential data included (AT).
 2617 + Indicates whether the authenticator added attested credential
 2618 data.
 2619 * Bit 7: Extension data included (ED).
 2620 + Indicates if the authenticator data has extensions.
 2621
 2622 signCount 4 Signature counter, 32-bit unsigned big-endian integer.
 2623 attestedCredentialData variable (if present) attested credential data
 2624 (if present). See 6.3.1 Attested credential data for details. Its
 2625 length depends on the length of the credential ID and credential public
 2626 key being attested.
 2627 extensions variable (if present) Extension-defined authenticator data.
 2628 This is a CBOR [RFC7049] map with extension identifiers as keys, and
 2629 authenticator extension outputs as values. See 9 WebAuthn Extensions
 2630 for details.
 2631
 2632 NOTE: The names in the Name column in the above table are only for
 2633 reference within this document, and are not present in the actual
 2634 representation of the authenticator data.
 2635
 2636 The RP ID is originally received from the client when the credential is
 2637 created, and again when an assertion is generated. However, it differs
 2638 from other client data in some important ways. First, unlike the client
 2639 data, the RP ID of a credential does not change between operations but
 2640 instead remains the same for the lifetime of that credential. Secondly,
 2641 it is validated by the authenticator during the
 2642 authenticatorGetAssertion operation, by verifying that the RP ID
 2643 associated with the requested credential exactly matches the RP ID
 2644 supplied by the client, and that the RP ID is a registrable domain
 2645 suffix of or is equal to the effective domain of the RP's origin's
 2646 effective domain.
 2647
 2648 The UP flag SHALL be set if and only if the authenticator detected a
 2649 user through an authenticator specific gesture. The RFU bits SHALL be
 2650 set to zero.
 2651
 2652 For attestation signatures, the authenticator MUST set the AT flag and
 2653 include the attestedCredentialData. For authentication signatures, the
 2654 AT flag MUST NOT be set and the attestedCredentialData MUST NOT be
 2655 included.
 2656
 2657 If the authenticator does not include any extension data, it MUST set
 2658 the ED flag to zero, and to one if extension data is included.
 2659
 2660 The figure below shows a visual representation of the authenticator
 2661 data structure.
 2662 [\[fido-signature-formats-figure1.html\]](#) Authenticator data layout.

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
 2480 bytes plus the length of the attested credential data, plus the length
 2481 of the CBOR map that follows.

2482 **6.1.1. Signature Counter Considerations**

2483 Authenticators **MUST** implement a signature counter feature. The
 2484 signature counter is incremented for each successful
 2485 authenticatorGetAssertion operation by some positive value, and its
 2486 value is returned to the Relying Party within the authenticator data.
 2487 The signature counter's purpose is to aid Relying Parties in detecting
 2488 cloned authenticators. Clone detection is more important for
 2489 authenticators with limited protection measures.

2490 An Relying Party stores the signature counter of the most recent
 2491 authenticatorGetAssertion operation. Upon a new
 2492 authenticatorGetAssertion operation, the Relying Party compares the
 2493 stored signature counter value with the new signCount value returned in
 2494 the assertion's authenticator data. If this new signCount value is less
 2495 than or equal to the stored value, a cloned authenticator may exist, or
 2496 the authenticator may be malfunctioning.

2500 Detecting a signature counter mismatch does not indicate whether the
 2501 current operation was performed by a cloned authenticator or the
 2502 original authenticator. Relying Parties should address this situation
 2503 appropriately relative to their individual situations, i.e., their risk
 2504 tolerance.

2505 **Authenticators:**

- 2506 * should implement per-RP ID signature counters. This prevents the
 2507 signature counter value from being shared between Relying Parties
 2508 and being possibly employed as a correlation handle for the user.
 2509 Authenticators may implement a global signature counter, i.e., on a
 2510 per-authenticator basis, but this is less privacy-friendly for
 2511 users.
- 2512 * should ensure that the signature counter value does not
 2513 accidentally decrease (e.g., due to hardware failures).

2514 **6.2. Authenticator operations**

2515 A **client must** connect to an authenticator in order to invoke **any of the**
 2516 operations of that authenticator. This connection defines an
 2517 authenticator session. An authenticator must maintain isolation between
 2518 sessions. It may do this by only allowing one session to exist at any
 2519 particular time, or by providing more complicated session management.

2520 The following operations can be invoked by the client in an
 2521 authenticator session.

2522 **6.2.1. The authenticatorMakeCredential operation**

2525 It takes the following input parameters:

2530

2531

2663 Note that the authenticator data describes its own length: If the AT
 2664 and ED flags are not set, it is always 37 bytes long. The attested
 2665 credential data (which is only present if the AT flag is set) describes
 2666 its own length. If the ED flag is set, then the total length is 37
 2667 bytes plus the length of the attested credential data, plus the length
 2668 of the CBOR map that follows.

2669 **6.1.1. Signature Counter Considerations**

2670 Authenticators **MUST** implement a signature counter feature. The
 2671 signature counter is incremented for each successful
 2672 authenticatorGetAssertion operation by some positive value, and its
 2673 value is returned to the Relying Party within the authenticator data.
 2674 The signature counter's purpose is to aid Relying Parties in detecting
 2675 cloned authenticators. Clone detection is more important for
 2676 authenticators with limited protection measures.

2677 An Relying Party stores the signature counter of the most recent
 2678 authenticatorGetAssertion operation. Upon a new
 2679 authenticatorGetAssertion operation, the Relying Party compares the
 2680 stored signature counter value with the new signCount value returned in
 2681 the assertion's authenticator data. If this new signCount value is less
 2682 than or equal to the stored value, a cloned authenticator may exist, or
 2683 the authenticator may be malfunctioning.

2684 Detecting a signature counter mismatch does not indicate whether the
 2685 current operation was performed by a cloned authenticator or the
 2686 original authenticator. Relying Parties should address this situation
 2687 appropriately relative to their individual situations, i.e., their risk
 2688 tolerance.

2689 **Authenticators:**

- 2690 * should implement per-RP ID signature counters. This prevents the
 2691 signature counter value from being shared between Relying Parties
 2692 and being possibly employed as a correlation handle for the user.
 2693 Authenticators may implement a global signature counter, i.e., on a
 2694 per-authenticator basis, but this is less privacy-friendly for
 2695 users.
- 2696 * should ensure that the signature counter value does not
 2697 accidentally decrease (e.g., due to hardware failures).

2698 **6.2. Authenticator operations**

2699 A **WebAuthn Client MUST** connect to an authenticator in order to invoke
 2700 **any of the** operations of that authenticator. This connection defines an
 2701 authenticator session. An authenticator must maintain isolation between
 2702 sessions. It may do this by only allowing one session to exist at any
 2703 particular time, or by providing more complicated session management.

2704 The following operations can be invoked by the client in an
 2705 authenticator session.

2706 **6.2.1. Lookup Credential Source by Credential ID algorithm**

2707 The result of looking up a credential id credentialId in an
 2708 authenticator authenticator is the result of the following algorithm:

- 2709 1. If authenticator can decrypt credentialId into a public key
 2710 credential source credSource:
 - 2711 1. Set credSource.id to credentialId.
 - 2712 2. Return credSource.
- 2713 2. For each public key credential source credSource of authenticator's
 2714 credentials map:
 - 2715 1. If credSource.id is credentialId, return credSource.
- 2716 3. Return null.

2717 **6.2.2. The authenticatorMakeCredential operation**

2718 It takes the following input parameters:

2719

2720

2721

2722

2723

2724

2725

2726

2727

2728

2729

2730

2731

2732

2532 hash
 2533 The hash of the serialized client data, provided by the client.
 2534
 2535 rpEntity
 2536 The Relying Party's PublicKeyCredentialRpEntity.
 2537
 2538 userEntity
 2539 The user account's PublicKeyCredentialUserEntity, containing the
 2540 user handle given by the Relying Party.
 2541
 2542 requireResidentKey
 2543 The authenticatorSelection.requireResidentKey value given by the
 2544 Relying Party.
 2545
 2546 requireUserPresence
 2547 A Boolean value provided by the client, which in invocations
 2548 from a WebAuthn Client's [[Create]](origin, options,
 2549 sameOriginWithAncestors) method is always set to the inverse of
 2550 requireUserVerification.
 2551
 2552 requireUserVerification
 2553 The effective user verification requirement for credential
 2554 creation, a Boolean value provided by the client.
 2555
 2556 credTypesAndPubKeyAlgs
 2557 A sequence of pairs of PublicKeyCredentialType and public key
 2558 algorithms (COSEAlgorithmIdentifier) requested by the Relying
 2559 Party. This sequence is ordered from most preferred to least
 2560 preferred. The platform makes a best-effort to create the most
 2561 preferred credential that it can.
 2562
 2563 excludeCredentialDescriptorList
 2564 An optional list of PublicKeyCredentialDescriptor objects
 2565 provided by the Relying Party with the intention that, if any of
 2566 these are known to the authenticator, it should not create a new
 2567 credential. excludeCredentialDescriptorList contains a list of
 2568 known credentials.
 2569
 2570 extensions
 2571 A map from extension identifiers to their authenticator
 2572 extension inputs, created by the client based on the extensions
 2573 requested by the Relying Party, if any.
 2574
 2575 Note: Before performing this operation, all other operations in
 2576 progress in the authenticator session **must** be aborted by running the
 2577 authenticatorCancel operation.
 2578
 2579 When this operation is invoked, the authenticator **must** perform the
 2580 following procedure:
 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 2. Check if at least one of the specified combinations of
 2585 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.

2733 hash
 2734 The hash of the serialized client data, provided by the client.
 2735
 2736 rpEntity
 2737 The Relying Party's PublicKeyCredentialRpEntity.
 2738
 2739 userEntity
 2740 The user account's PublicKeyCredentialUserEntity, containing the
 2741 user handle given by the Relying Party.
 2742
 2743 requireResidentKey
 2744 The authenticatorSelection.requireResidentKey value given by the
 2745 Relying Party.
 2746
 2747 requireUserPresence
 2748 A Boolean value provided by the client, which in invocations
 2749 from a WebAuthn Client's [[Create]](origin, options,
 2750 sameOriginWithAncestors) method is always set to the inverse of
 2751 requireUserVerification.
 2752
 2753 requireUserVerification
 2754 The effective user verification requirement for credential
 2755 creation, a Boolean value provided by the client.
 2756
 2757 credTypesAndPubKeyAlgs
 2758 A sequence of pairs of PublicKeyCredentialType and public key
 2759 algorithms (COSEAlgorithmIdentifier) requested by the Relying
 2760 Party. This sequence is ordered from most preferred to least
 2761 preferred. The platform makes a best-effort to create the most
 2762 preferred credential that it can.
 2763
 2764 excludeCredentialDescriptorList
 2765 An optional list of PublicKeyCredentialDescriptor objects
 2766 provided by the Relying Party with the intention that, if any of
 2767 these are known to the authenticator, it should not create a new
 2768 credential. excludeCredentialDescriptorList contains a list of
 2769 known credentials.
 2770
 2771 extensions
 2772 A CBOR map from extension identifiers to their authenticator
 2773 extension inputs, created by the client based on the extensions
 2774 requested by the Relying Party, if any.
 2775
 2776 Note: Before performing this operation, all other operations in
 2777 progress in the authenticator session **MUST** be aborted by running the
 2778 authenticatorCancel operation.
 2779
 2780 When this operation is invoked, the authenticator **MUST** perform the
 2781 following procedure:
 2782 1. Check if all the supplied parameters are syntactically well-formed
 2783 and of the correct length. If not, return an error code equivalent
 2784 to "UnknownError" and terminate the operation.
 2785 2. Check if at least one of the specified combinations of
 2786 PublicKeyCredentialType and cryptographic parameters in
 2787 credTypesAndPubKeyAlgs is supported. If not, return an error code
 2788 equivalent to "NotSupportedError" and terminate the operation.
 2789 3. For each descriptor of excludeCredentialDescriptorList:
 2790 1. If looking up descriptor.id in this authenticator returns
 2791 non-null, and the returned item's RP ID and type match
 2792 rpEntity.id and excludeCredentialDescriptorList.type
 2793 respectively, then obtain user consent for creating a new
 2794 credential. The method of obtaining user consent **MUST** include
 2795 a test of user presence. If the user
 2796
 2797 confirms consent to create a new credential
 2798 return an error code equivalent to
 2799 "InvalidStateError" and terminate the operation.
 2800
 2801 does not consent to create a new credential
 2802 return an error code equivalent to "NotAllowedError"

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.

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 a per-RP ID signature counter
 2635 allocate the counter, associate it with the RP ID, and
 2636 initialize the counter value as zero.
 2637
 2638

2803 and terminate the operation.
 2804
 2805 4. If requireResidentKey is true and the authenticator cannot store a
 2806 Client-side-resident Credential Private Key, return an error code
 2807 equivalent to "ConstraintError" and terminate the operation.
 2808 5. If requireUserVerification is true and the authenticator cannot
 2809 perform user verification, return an error code equivalent to
 2810 "ConstraintError" and terminate the operation.
 2811 6. Obtain user consent for creating a new credential. The prompt for
 2812 obtaining this consent is shown by the authenticator if it has its
 2813 own output capability, or by the user agent otherwise. The prompt
 2814 SHOULD display rpEntity.id, rpEntity.name, userEntity.name and
 2815 userEntity.displayName, if possible.
 2816 If requireUserVerification is true, the method of obtaining user
 2817 consent MUST include user verification.
 2818 If requireUserPresence is true, the method of obtaining user
 2819 consent MUST include a test of user presence.
 2820 If the user **does not** consent or if user verification fails, return
 2821 an error code equivalent to "NotAllowedError" and terminate the
 2822 operation.
 2823 7. Once user consent has been obtained, generate a new credential
 2824 object:
 2825 1. Let (publicKey, privateKey) be a new pair of cryptographic
 2826 **keys** using the combination of PublicKeyCredentialType and
 2827 cryptographic parameters represented by the first item in
 2828 credTypesAndPubKeyAlgs that is supported by this
 2829 authenticator.
 2830 2. Let userHandle be userEntity.id.
 2831 3. Let credentialSource be a new public key credential source
 2832 with the fields:
 2833 type
 2834 public-key.
 2835 privateKey
 2836 privateKey
 2837 rpId
 2838 rpEntity.id
 2839 userHandle
 2840 userHandle
 2841 otherUI
 2842 Any other information the authenticator chooses to
 2843 include.
 2844
 2845 4. If requireResidentKey is true or the authenticator chooses to
 2846 create a Client-side-resident Credential Private Key:
 2847 1. Let credentialId be a new credential id.
 2848 2. Set credentialSource.id to credentialId.
 2849 3. Let credentials be this authenticator's credentials map.
 2850 4. Set credentials[(rpEntity.id, userHandle)] to
 2851 credentialSource.
 2852 5. Otherwise:
 2853 1. Let credentialId be the result of serializing and
 2854 encrypting credentialSource so that only this
 2855 authenticator can decrypt it.
 2856 8. If any error occurred while creating the new credential object,
 2857 return an error code equivalent to "UnknownError" and terminate the
 2858 operation.
 2859 9. Let processedExtensions be the result of authenticator extension
 2860 processing for each supported extension identifier -> **authenticator**
 2861 **extension input in** extensions.
 2862 10. If the authenticator supports:
 2863 a per-RP ID signature counter
 2864 allocate the counter, associate it with the RP ID, and
 2865 initialize the counter value as zero.
 2866
 2867
 2868
 2869
 2870
 2871
 2872

2635 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
 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 `rpld` (as determined by an exact match
 2707 of `rpld`).
 2708 4. Remove any items from `allowCredentialDescriptorList` that do not

2873 a global signature counter
 2874 Use the global signature counter's actual value when
 2875 generating authenticator data.
 2876
 2877 a per credential signature counter
 2878 allocate the counter, associate it with the new
 2879 credential, and initialize the counter value as zero.
 2880
 2881 11. Let `attestedCredentialData` be the attested credential data byte
 2882 array including the `credentialId` and `publicKey`.
 2883 12. Let `authenticatorData` be the byte array specified in 6.1
 2884 Authenticator data, including `attestedCredentialData` as the
 2885 `attestedCredentialData` and `processedExtensions`, if any, as the
 2886 extensions.
 2887 13. Return the attestation object for the new credential created by the
 2888 procedure specified in 6.3.4 Generating an Attestation Object
 2889 using an authenticator-chosen attestation statement format,
 2890 `authenticatorData`, and hash. For more details on attestation, see
 2891 6.3 Attestation.
 2892
 2893 On successful completion of this operation, the authenticator returns
 2894 the attestation object to the client.
 2895
 2896 6.2.3. The `authenticatorGetAssertion` operation
 2897
 2898 It takes the following input parameters:
 2899
 2900 `rpld`
 2901 The caller's RP ID, as determined by the user agent and the
 2902 client.
 2903
 2904 `hash`
 2905 The hash of the serialized client data, provided by the client.
 2906
 2907 `allowCredentialDescriptorList`
 2908 An optional list of `PublicKeyCredentialDescriptors` describing
 2909 credentials acceptable to the Relying Party (possibly filtered
 2910 by the client), if any.
 2911
 2912 `requireUserPresence`
 2913 A Boolean value provided by the client, which in invocations
 2914 from a `WebAuthn Client's` `[[DiscoverFromExternalSource]](origin,`
 2915 `options, sameOriginWithAncestors)` method is always set to the
 2916 inverse of `requireUserVerification`.
 2917
 2918 `requireUserVerification`
 2919 The effective user verification requirement for assertion, a
 2920 Boolean value provided by the client.
 2921
 2922 `extensions`
 2923 A **CBOR** map from extension identifiers to their authenticator
 2924 extension inputs, created by the client based on the extensions
 2925 requested by the Relying Party, if any.
 2926
 2927 Note: Before performing this operation, all other operations in
 2928 progress in the authenticator session must be aborted by running the
 2929 `authenticatorCancel` operation.
 2930
 2931 When this method is invoked, the authenticator **MUST** perform the
 2932 following procedure:
 2933 1. Check if all the supplied parameters are syntactically well-formed
 2934 and of the correct length. If not, return an error code equivalent
 2935 to "UnknownError" and terminate the operation.
 2936 2. Let `credentialOptions` be a new empty set of public key credential
 2937 sources.
 2938 3. If `allowCredentialDescriptorList` was supplied, then for each
 2939 descriptor of `allowCredentialDescriptorList`:
 2940 1. Let `credSource` be the result of looking up `descriptor.id` in
 2941 this authenticator.
 2942 2. If `credSource` is not null, append it to `credentialOptions`.

2705 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
 2720 is greater than 1
 2721 Prompt the user to select selectedCredential from the
 2722 credentials matching the items in
 2723 allowCredentialDescriptorList.
 2724
 2725

2726 7. Obtain user consent for using selectedCredential. The prompt for
 2727 obtaining this consent may be shown by the authenticator if it has
 2728 its own output capability, or by the user agent otherwise. The
 2729 prompt SHOULD display the rpid and any additional displayable data
 2730 associated with selectedCredential, if possible.

2731 If requireUserVerification is true, the method of obtaining user
 2732 consent MUST include user verification.

2733 If requireUserPresence is true, the method of obtaining user
 2734 consent MUST include a test of user presence.

2735 If the user denies consent or if user verification fails, return an
 2736 error code equivalent to "NotAllowedError" and terminate the
 2737 operation.

2738 8. Let processedExtensions be the result of authenticator extension
 2739 processing for each supported extension identifier/input pair in
 2740 extensions.

2741 9. Increment the RP ID-associated signature counter or the global
 2742 signature counter value, depending on which approach is implemented
 2743 by the authenticator, by some positive value.

2744 10. Let authenticatorData be the byte array specified in 6.1
 2745 Authenticator data including processedExtensions, if any, as the
 2746 extensions and excluding attestedCredentialData.

2747 11. Let signature be the assertion signature of the concatenation
 2748 authenticatorData || hash using the private key of
 2749 selectedCredential as shown in Figure 2, below. A simple,
 2750 undelimited concatenation is safe to use here because the
 2751 authenticator data describes its own length. The hash of the
 2752 serialized client data (which potentially has a variable length) is
 2753 always the last element.

2754 **Generating an assertion signature** Generating an assertion
 2755 signature.

2756 12. If any error occurred while generating the assertion signature,
 2757 return an error code equivalent to "UnknownError" and terminate the
 2758 operation.

2759 13. Return to the user agent:
 2760 + selectedCredential's credential ID, if either a list of
 2761 credentials of size 2 or greater was supplied by the client,
 2762 or no such list was supplied. Otherwise, return only the below
 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

2943 4. Otherwise (allowCredentialDescriptorList was not supplied), for
 2944 each key -> credSource of this authenticator's credentials map,
 2945 append credSource to credentialOptions.
 2946 5. Remove any items from credentialOptions whose rpid is not equal to
 2947 rpid.
 2948 6. If credentialOptions is now empty, return an error code equivalent
 2949 to "NotAllowedError" and terminate the operation.
 2950 7. Prompt the user to select a public key credential source
 2951 selectedCredential from credentialOptions. Obtain user consent for
 2952 using selectedCredential. The prompt for obtaining this consent may
 2953 be shown by the authenticator if it has its own output capability,
 2954 or by the user agent otherwise.

2955 If requireUserVerification is true, the method of obtaining user
 2956 consent MUST include user verification.
 2957 If requireUserPresence is true, the method of obtaining user
 2958 consent MUST include a test of user presence.
 2959 If the user does not consent, return an error code equivalent to
 2960 "NotAllowedError" and terminate the operation.

2961 8. Let processedExtensions be the result of authenticator extension
 2962 processing for each supported extension identifier -> authenticator
 2963 extension input in extensions.

2964 9. Increment the RP ID-associated signature counter or the global
 2965 signature counter value, depending on which approach is implemented
 2966 by the authenticator, by some positive value.

2967 10. Let authenticatorData be the byte array specified in 6.1
 2968 Authenticator data including processedExtensions, if any, as the
 2969 extensions and excluding attestedCredentialData.

2970 11. Let signature be the assertion signature of the concatenation
 2971 authenticatorData || hash using the privateKey of
 2972 selectedCredential as shown in Figure 2, below. A simple,
 2973 undelimited concatenation is safe to use here because the
 2974 authenticator data describes its own length. The hash of the
 2975 serialized client data (which potentially has a variable length) is
 2976 always the last element.
 2977 **[[fido-signature-formats-figure2.html](#)]** Generating an assertion
 2978 signature.

2979 12. If any error occurred while generating the assertion signature,
 2980 return an error code equivalent to "UnknownError" and terminate the
 2981 operation.

2982 13. Return to the user agent:
 2983 + selectedCredential.id, if either a list of credentials (i.e.,
 2984 allowCredentialDescriptorList) of length 2 or greater was
 2985 supplied by the client, or no such list was supplied.
 2986 Note: If, within allowCredentialDescriptorList, the client
 2987 supplied exactly one credential and it was successfully
 2988 employed, then its credential ID is not returned since the
 2989 client already knows it. This saves transmitting these bytes
 2990 over what may be a constrained connection in what is likely a
 2991 common case.
 2992 + authenticatorData
 2993 + signature
 2994 + selectedCredential.userHandle
 2995 Note: the returned userHandle value may be null, see:
 2996 userHandleResult.
 2997

2998
 2999 If the authenticator cannot find any credential corresponding to the
 3000 specified Relying Party that matches the specified criteria, it
 3001 terminates the operation and returns an error.

2777 6.2.3. The authenticatorCancel operation
 2778
 2779 This operation takes no input parameters and returns no result.
 2780
 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.

6.3. Attestation

2792 Authenticators **must** also provide some form of attestation. The basic
 2793 requirement is that the authenticator can produce, for each credential
 2794 public key, an attestation statement verifiable by the Relying Party.
 2795 Typically, this attestation statement contains a signature by an
 2796 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 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 In general, there is no simple mapping between attestation statement
 2843 formats and attestation types. For example, the "packed" attestation
 2844 statement format defined in 8.2 Packed Attestation Statement Format
 2845 can be used in conjunction with all attestation types, while other
 2846

3002 6.2.4. The authenticatorCancel operation
 3003
 3004 This operation takes no input parameters and returns no result.
 3005
 3006 When this operation is invoked by the client in an authenticator
 3007 session, it has the effect of terminating any
 3008 authenticatorMakeCredential or authenticatorGetAssertion operation
 3009 currently in progress in that authenticator session. The authenticator
 3010 stops prompting for, or accepting, any user input related to
 3011 authorizing the canceled operation. The client ignores any further
 3012 responses from the authenticator for the canceled operation.
 3013
 3014 This operation is ignored if it is invoked in an authenticator session
 3015 which does not have an authenticatorMakeCredential or
 3016 authenticatorGetAssertion operation currently in progress.

6.3. Attestation

3017 Authenticators **MUST** also provide some form of attestation. The basic
 3018 requirement is that the authenticator can produce, for each credential
 3019 public key, an attestation statement verifiable by the Relying Party.
 3020 Typically, this attestation statement contains a signature by an
 3021 attestation private key over the attested credential public key and a
 3022 challenge, as well as a certificate or similar data providing
 3023 provenance information for the attestation public key, enabling the
 3024 Relying Party to make a trust decision. However, if an attestation key
 3025 pair is not available, then the authenticator **MUST** perform self
 3026 attestation of the credential public key with the corresponding
 3027 credential private key. All this information is returned by
 3028 authenticators any time a new public key credential is generated, in
 3029 the overall form of an attestation object. The relationship of the
 3030 attestation object with authenticator data (containing attested
 3031 credential data) and the attestation statement is illustrated in figure
 3032 3, below.
 3033 **Attestation Object Layout diagram Attestation object layout**
 3034 **illustrating the included authenticator data (containing attested**
 3035 **credential data) and the attestation statement.**

3036 This figure illustrates only the packed attestation statement format.
 3037 Several additional attestation statement formats are defined in 8
 3038 Defined Attestation Statement Formats.

3039 An important component of the attestation object is the attestation
 3040 statement. This is a specific type of signed data object, containing
 3041 statements about a public key credential itself and the authenticator
 3042 that created it. It contains an attestation signature created using the
 3043 key of the attesting authority (except for the case of self
 3044 attestation, when it is created using the credential private key). In
 3045 order to correctly interpret an attestation statement, a Relying Party
 3046 needs to understand these two aspects of attestation:
 3047 1. The attestation statement format is the manner in which the
 3048 signature is represented and the various contextual bindings are
 3049 incorporated into the attestation statement by the authenticator.
 3050 In other words, this defines the syntax of the statement. Various
 3051 existing devices and platforms (such as TPMs and the Android OS)
 3052 have previously defined attestation statement formats. This
 3053 specification supports a variety of such formats in an extensible
 3054 way, as defined in 6.3.2 Attestation Statement Formats.
 3055 2. The attestation type defines the semantics of attestation
 3056 statements and their underlying trust models. Specifically, it
 3057 defines how a Relying Party establishes trust in a particular
 3058 attestation statement, after verifying that it is cryptographically
 3059 valid. This specification supports a number of attestation types,
 3060 as described in 6.3.3 Attestation Types.

3061 In general, there is no simple mapping between attestation statement
 3062 formats and attestation types. For example, the "packed" attestation
 3063 statement format defined in 8.2 Packed Attestation Statement Format
 3064 can be used in conjunction with all attestation types, while other
 3065
 3066
 3067
 3068
 3069
 3070

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.

6.3.1. Attested credential data

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

| Name | Length (in bytes) | Description |
|--------------------|-------------------|----------------------------------|
| aaguid | 16 | The AAGUID of the authenticator. |
| credentialIdLength | 2 | Byte length L of Credential ID |

credentialId L Credential ID
 credentialPublicKey variable The credential public key encoded in COSE_Key format, as defined in Section 7 of [RFC8152]. **The encoded credential public key MUST contain the "alg" parameter and MUST NOT contain any other optional parameters. The "alg" parameter MUST contain a COSEAlgorithmIdentifier value.**

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

2883
2884
2885
2886
2887

3071 formats and types have more limited applicability.
 3072
 3073 The privacy, security and operational characteristics of attestation
 3074 depend on:
 3075 * The attestation type, which determines the trust model,
 3076 * The attestation statement format, which **MAY** constrain the strength
 3077 of the attestation by limiting what can be expressed in an
 3078 attestation statement, and
 3079 * The characteristics of the individual authenticator, such as its
 3080 construction, whether part or all of it runs in a secure operating
 3081 environment, and so on.
 3082
 3083 It is expected that most authenticators will support a small number of
 3084 attestation types and attestation statement formats, while Relying
 3085 Parties will decide what attestation types are acceptable to them by
 3086 policy. Relying Parties will also need to understand the
 3087 characteristics of the authenticators that they trust, based on
 3088 information they have about these authenticators. For example, the FIDO
 3089 Metadata Service [FIDOMetadataService] provides one way to access such
 3090 information.

6.3.1. Attested credential data

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

| Name | Length (in bytes) | Description |
|--------------------|-------------------|--|
| aaguid | 16 | The AAGUID of the authenticator. |
| credentialIdLength | 2 | Byte length L of Credential ID, 16-bit unsigned big-endian integer. |

credentialId L Credential ID
 credentialPublicKey variable The credential public key encoded in COSE_Key format, as defined in Section 7 of [RFC8152], **using the CTAP2 canonical CBOR encoding form. The COSE_Key-encoded credential public key MUST contain the optional "alg" parameter and MUST NOT contain any other optional parameters. The "alg" parameter MUST contain a COSEAlgorithmIdentifier value. The encoded credential public key MUST also contain any additional required parameters stipulated by the relevant key type specification, i.e., required for the key type "kty" and algorithm "alg" (see Section 8 of [RFC8152]).**

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

6.3.1.1. Examples of credentialPublicKey Values encoded in COSE_Key format

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

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

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

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

2888 6.3.2. Attestation Statement Formats

2888 As described above, an attestation statement format is a data format
2889 which represents a cryptographic signature by an authenticator over a
2890 set of contextual bindings. Each attestation statement format MUST be
2891 defined using the following template:

- 2892 * Attestation statement format identifier:
2893 * Supported attestation types:
2894 * Syntax: The syntax of an attestation statement produced in this
2895 format, defined using [CDDL] for the extension point \$attStmtFormat
2896 defined in 6.3.4 Generating an Attestation Object.
2897 * Signing procedure: The signing procedure for computing an
2898 attestation statement in this format given the public key
2899 credential to be attested, the authenticator data structure
2900 containing the authenticator data for the attestation, and the hash
2901 of the serialized client data.
2902 * Verification procedure: The procedure for verifying an attestation
2903 statement, which takes the following verification procedure inputs:
2904 + attStmt: The attestation statement structure
2905 + authenticatorData: The authenticator data claimed to have been
2906 used for the attestation
2907 + clientDataHash: The hash of the serialized client data
2908 The procedure returns either:
2909 + An error indicating that the attestation is invalid, or
2910 + The attestation type, and the trust path. This attestation
2911 trust path is either empty (in case of self attestation), an

```
3141 ; e.g., in hex: 1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9e
3142 ecd0084d19c
3143 }
3144
3145 Below is the above Elliptic Curve public key encoded in the CTAP2
3146 canonical CBOR encoding form, whitespace and line breaks are included
3147 here for clarity and to match the [CDDL] presentation above:
3148 A5
3149 01 02
3150
3151 03 26
3152
3153 20 01
3154
3155 21 58 20 65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de108de439c08551d
3156
3157 22 58 20 1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9eecd0084d19c
3158
```

3159 Below is an example of a COSE_Key-encoded 2048-bit RSA public key (see
3160 [RFC8230] Section 4), to be used with the PS256 signature algorithm
3161 (RSASSA-PSS with SHA-256, see [RFC8230] Section 2):

```
3162 {
3163 1: 3, ; kty: RSA key type
3164 3: -37, ; alg: PS256
3165 -1: n, ; n: RSA modulus n byte string 256 bytes in length
3166 ; e.g., in hex (middle bytes elided for brevity): DB5F651550...6
3167 DC6548ACC3
3168 -2: e ; e: RSA public exponent e byte string 3 bytes in length
3169 ; e.g., in hex: 010001
3170 }
3171
```

3172 Below is an example of the same COSE_Key-encoded RSA public key as
3173 above, to be used with the RS256 signature algorithm (RSASSA-PKCS1-v1_5
3174 with SHA-256, see 11.3 COSE Algorithm Registrations):

```
3175 {
3176 1: 3, ; kty: RSA key type
3177 3: -257, ; alg: RS256
3178 -1: n, ; n: RSA modulus n byte string 256 bytes in length
3179 ; e.g., in hex (middle bytes elided for brevity): DB5F651550...6
3180 DC6548ACC3
3181 -2: e ; e: RSA public exponent e byte string 3 bytes in length
3182 ; e.g., in hex: 010001
3183 }
3184
```

3185 6.3.2. Attestation Statement Formats

3186 As described above, an attestation statement format is a data format
3187 which represents a cryptographic signature by an authenticator over a
3188 set of contextual bindings. Each attestation statement format MUST be
3189 defined using the following template:

- 3190 * Attestation statement format identifier:
3191 * Supported attestation types:
3192 * Syntax: The syntax of an attestation statement produced in this
3193 format, defined using [CDDL] for the extension point \$attStmtFormat
3194 defined in 6.3.4 Generating an Attestation Object.
3195 * Signing procedure: The signing procedure for computing an
3196 attestation statement in this format given the public key
3197 credential to be attested, the authenticator data structure
3198 containing the authenticator data for the attestation, and the hash
3199 of the serialized client data.
3200 * Verification procedure: The procedure for verifying an attestation
3201 statement, which takes the following verification procedure inputs:
3202 + attStmt: The attestation statement structure
3203 + authenticatorData: The authenticator data claimed to have been
3204 used for the attestation
3205 + clientDataHash: The hash of the serialized client data
3206 The procedure returns either:
3207 + An error indicating that the attestation is invalid, or
3208 + The attestation type, and the trust path. This attestation
3209 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
 2920 6.3.3. Attestation Types
 2921
 2922 WebAuthn supports multiple attestation types:
 2923
 2924 **Basic Attestation**
 2925 In the case of basic attestation [UAFProtocol], the
 2926 authenticator's attestation key pair is specific to an
 2927 authenticator model. Thus, authenticators of the same model
 2928 often share the same attestation key pair. See 6.3.5.1 Privacy
 2929 for further information.
 2930
 2931 **Self Attestation**
 2932 In the case of self attestation, also known as surrogate basic
 2933 attestation [UAFProtocol], the Authenticator does not have any
 2934 specific attestation key. Instead it uses the credential private
 2935 key to create the attestation signature. Authenticators without
 2936 meaningful protection measures for an attestation private key
 2937 typically use this attestation type.
 2938
 2939 **Privacy CA**
 2940 In this case, the Authenticator owns an authenticator-specific
 2941 (endorsement) key. This key is used to securely communicate with
 2942 a trusted third party, the Privacy CA. The Authenticator can
 2943 generate multiple attestation key pairs and asks the Privacy CA
 2944 to issue an attestation certificate for it. Using this approach,
 2945 the Authenticator can limit the exposure of the endorsement key
 2946 (which is a global correlation handle) to Privacy CA(s).
 2947 Attestation keys can be requested for each public key credential
 2948 individually.
 2949
 2950
 2951 Note: This concept typically leads to multiple attestation
 2952 certificates. The attestation certificate requested most
 2953 recently is called "active".
 2954
 2955 **Elliptic Curve based Direct Anonymous Attestation (ECDAAs)**
 2956 In this case, the Authenticator receives direct anonymous
 2957 attestation (DAA) credentials from a single DAA-Issuer. These
 2958 DAA credentials are used along with blinding to sign the
 2959 attested credential data. The concept of blinding avoids the DAA
 2960 credentials being misused as global correlation handle. WebAuthn
 2961 supports DAA using elliptic curve cryptography and bilinear
 2962 pairings, called ECDAAs (see [FIDOEcdaaAlgorithm]) in this
 2963 specification. Consequently we denote the DAA-Issuer as
 2964 ECDAAs-Issuer (see [FIDOEcdaaAlgorithm]).
 2965
 2966 6.3.4. Generating an Attestation Object
 2967
 2968 To generate an attestation object (see: Figure 3) given:
 2969
 2970 attestationFormat
 2971 An attestation statement format.
 2972
 2973 authData
 2974 A byte array containing authenticator data.
 2975
 2976 hash
 2977 The hash of the serialized client data.

3211 identifier of an ECDAAs-issuer public key (in the case of
 3212 ECDAAs), or a set of X.509 certificates.
 3213
 3214 The initial list of specified attestation statement formats is in 8
 3215 Defined Attestation Statement Formats.
 3216
 3217 6.3.3. Attestation Types
 3218
 3219 WebAuthn supports multiple attestation types:
 3220
 3221 **Basic Attestation (Basic)**
 3222 In the case of basic attestation [UAFProtocol], the
 3223 authenticator's attestation key pair is specific to an
 3224 authenticator model. Thus, authenticators of the same model
 3225 often share the same attestation key pair. See 14.1 Attestation
 3226 Privacy for further information.
 3227
 3228 **Self Attestation (Self)**
 3229 In the case of self attestation, also known as surrogate basic
 3230 attestation [UAFProtocol], the Authenticator does not have any
 3231 specific attestation key. Instead it uses the credential private
 3232 key to create the attestation signature. Authenticators without
 3233 meaningful protection measures for an attestation private key
 3234 typically use this attestation type.
 3235
 3236 **Attestation CA (AttCA)**
 3237 In this case, an authenticator is based on a Trusted Platform
 3238 Module (TPM) and holds an authenticator-specific "endorsement
 3239 key" (EK). This key is used to securely communicate with a
 3240 trusted third party, the Attestation CA
 3241 [TCG-CMCPProfile-AIKCertEnroll] (formerly known as a "Privacy
 3242 CA"). The authenticator can generate multiple attestation
 3243 identity key pairs (AIK) and requests an Attestation CA to issue
 3244 an AIK certificate for each. Using this approach, such an
 3245 authenticator can limit the exposure of the EK (which is a
 3246 global correlation handle) to Attestation CA(s). AIKs can be
 3247 requested for each authenticator-generated public key credential
 3248 individually, and conveyed to Relying Parties as attestation
 3249 certificates.
 3250
 3251 Note: This concept typically leads to multiple attestation
 3252 certificates. The attestation certificate requested most
 3253 recently is called "active".
 3254
 3255 **Elliptic Curve based Direct Anonymous Attestation (ECDAAs)**
 3256 In this case, the Authenticator receives direct anonymous
 3257 attestation (DAA) credentials from a single DAA-Issuer. These
 3258 DAA credentials are used along with blinding to sign the
 3259 attested credential data. The concept of blinding avoids the DAA
 3260 credentials being misused as global correlation handle. WebAuthn
 3261 supports DAA using elliptic curve cryptography and bilinear
 3262 pairings, called ECDAAs (see [FIDOEcdaaAlgorithm]) in this
 3263 specification. Consequently we denote the DAA-Issuer as
 3264 ECDAAs-Issuer (see [FIDOEcdaaAlgorithm]).
 3265
 3266 **No attestation statement (None)**
 3267 In this case, no attestation information is available.
 3268
 3269 6.3.4. Generating an Attestation Object
 3270
 3271 To generate an attestation object (see: Figure 3) given:
 3272
 3273 attestationFormat
 3274 An attestation statement format.
 3275
 3276 authData
 3277 A byte array containing authenticator data.
 3278
 3279 hash
 3280 The hash of the serialized client data.

```

2977 the authenticator MUST:
2978 1. Let attStmt be the result of running attestationFormat's signing
2979 procedure given authData and hash.
2980 2. Let fmt be attestationFormat's attestation statement format
2981 identifier
2982 3. Return the attestation object as a CBOR map with the following
2983 syntax, filled in with variables initialized by this algorithm:
2984 attObj = {
2985   authData: bytes,
2986   $$attStmtType
2987 }
2988
2989 attStmtTemplate = (
2990   fmt: text,
2991   attStmt: { * tstr => any } ; Map is filled in by each
2992 concrete attStmtType
2993 )
2994
2995 ; Every attestation statement format must have the above fields
2996 attStmtTemplate .within $$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 ECDAA-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 attestation keys for their devices can issue new attestation certificates for these keys from a new intermediate CA or from a new root CA. If the root CA changes, the Relying Parties must update their trusted root certificates accordingly.

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

```

3281 the authenticator MUST:
3282 1. Let attStmt be the result of running attestationFormat's signing
3283 procedure given authData and hash.
3284 2. Let fmt be attestationFormat's attestation statement format
3285 identifier
3286 3. Return the attestation object as a CBOR map with the following
3287 syntax, filled in with variables initialized by this algorithm:
3288 attObj = {
3289   authData: bytes,
3290   $$attStmtType
3291 }
3292
3293 attStmtTemplate = (
3294   fmt: text,
3295   attStmt: { * tstr => any } ; Map is filled in by each
3296 concrete attStmtType
3297 )
3298
3299 ; Every attestation statement format must have the above fields
3300 attStmtTemplate .within $$attStmtType
3301
3302 6.3.5. Signature Formats for Packed Attestation, FIDO U2F Attestation, and
3303 Assertion Signatures
3304

```

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

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

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 AuthenticatorAttestationResponse or AuthenticatorAssertionResponse
3083 structure, respectively, from the client. It must then deliver the
3084 contents of this structure to the Relying Party server, using methods
3085 outside the scope of this specification. This section describes the
3086 operations that the Relying Party must perform upon receipt of these
3087 structures.
3088
3089

3090 7.1. Registering a new credential

3091 When registering a new credential, represented by a
3092 AuthenticatorAttestationResponse structure, as part of a registration
3093 ceremony, a Relying Party MUST proceed as follows:
3094 1. Perform JSON deserialization on the clientDataJSON field of the
3095 AuthenticatorAttestationResponse object to extract the client data
3096 C claimed as collected during the credential creation.
3097 2. Verify that the type in C is the string webauthn.create.
3098
3099

3305
3306 * For COSEAlgorithmIdentifier -7 (ES256), and other ECDSA-based
3307 algorithms, a signature value is encoded as an ASN.1 DER
3308 Ecdsa-Sig-Value, as defined in [RFC3279] section 2.2.3.
3309 Example:
3310 30 44 ; SEQUENCE (68 Bytes)
3311 02 20 ; INTEGER (32 Bytes)
3312 | 3d 46 28 7b 8c 6e 8c 8c 26 1c 1b 88 f2 73 b0 9a
3313 | 32 a6 cf 28 09 fd 6e 30 d5 a7 9f 26 37 00 8f 54
3314 02 20 ; INTEGER (32 Bytes)
3315 | 4e 72 23 6e a3 90 a9 a1 7b cf 5f 7a 09 d6 3a b2
3316 | 17 6c 92 bb 8e 36 c0 41 98 a2 7b 90 9b 6e 8f 13
3317

Note: As CTAP1/U2F devices are already producing signatures values
in this format, CTAP2 devices will also produce signatures values
in the same format, for consistency reasons. It is recommended that
any new attestation formats defined not use ASN.1 encodings, but
instead represent signatures as equivalent fixed-length byte arrays
without internal structure, using the same representations as used
by COSE signatures as defined in [RFC8152] and [RFC8230].
* For COSEAlgorithmIdentifier -257 (RS256), sig contains the
signature generated using the RSASSA-PKCS1-v1_5 signature scheme
defined in section 8.2.1 in [RFC8017] with SHA-256 as the hash
function. The signature is not ASN.1 wrapped.
* For COSEAlgorithmIdentifier -37 (PS256), sig contains the signature
generated using the RSASSA-PSS signature scheme defined in section
8.1.1 in [RFC8017] with SHA-256 as the hash function. The signature
is not ASN.1 wrapped.

3318 7. Relying Party Operations

3319 Upon successful execution of create() or get(), the Relying Party's
3320 script receives a PublicKeyCredential containing an
3321 AuthenticatorAttestationResponse or AuthenticatorAssertionResponse
3322 structure, respectively, from the client. It must then deliver the
3323 contents of this structure to the Relying Party server, using methods
3324 outside the scope of this specification. This section describes the
3325 operations that the Relying Party must perform upon receipt of these
3326 structures.
3327
3328

3329 7.1. Registering a new credential

3330 When registering a new credential, represented by an
3331 AuthenticatorAttestationResponse structure response and an
3332 AuthenticationExtensionsClientOutputs structure clientExtensionResults,
3333 as part of a registration ceremony, a Relying Party MUST proceed as
3334 follows:
3335 1. Let JSONtext be the result of running UTF-8 decode on the value of
3336 response.clientDataJSON.
3337

- 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.
- 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 ECDAA-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 ECDAA was used, verify that the identifier of the

- 3354 Note: Using any implementation of UTF-8 decode is acceptable as
- 3355 long as it yields the same result as that yielded by the UTF-8
- 3356 decode algorithm. In particular, any leading byte order mark (BOM)
- 3357 MUST be stripped.
- 3358 2. Let C, the client data claimed as collected during the credential
- 3359 creation, be the result of running an implementation-specific JSON
- 3360 parser on JSONtext.
- 3361 Note: C may be any implementation-specific data structure
- 3362 representation, as long as C's components are referenceable, as
- 3363 required by this algorithm.
- 3364 3. Verify that the value of C.type is webauthn.create.
- 3365 4. Verify that the value of C.challenge matches the challenge that was
- 3366 sent to the authenticator in the create() call.
- 3367 5. Verify that the value of C.origin matches the Relying Party's
- 3368 origin.
- 3369 6. Verify that the value of C.tokenBinding.status matches the state of
- 3370 Token Binding for the TLS connection over which the assertion was
- 3371 obtained. If Token Binding was used on that TLS connection, also
- 3372 verify that C.tokenBinding.id matches the base64url encoding of the
- 3373 Token Binding ID for the connection.
- 3374 7. Compute the hash of response.clientDataJSON using SHA-256.
- 3375 8. Perform CBOR decoding on the attestationObject field of the
- 3376 AuthenticatorAttestationResponse structure to obtain the
- 3377 attestation statement format fmt, the authenticator data authData,
- 3378 and the attestation statement attStmt.
- 3379 9. Verify that the RP ID hash in authData is indeed the SHA-256 hash
- 3380 of the RP ID expected by the RP.
- 3381 10. If user verification is required for this registration, verify that
- 3382 the User Verified bit of the flags in authData is set.
- 3383 11. If user verification is not required for this registration, verify
- 3384 that the User Present bit of the flags in authData is set.
- 3385 12. Verify that the values of the client extension outputs in
- 3386 clientExtensionResults and the authenticator extension outputs in
- 3387 the extensions in authData are as expected, considering the client
- 3388 extension input values that were given as the extensions option in
- 3389 the create() call. In particular, any extension identifier values
- 3390 in the clientExtensionResults and the extensions in authData MUST
- 3391 be also be present as extension identifier values in the extensions
- 3392 member of options, i.e., no extensions are present that were not
- 3393 requested. In the general case, the meaning of "are as expected" is
- 3394 specific to the Relying Party and which extensions are in use.
- 3395 Note: Since all extensions are OPTIONAL for both the client and the
- 3396 authenticator, the Relying Party MUST be prepared to handle cases
- 3397 where none or not all of the requested extensions were acted upon.
- 3398 13. Determine the attestation statement format by performing a USASCII
- 3399 case-sensitive match on fmt against the set of supported WebAuthn
- 3400 Attestation Statement Format Identifier values. The up-to-date list
- 3401 of registered WebAuthn Attestation Statement Format Identifier
- 3402 values is maintained in the in the IANA registry of the same name
- 3403 [WebAuthn-Registries].
- 3404 14. Verify that attStmt is a correct attestation statement, conveying a
- 3405 valid attestation signature, by using the attestation statement
- 3406 format fmt's verification procedure given attStmt, authData and the
- 3407 hash of the serialized client data computed in step 7.
- 3408 Note: Each attestation statement format specifies its own
- 3409 verification procedure. See 8 Defined Attestation Statement
- 3410 Formats for the initially-defined formats, and
- 3411 [WebAuthn-Registries] for the up-to-date list.
- 3412 15. If validation is successful, obtain a list of acceptable trust
- 3413 anchors (attestation root certificates or ECDAA-Issuer public keys)
- 3414 for that attestation type and attestation statement format fmt,
- 3415 from a trusted source or from policy. For example, the FIDO
- 3416 Metadata Service [FIDOMetadataService] provides one way to obtain
- 3417 such information, using the aaguid in the attestedCredentialData in
- 3418 authData.
- 3419 16. Assess the attestation trustworthiness using the outputs of the
- 3420 verification procedure in step 14, as follows:
- 3421 + If self attestation was used, check if self attestation is
- 3422 acceptable under Relying Party policy.
- 3423 + If ECDAA was used, verify that the identifier of the

- 3197 origin.
- 3198 7. Verify that the tokenBindingId member of C (if present) matches the
- 3199 Token Binding ID for the TLS connection over which the signature
- 3200 was obtained.
- 3201 8. Verify that the clientExtensions member of C is a subset of the
- 3202 extensions requested by the Relying Party and that the
- 3203 authenticatorExtensions in C is also a subset of the extensions
- 3204 requested by the Relying Party.
- 3205 9. Verify that the rpIdHash in aData is the SHA-256 hash of the RP ID
- 3206 expected by the Relying Party.
- 3207 10. Let hash be the result of computing a hash over the cData using the
- 3208 algorithm represented by the hashAlgorithm member of C.
- 3209 11. Using the credential public key looked up in step 1, verify that

3210 sig is a valid signature over the binary concatenation of aData and
3211 hash.

- 3212 12. If the signature counter value adata.signCount is nonzero or the
- 3213 value stored in conjunction with credential's id attribute is
- 3214 nonzero, then run the following substep:
- 3215 + If the signature counter value adata.signCount is
- 3216
- 3217 greater than the signature counter value stored in
- 3218 conjunction with credential's id attribute.
- 3219 Update the stored signature counter value,
- 3220 associated with credential's id attribute, to be the
- 3221 value of adata.signCount.
- 3222
- 3223 less than or equal to the signature counter value stored in
- 3224 conjunction with credential's id attribute.
- 3225 This is a signal that the authenticator may be
- 3226 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 WebAuthn supports pluggable attestation statement formats. This section
3241 defines an initial set of such formats.
3242

- 3487 representation, as long as C's components are referenceable, as
- 3488 required by this algorithm.
- 3489 7. Verify that the value of C.type is the string webauthn.get.
- 3490 8. Verify that the value of C.challenge matches the challenge that was
- 3491 sent to the authenticator in the PublicKeyCredentialRequestOptions
- 3492 passed to the get() call.
- 3493 9. Verify that the value of C.origin matches the Relying Party's
- 3494 origin.
- 3495 10. Verify that the value of C.tokenBinding.status matches the state of
- 3496 Token Binding for the TLS connection over which the attestation was
- 3497 obtained. If Token Binding was used on that TLS connection, also
- 3498 verify that C.tokenBinding.id matches the base64url encoding of the
- 3499 Token Binding ID for the connection.
- 3500 11. Verify that the rpIdHash in aData is the SHA-256 hash of the RP ID

- 3501 expected by the Relying Party.
- 3502 12. If user verification is required for this assertion, verify that
- 3503 the User Verified bit of the flags in aData is set.
- 3504 13. If user verification is not required for this assertion, verify
- 3505 that the User Present bit of the flags in aData is set.
- 3506 14. Verify that the values of the client extension outputs in
- 3507 clientExtensionResults and the authenticator extension outputs in
- 3508 the extensions in authData are as expected, considering the client
- 3509 extension input values that were given as the extensions option in
- 3510 the get() call. In particular, any extension identifier values in
- 3511 the clientExtensionResults and the extensions in authData MUST be
- 3512 also be present as extension identifier values in the extensions
- 3513 member of options, i.e., no extensions are present that were not
- 3514 requested. In the general case, the meaning of "are as expected" is
- 3515 specific to the Relying Party and which extensions are in use.
- 3516 Note: Since all extensions are OPTIONAL for both the client and the
- 3517 authenticator, the Relying Party MUST be prepared to handle cases
- 3518 where none or not all of the requested extensions were acted upon.
- 3519 15. Let hash be the result of computing a hash over the cData using
- 3520 SHA-256.
- 3521 16. Using the credential public key looked up in step 3, verify that
- 3522 sig is a valid signature over the binary concatenation of aData and
- 3523 hash.
- 3524 17. If the signature counter value adata.signCount is nonzero or the
- 3525 value stored in conjunction with credential's id attribute is
- 3526 nonzero, then run the following sub-step:
- 3527 + If the signature counter value adata.signCount is
- 3528

- 3529 greater than the signature counter value stored in
- 3530 conjunction with credential's id attribute.
- 3531 Update the stored signature counter value,
- 3532 associated with credential's id attribute, to be the
- 3533 value of adata.signCount.
- 3534
- 3535 less than or equal to the signature counter value stored in
- 3536 conjunction with credential's id attribute.
- 3537 This is a signal that the authenticator may be
- 3538 cloned, i.e. at least two copies of the credential
- 3539 private key may exist and are being used in
- 3540 parallel. Relying Parties should incorporate this
- 3541 information into their risk scoring. Whether the
- 3542 Relying Party updates the stored signature counter
- 3543 value in this case, or not, or fails the
- 3544 authentication ceremony or not, is Relying
- 3545 Party-specific.
- 3546

- 3547 18. If all the above steps are successful, continue with the
- 3548 authentication ceremony as appropriate. Otherwise, fail the
- 3549 authentication ceremony.

3550 8. Defined Attestation Statement Formats

3551 WebAuthn supports pluggable attestation statement formats. This section
3552 defines an initial set of such formats.
3553
3554

3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310

8.1. Attestation Statement Format Identifiers

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

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

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

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

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

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

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

8.2. Packed Attestation Statement Format

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

Attestation statement format identifier
packed

Attestation types supported
All

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

```

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

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

3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3620
3621
3622
3623
3624

8.1. Attestation Statement Format Identifiers

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

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

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

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

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

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

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

8.2. Packed Attestation Statement Format

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

Attestation statement format identifier
packed

Attestation types supported
All

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

```

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

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

3311 }
 3312
 3313 The semantics of the fields are as follows:
 3314
 3315 alg
 3316 A COSEAlgorithmIdentifier containing the identifier of the
 3317 algorithm used to generate the attestation signature.
 3318
 3319 sig
 3320 A byte string containing the attestation signature.
 3321
 3322 x5c
 3323 The elements of this array contain the attestation
 3324 certificate and its certificate chain, each encoded in
 3325 X.509 format. The attestation certificate **must** be the
 3326 first element in the array.
 3327
 3328 ecdaaKeyId
 3329 The identifier of the ECDAA-Issuer public key. This is the
 3330 BigIntegerToB encoding of the component "c" of the
 3331 ECDAA-Issuer public key as defined section 3.3, step 3.5
 3332 in [FIDOEcdaaAlgorithm].
 3333
 3334 Signing procedure
 3335 The signing procedure for this attestation statement format is
 3336 similar to the procedure for generating assertion signatures.
 3337
 3338 1. Let authenticatorData denote the authenticator data for the
 3339 attestation, and let clientDataHash denote the hash of the
 3340 serialized client data.
 3341 2. If Basic or **Privacy** CA attestation is in use, the
 3342 **authenticator produces the sig by concatenating**
 3343 **authenticatorData and clientDataHash, and signing the result**
 3344 **using an attestation private key selected through an**
 3345 **authenticator-specific mechanism. It sets x5c to the**
 3346 **certificate chain of the attestation public key and alg to the**
 3347 **algorithm of the attestation private key.**
 3348 3. If ECDAA is in use, the authenticator produces sig by
 3349 concatenating authenticatorData and clientDataHash, and
 3350 signing the result using ECDAA-Sign (see section 3.5 of
 3351 [FIDOEcdaaAlgorithm]) after selecting an ECDAA-Issuer public
 3352 key related to the ECDAA signature private key through an
 3353 authenticator-specific mechanism (see [FIDOEcdaaAlgorithm]).
 3354 It sets alg to the algorithm of the selected ECDAA-Issuer
 3355 public key and ecdaaKeyId to the identifier of the
 3356 ECDAA-Issuer public key (see above).
 3357 4. If self attestation is in use, the authenticator produces sig
 3358 by concatenating authenticatorData and clientDataHash, and
 3359 signing the result using the credential private key. It sets
 3360 alg to the algorithm of the credential private key, and omits
 3361 the other fields.
 3362
 3363 Verification procedure
 3364 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
 3369 defined above, and perform CBOR decoding on it to extract the
 3370 contained fields.
 3371 2. If x5c is present, this indicates that the attestation type is
 3372 not ECDAA. 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.

3625 }
 3626 alg: COSEAlgorithmIdentifier
 3627 sig: bytes,
 3628 }
 3629 The semantics of the fields are as follows:
 3630
 3631 alg
 3632 A COSEAlgorithmIdentifier containing the identifier of the
 3633 algorithm used to generate the attestation signature.
 3634
 3635 sig
 3636 A byte string containing the attestation signature.
 3637
 3638 x5c
 3639 The elements of this array contain the attestation
 3640 certificate and its certificate chain, each encoded in
 3641 X.509 format. The attestation certificate **MUST** be the
 3642 first element in the array.
 3643
 3644 ecdaaKeyId
 3645 The identifier of the ECDAA-Issuer public key. This is the
 3646 BigIntegerToB encoding of the component "c" of the
 3647 ECDAA-Issuer public key as defined section 3.3, step 3.5
 3648 in [FIDOEcdaaAlgorithm].
 3649
 3650 Signing procedure
 3651 The signing procedure for this attestation statement format is
 3652 similar to the procedure for generating assertion signatures.
 3653
 3654 1. Let authenticatorData denote the authenticator data for the
 3655 attestation, and let clientDataHash denote the hash of the
 3656 serialized client data.
 3657 2. If Basic or **AttCA** attestation is in use, the **authenticator**
 3658 **produces the sig by concatenating authenticatorData and**
 3659 **clientDataHash, and signing the result using an attestation**
 3660 **private key selected through an authenticator-specific**
 3661 **mechanism. It sets x5c to the certificate chain of the**
 3662 **attestation public key and alg to the algorithm of the**
 3663 **attestation private key.**
 3664 3. If ECDAA is in use, the authenticator produces sig by
 3665 concatenating authenticatorData and clientDataHash, and
 3666 signing the result using ECDAA-Sign (see section 3.5 of
 3667 [FIDOEcdaaAlgorithm]) after selecting an ECDAA-Issuer public
 3668 key related to the ECDAA signature private key through an
 3669 authenticator-specific mechanism (see [FIDOEcdaaAlgorithm]).
 3670 It sets alg to the algorithm of the selected ECDAA-Issuer
 3671 public key and ecdaaKeyId to the identifier of the
 3672 ECDAA-Issuer public key (see above).
 3673 4. If self attestation is in use, the authenticator produces sig
 3674 by concatenating authenticatorData and clientDataHash, and
 3675 signing the result using the credential private key. It sets
 3676 alg to the algorithm of the credential private key and omits
 3677 the other fields.
 3678
 3679 Verification procedure
 3680 Given the verification procedure inputs attStmt,
 3681 authenticatorData and clientDataHash, the verification procedure
 3682 is as follows:
 3683
 3684 1. Verify that attStmt is valid CBOR conforming to the syntax
 3685 defined above and perform CBOR decoding on it to extract the
 3686 contained fields.
 3687 2. If x5c is present, this indicates that the attestation type is
 3688 not ECDAA. In this case:
 3689 o Verify that sig is a valid signature over the
 3690 concatenation of authenticatorData and clientDataHash
 3691 using the attestation public key in x5c with the
 3692 algorithm specified in alg.
 3693 o Verify that x5c meets the requirements in 8.2.1 Packed
 3694 attestation statement certificate requirements.

3375 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 o If successful, return attestation type Basic and
 3383 attestation trust path x5c.

3384 3. If ecdaaKeyld is present, then the attestation type is ECDA.
 3385 In this case:
 3386 o Verify that sig is a valid signature over the
 3387 concatenation of authenticatorData and clientDataHash
 3388 using ECDA-Verify with ECDA-Issuer public key
 3389 identified by ecdaaKeyld (see [FIDOEcdaaAlgorithm]).
 3390 o If successful, return attestation type ECDA and
 3391 attestation trust path ecdaaKeyld.

3392 4. If neither x5c nor ecdaaKeyld is present, self attestation is
 3393 in use.
 3394 o Validate that alg matches the algorithm of the
 3395 credentialPublicKey in authenticatorData.
 3396 o Verify that sig is a valid signature over the
 3397 concatenation of authenticatorData and clientDataHash
 3398 using the credential public key with alg.
 3399 o If successful, return attestation type Self and empty
 3400 attestation trust path.

3401 8.2.1. Packed attestation statement certificate requirements

3402 The attestation certificate MUST have the following fields/extensions:
 3403 * Version **must** be set to 3.

3404 * Subject field MUST be set to:

3405 Subject-C
 3406 Country where the Authenticator vendor is incorporated

3407 Subject-O
 3408 Legal name of the Authenticator vendor

3409 Subject-OU
 3410 Authenticator Attestation

3411 Subject-CN
 3412 No stipulation.

3413 * If the related attestation root certificate is used for multiple
 3414 authenticator models, the Extension OID 1 3 6 1 4 1 45724 1 1 4
 3415 (id-fido-gen-ce-aaguid) MUST be present, containing the AAGUID as
 3416 value.

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].

8.3. TPM Attestation Statement Format

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

3695 o If x5c contains an extension with OID
 3696 1.3.6.1.4.1.45724.1.1.4 (id-fido-gen-ce-aaguid) verify
 3697 that the value of this extension matches the aaguid in
 3698 authenticatorData.

3699 o If successful, return attestation type Basic and
 3700 attestation trust path x5c.

3701 3. If ecdaaKeyld is present, then the attestation type is ECDA.
 3702 In this case:
 3703 o Verify that sig is a valid signature over the
 3704 concatenation of authenticatorData and clientDataHash
 3705 using ECDA-Verify with ECDA-Issuer public key
 3706 identified by ecdaaKeyld (see [FIDOEcdaaAlgorithm]).
 3707 o If successful, return attestation type ECDA and
 3708 attestation trust path ecdaaKeyld.

3709 4. If neither x5c nor ecdaaKeyld is present, self attestation is
 3710 in use.
 3711 o Validate that alg matches the algorithm of the
 3712 credentialPublicKey in authenticatorData.
 3713 o Verify that sig is a valid signature over the
 3714 concatenation of authenticatorData and clientDataHash
 3715 using the credential public key with alg.
 3716 o If successful, return attestation type Self and empty
 3717 attestation trust path.

3718 8.2.1. Packed attestation statement certificate requirements

3719 The attestation certificate MUST have the following fields/extensions:
 3720 * Version **MUST** be set to 3 (which is indicated by an ASN.1 INTEGER
 3721 with value 2).

3722 * Subject field MUST be set to:

3723 Subject-C
 3724 ISO 3166 code specifying the country where the
 3725 Authenticator vendor is incorporated (PrintableString)

3726 Subject-O
 3727 Legal name of the Authenticator vendor (UTF8String)

3728 Subject-OU
 3729 Literal string "Authenticator Attestation" (UTF8String)

3730 Subject-CN
 3731 A UTF8String of the vendor's choosing

3732 * If the related attestation root certificate is used for multiple
 3733 authenticator models, the Extension OID 1.3.6.1.4.1.45724.1.1.4
 3734 (id-fido-gen-ce-aaguid) MUST be present, containing the AAGUID as a
 3735 16-byte OCTET STRING. The extension MUST NOT be marked as critical.
 3736 Note that an X.509 Extension encodes the DER-encoding of the value
 3737 in an OCTET STRING. Thus, the AAGUID must be wrapped in two OCTET
 3738 STRINGS to be valid. Here is a sample, encoded Extension structure:
 3739 30 21 -- SEQUENCE
 3740 06 0b 2b 06 01 04 01 82 e5 1c 01 01 04 -- 1.3.6.1.4.1.45724.1.1.4
 3741 04 12 -- OCTET STRING
 3742 04 10 -- OCTET STRING
 3743 cd 8c 39 5c 26 ed ee de -- AAGUID
 3744 65 3b 00 79 7d 03 ca 3c

3752 * The Basic Constraints extension MUST have the CA component set to
 3753 false.

3754 * An Authority Information Access (AIA) extension with entry
 3755 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are
 3756 both **OPTIONAL** as the status of many attestation certificates is
 3757 available through authenticator metadata services. See, for
 3758 example, the FIDO Metadata Service [FIDOMetadataService].

8.3. TPM Attestation Statement Format

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

```

3436 Attestation statement format identifier
3437 tpm
3438
3439 Attestation types supported
3440 Privacy CA, ECDAA
3441
3442 Syntax
3443 The syntax of a TPM Attestation statement is as follows:
3444
3445 $$attStmtType // = (
3446     fmt: "tpm",
3447     attStmt: tpmStmtFormat
3448 )
3449
3450 tpmStmtFormat = {
3451     ver: "2.0",
3452     (
3453         alg: COSEAlgorithmIdentifier,
3454         x5c: [ aikCert: bytes, * (caCert: bytes) ]
3455     ) //
3456     1 for ED512) alg: COSEAlgorithmIdentifier, (-260 for ED256 / -26
3457     ecdaaKeyId: bytes
3458 ),
3459     sig: bytes,
3460     certInfo: bytes,
3461     pubArea: bytes
3462 }
3463
3464 The semantics of the above fields are as follows:
3465
3466 ver
3467 The version of the TPM specification to which the
3468 signature conforms.
3469
3470 alg
3471 A COSEAlgorithmIdentifier containing the identifier of the
3472 algorithm used to generate the attestation signature.
3473
3474 x5c
3475 The AIK certificate used for the attestation and its
3476 certificate chain, in X.509 encoding.
3477
3478 ecdaaKeyId
3479 The identifier of the ECDAA-Issuer public key. This is the
3480 BigNumberToB encoding of the component "c" as defined
3481 section 3.3, step 3.5 in [FIDOecdaaAlgorithm].
3482
3483 sig
3484 The attestation signature, in the form of a TPMT_SIGNATURE
3485 structure as specified in [TPMv2-Part2] section 11.3.4.
3486
3487 certInfo
3488 The TPMS_ATTEST structure over which the above signature
3489 was computed, as specified in [TPMv2-Part2] section
3490 10.12.8.
3491
3492 pubArea
3493 The TPMT_PUBLIC structure (see [TPMv2-Part2] section
3494 12.2.4) used by the TPM to represent the credential public
3495 key.
3496
3497 Signing procedure
3498 Let authenticatorData denote the authenticator data for the
3499 attestation, and let clientDataHash denote the hash of the
3500 serialized client data.
3501
3502 Concatenate authenticatorData and clientDataHash to form

```

```

3765 Attestation statement format identifier
3766 tpm
3767
3768 Attestation types supported
3769 AttCA, ECDAA
3770
3771 Syntax
3772 The syntax of a TPM Attestation statement is as follows:
3773
3774 $$attStmtType // = (
3775     fmt: "tpm",
3776     attStmt: tpmStmtFormat
3777 )
3778
3779 tpmStmtFormat = {
3780     ver: "2.0",
3781     (
3782         alg: COSEAlgorithmIdentifier,
3783         x5c: [ aikCert: bytes, * (caCert: bytes) ]
3784     ) //
3785     1 for ED512) alg: COSEAlgorithmIdentifier, (-260 for ED256 / -26
3786     ecdaaKeyId: bytes
3787 ),
3788     sig: bytes,
3789     certInfo: bytes,
3790     pubArea: bytes
3791 }
3792
3793 The semantics of the above fields are as follows:
3794
3795 ver
3796 The version of the TPM specification to which the
3797 signature conforms.
3798
3799 alg
3800 A COSEAlgorithmIdentifier containing the identifier of the
3801 algorithm used to generate the attestation signature.
3802
3803 x5c
3804 The AIK certificate used for the attestation and its
3805 certificate chain, in X.509 encoding.
3806
3807 ecdaaKeyId
3808 The identifier of the ECDAA-Issuer public key. This is the
3809 BigNumberToB encoding of the component "c" as defined
3810 section 3.3, step 3.5 in [FIDOecdaaAlgorithm].
3811
3812 sig
3813 The attestation signature, in the form of a TPMT_SIGNATURE
3814 structure as specified in [TPMv2-Part2] section 11.3.4.
3815
3816 certInfo
3817 The TPMS_ATTEST structure over which the above signature
3818 was computed, as specified in [TPMv2-Part2] section
3819 10.12.8.
3820
3821 pubArea
3822 The TPMT_PUBLIC structure (see [TPMv2-Part2] section
3823 12.2.4) used by the TPM to represent the credential public
3824 key.
3825
3826 Signing procedure
3827 Let authenticatorData denote the authenticator data for the
3828 attestation, and let clientDataHash denote the hash of the
3829 serialized client data.
3830
3831 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 If x5c is present, this indicates that the attestation type is
3547 not ECDA. In this case:
3548
3549 + Verify the sig is a valid signature over certInfo using the
3550 attestation public key in x5c with the algorithm specified in
3551 alg.
3552 + Verify that x5c meets the requirements in 8.3.1 TPM
3553 attestation statement certificate requirements.
3554 + If x5c contains an extension with OID 1.3.6.1.4.1.45724.1.1.4
3555 (id-fido-gen-ce-aaguid) verify that the value of this
3556 extension matches the aaguid in authenticatorData.
3557 + If successful, return attestation type Privacy CA and
3558 attestation trust path x5c.
3559
3560 If ecdAaKeyID is present, then the attestation type is ECDA.
3561
3562 + Perform ECDA-Verify on sig to verify that it is a valid
3563 signature over certInfo (see [FIDOecdaaAlgorithm]).
3564 + If successful, return attestation type ECDA and the
3565 identifier of the ECDA-Issuer public key ecdAaKeyID.

3566 8.3.1. TPM attestation statement certificate requirements
3567
3568 TPM attestation certificate MUST have the following fields/extensions:
3569
3570

3835 attToBeSigned.
3836
3837 Generate a signature using the procedure specified in
3838 [TPMv2-Part3] Section 18.2, using the attestation private key
3839 and setting the extraData parameter to the digest of
3840 attToBeSigned using the hash algorithm corresponding to the
3841 "alg" signature algorithm. (For the "RS256" algorithm, this
3842 would be a SHA-256 digest.)
3843
3844 Set the pubArea field to the public area of the credential
3845 public key, the certInfo field to the output parameter of the
3846 same name, and the sig field to the signature obtained from the
3847 above procedure.
3848
3849 Verification procedure
3850 Given the verification procedure inputs attStmt,
3851 authenticatorData and clientDataHash, the verification procedure
3852 is as follows:
3853
3854 Verify that attStmt is valid CBOR conforming to the syntax
3855 defined above and perform CBOR decoding on it to extract the
3856 contained fields.
3857
3858 Verify that the public key specified by the parameters and
3859 unique fields of pubArea is identical to the credentialPublicKey
3860 in the attestedCredentialData in authenticatorData.
3861
3862 Concatenate authenticatorData and clientDataHash to form
3863 attToBeSigned.
3864
3865 Validate that certInfo is valid:
3866
3867 + Verify that magic is set to TPM_GENERATED_VALUE.
3868 + Verify that type is set to TPM_ST_ATTEST_CERTIFY.
3869 + Verify that extraData is set to the hash of attToBeSigned
3870 using the hash algorithm employed in "alg".
3871 + Verify that attested contains a TPMS_CERTIFY_INFO structure as
3872 specified in [TPMv2-Part2] section 10.12.3, whose name field
3873 contains a valid Name for pubArea, as computed using the
3874 algorithm in the nameAlg field of pubArea using the procedure
3875 specified in [TPMv2-Part1] section 16.
3876 + Note that the remaining fields in the "Standard Attestation
3877 Structure" [TPMv2-Part1] section 31.2, i.e., qualifiedSigner,
3878 clockInfo and firmwareVersion are ignored. These fields MAY be
3879 used as an input to risk engines.

3880 If x5c is present, this indicates that the attestation type is
3881 not ECDA. In this case:
3882
3883 + Verify the sig is a valid signature over certInfo using the
3884 attestation public key in x5c with the algorithm specified in
3885 alg.
3886 + Verify that x5c meets the requirements in 8.3.1 TPM
3887 attestation statement certificate requirements.
3888 + If x5c contains an extension with OID 1.3.6.1.4.1.45724.1.1.4
3889 (id-fido-gen-ce-aaguid) verify that the value of this
3890 extension matches the aaguid in authenticatorData.
3891 + If successful, return attestation type AttCA and attestation
3892 trust path x5c.
3893
3894 If ecdAaKeyID is present, then the attestation type is ECDA.
3895
3896 + Perform ECDA-Verify on sig to verify that it is a valid
3897 signature over certInfo (see [FIDOecdaaAlgorithm]).
3898 + If successful, return attestation type ECDA and the
3899 identifier of the ECDA-Issuer public key ecdAaKeyID.

3900 8.3.1. TPM attestation statement certificate requirements
3901
3902 TPM attestation certificate MUST have the following fields/extensions:
3903
3904

- 3571 * Version **must** be set to 3.
- 3572 * Subject field **MUST** be set to empty.
- 3573 * The Subject Alternative Name extension **must** be set as defined in
- 3574 [TPMv2-EK-Profile] section 3.2.9.
- 3575 * The Extended Key Usage extension **MUST** contain the
- 3576 "joint-iso-itu-t(2) internationalorganizations(23) 133 tcg-kp(8)
- 3577 tcg-kp-AIKCertificate(3)" OID.
- 3578 * The Basic Constraints extension **MUST** have the CA component set to
- 3579 false.
- 3580 * An Authority Information Access (AIA) extension with entry
- 3581 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are
- 3582 both **optional** as the status of many attestation certificates is
- 3583 available through metadata services. See, for example, the FIDO
- 3584 Metadata Service [FIDOMetadataService].
- 3585

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 serialized client data.

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

The authenticator produces sig by concatenating authenticatorData and clientDataHash, and signing the result using the credential private key. It sets alg to the algorithm of the signature format.

Verification procedure

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

- 3905 * Version **MUST** be set to 3.
- 3906 * Subject field **MUST** be set to empty.
- 3907 * The Subject Alternative Name extension **MUST** be set as defined in
- 3908 [TPMv2-EK-Profile] section 3.2.9.
- 3909 * The Extended Key Usage extension **MUST** contain the
- 3910 "joint-iso-itu-t(2) internationalorganizations(23) 133 tcg-kp(8)
- 3911 tcg-kp-AIKCertificate(3)" OID.
- 3912 * The Basic Constraints extension **MUST** have the CA component set to
- 3913 false.
- 3914 * An Authority Information Access (AIA) extension with entry
- 3915 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are
- 3916 both **OPTIONAL** as the status of many attestation certificates is
- 3917 available through metadata services. See, for example, the FIDO
- 3918 Metadata Service [FIDOMetadataService].
- 3919

8.4. Android Key Attestation Statement Format

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

Attestation statement format identifier
android-key

Attestation types supported
Basic

Syntax

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

```

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

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

Signing procedure

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

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

The authenticator produces sig by concatenating authenticatorData and clientDataHash, and signing the result using the credential private key. It sets alg to the algorithm of the signature format.

Verification procedure

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

3641 + Verify that attStmt is valid CBOR conforming to the syntax
 3642 defined above, and perform CBOR decoding on it to extract the
 3643 contained fields.
 3644 + Verify that the public key in the first certificate in the
 3645 series of certificates represented by the signature matches
 3646 the credentialPublicKey in the attestedCredentialData in
 3647 authenticatorData.
 3648

3649 + Verify that in the attestation certificate extension data:
 3650 o The value of the attestationChallenge field is identical
 3651 to the concatenation of authenticatorData and
 3652 clientDataHash.
 3653 o The AuthorizationList.allApplications field is not
 3654 present, since PublicKeyCredentials must be bound to the
 3655 RP ID.
 3656 o The value in the AuthorizationList.origin field is equal
 3657 to KM_TAG_GENERATED.
 3658 o The value in the AuthorizationList.purpose field is equal
 3659 to KM_PURPOSE_SIGN.
 3660 + If successful, return attestation type Basic with the
 3661 attestation trust path set to the entire attestation
 3662 statement.

8.5. Android SafetyNet Attestation Statement Format

When the authenticator in question is a platform-provided Authenticator on certain Android platforms, the attestation statement is based on the SafetyNet API. In this case the authenticator data is completely controlled by the caller of the SafetyNet API (typically an application running on the Android platform) and the attestation statement only provides some statements about the health of the platform and the identity of the calling application. This attestation does not provide information regarding provenance of the authenticator and its associated data. Therefore platform-provided authenticators should make use of the Android Key Attestation when available, even if the SafetyNet API is also present.

Attestation statement format identifier
android-safetynet

Attestation types supported
Basic Attestation

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

```

    $$attStmtType ::= (
        fmt: "android-safetynet",
        attStmt: safetynetStmtFormat
    )
    safetynetStmtFormat = {
        ver: text,
        response: bytes
    }
    
```

The semantics of the above fields are as follows:

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

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

3975 + Verify that attStmt is valid CBOR conforming to the syntax
 3976 defined above and perform CBOR decoding on it to extract the
 3977 contained fields.
 3978 + Verify that sig is a valid signature over the concatenation of
 3979 authenticatorData and clientDataHash using the public key in
 3980 the first certificate in x5c with the algorithm specified in
 3981 alg.
 3982 + Verify that the public key in the first certificate in in x5c
 3983 matches the credentialPublicKey in the attestedCredentialData
 3984 in authenticatorData.
 3985 + Verify that in the attestation certificate extension data:
 3986 o The value of the attestationChallenge field is identical
 3987 to clientDataHash.
 3988

3989 o The AuthorizationList.allApplications field is not
 3990 present, since PublicKeyCredential must be bound to the
 3991 RP ID.
 3992 o The value in the AuthorizationList.origin field is equal
 3993 to KM_TAG_GENERATED.
 3994 o The value in the AuthorizationList.purpose field is equal
 3995 to KM_PURPOSE_SIGN.
 3996 + If successful, return attestation type Basic with the
 3997 attestation trust path set to x5c.

8.5. Android SafetyNet Attestation Statement Format

When the authenticator in question is a platform-provided Authenticator on certain Android platforms, the attestation statement is based on the SafetyNet API. In this case the authenticator data is completely controlled by the caller of the SafetyNet API (typically an application running on the Android platform) and the attestation statement only provides some statements about the health of the platform and the identity of the calling application. This attestation does not provide information regarding provenance of the authenticator and its associated data. Therefore platform-provided authenticators should make use of the Android Key Attestation when available, even if the SafetyNet API is also present.

Attestation statement format identifier
android-safetynet

Attestation types supported
Basic

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

```

    $$attStmtType ::= (
        fmt: "android-safetynet",
        attStmt: safetynetStmtFormat
    )
    safetynetStmtFormat = {
        ver: text,
        response: bytes
    }
    
```

The semantics of the above fields are as follows:

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

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

370E Signing procedure
 370F 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 u2fStmtFormat = {
 3762 x5c: [attestnCert: bytes, * (caCert: bytes)],
 3763 sig: bytes
 3764 }
 3765
 3766 The semantics of the above fields are as follows:
 3767
 3768 x5c
 3769 The elements of this array contain the attestation
 3770 certificate and its certificate chain, each encoded in
 3771 X.509 format. The attestation certificate **must** be the
 3772 first element in the array.
 3773
 3774 sig
 3775 The attestation signature. The signature was calculated
 3776 over the (raw) U2F registration response message
 3777

4043 Signing procedure
 4044 Let authenticatorData denote the authenticator data for the
 4045 attestation, and let clientDataHash denote the hash of the
 4046 serialized client data.
 4047
 4048 Concatenate authenticatorData and clientDataHash to form
 4049 attToBeSigned.
 4050
 4051 Request a SafetyNet attestation, providing attToBeSigned as the
 4052 nonce value. Set response to the result, and ver to the version
 4053 of Google Play Services running in the authenticator.
 4054
 4055 Verification procedure
 4056 Given the verification procedure inputs attStmt,
 4057 authenticatorData and clientDataHash, the verification procedure
 4058 is as follows:
 4059
 4060 + Verify that attStmt is valid CBOR conforming to the syntax
 4061 defined above and perform CBOR decoding on it to extract the
 4062 contained fields.
 4063 + Verify that response is a valid SafetyNet response of version
 4064 ver.
 4065 + Verify that the nonce in the response is identical to the
 4066 concatenation of authenticatorData and clientDataHash.
 4067 + Verify that the attestation certificate is issued to the
 4068 hostname "attest.android.com" (see SafetyNet online
 4069 documentation).
 4070 + Verify that the ctsProfileMatch attribute in the payload of
 4071 response is true.
 4072 + If successful, return attestation type Basic with the
 4073 attestation trust path set to the above attestation
 4074 certificate.
 4075
 4076 8.6. FIDO U2F Attestation Statement Format
 4077
 4078 This attestation statement format is used with FIDO U2F authenticators
 4079 using the formats defined in [FIDO-U2F-Message-Formats].
 4080
 4081 Attestation statement format identifier
 4082 fido-u2f
 4083
 4084 Attestation types supported
 4085 Basic, AttCA
 4086
 4087 Syntax
 4088 The syntax of a FIDO U2F attestation statement is defined as
 4089 follows:
 4090
 4091 \$\$attStmtType ::= (
 4092 fmt: "fido-u2f",
 4093 attStmt: u2fStmtFormat
 4094)
 4095
 4096 u2fStmtFormat = {
 4097 x5c: [attestnCert: bytes, * (caCert: bytes)],
 4098 sig: bytes
 4099 }
 4100
 4101 The semantics of the above fields are as follows:
 4102
 4103 x5c
 4104 The elements of this array contain the attestation
 4105 certificate and its certificate chain, each encoded in
 4106 X.509 format. The attestation certificate **MUST** be the
 4107 first element in the array.
 4108
 4109 sig
 4110 The attestation signature. The signature was calculated
 4111 over the (raw) U2F registration response message
 4112

377E [FIDO-U2F-Message-Formats] received by the platform from
 377F 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 attStmt,
 3804 authenticatorData and clientDataHash, the verification procedure
 3805 is as follows:
 3806
 3807 1. Verify that attStmt is valid CBOR conforming to the syntax
 3808 defined above, and perform CBOR decoding on it to extract the
 3809 contained fields.
 3810 2. Let attCert be value of the first element of x5c. Let
 3811 certificate public key be the public key conveyed by attCert.
 3812 If certificate public key is not an Elliptic Curve (EC) public
 3813 key over the P-256 curve, terminate this algorithm and return
 3814 an appropriate error.
 3815 3. Extract the claimed rpIdHash from authenticatorData, and the
 3816 claimed credentialId and credentialPublicKey from
 3817 authenticatorData.attestedCredentialData.
 3818 4. If clientDataHash is 256 bits long, set tbsHash to this value.
 3819 Otherwise set tbsHash to the SHA-256 hash of clientDataHash.
 3820 5. Convert the COSE_KEY formatted credentialPublicKey (see
 3821 Section 7 of [RFC8152]) to CTAP1/U2F public Key format
 3822 [FIDO-CTAP].
 3823 o Let publicKeyU2F represent the result of the conversion
 3824 operation and set its first byte to 0x04. Note: This
 3825 signifies uncompressed ECC key format.
 3826 o Extract the value corresponding to the "-2" key
 3827 (representing x coordinate) from credentialPublicKey,
 3828 confirm its size to be of 32 bytes and concatenate it
 3829 with publicKeyU2F. If size differs or "-2" key is not
 3830 found, terminate this algorithm and return an appropriate
 3831 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

4113 [FIDO-U2F-Message-Formats] received by the platform from
 4114 the authenticator.
 4115
 4116 Signing procedure
 4117 If the credential public key of the given credential is not of
 4118 algorithm -7 ("ES256"), stop and return an error. Otherwise, let
 4119 authenticatorData denote the authenticator data for the
 4120 attestation, and let clientDataHash denote the hash of the
 4121 serialized client data. (Since SHA-256 is used to hash the
 4122 serialized client data, clientDataHash will be 32 bytes long.)
 4123
 4124 Generate a Registration Response Message as specified in
 4125 [FIDO-U2F-Message-Formats] section 4.3, with the application
 4126 parameter set to the SHA-256 hash of the RP ID associated with
 4127 the given credential, the challenge parameter set to
 4128 clientDataHash, and the key handle parameter set to the
 4129 credential ID of the given credential. Set the raw signature
 4130 part of this Registration Response Message (i.e., without the
 4131 user public key, key handle, and attestation certificates) as
 4132 sig and set the attestation certificates of the attestation
 4133 public key as x5c.
 4134
 4135 Verification procedure
 4136 Given the verification procedure inputs attStmt,
 4137 authenticatorData and clientDataHash, the verification procedure
 4138 is as follows:
 4139
 4140 1. Verify that attStmt is valid CBOR conforming to the syntax
 4141 defined above and perform CBOR decoding on it to extract the
 4142 contained fields.
 4143 2. Let attCert be the value of the first element of x5c. Let
 4144 certificate public key be the public key conveyed by attCert.
 4145 If certificate public key is not an Elliptic Curve (EC) public
 4146 key over the P-256 curve, terminate this algorithm and return
 4147 an appropriate error.
 4148 3. Extract the claimed rpIdHash from authenticatorData, and the
 4149 claimed credentialId and credentialPublicKey from
 4150 authenticatorData.attestedCredentialData.
 4151 4. Convert the COSE_KEY formatted credentialPublicKey (see
 4152 Section 7 of [RFC8152]) to CTAP1/U2F public Key format
 4153 [FIDO-CTAP].
 4154 o Let publicKeyU2F represent the result of the conversion
 4155 operation and set its first byte to 0x04. Note: This
 4156 signifies uncompressed ECC key format.
 4157 o Extract the value corresponding to the "-2" key
 4158 (representing x coordinate) from credentialPublicKey,
 4159 confirm its size to be of 32 bytes and concatenate it
 4160 with publicKeyU2F. If size differs or "-2" key is not
 4161 found, terminate this algorithm and return an appropriate
 4162 error.
 4163 o Extract the value corresponding to the "-3" key
 4164 (representing y coordinate) from credentialPublicKey,
 4165 confirm its size to be of 32 bytes and concatenate it
 4166 with publicKeyU2F. If size differs or "-3" key is not
 4167 found, terminate this algorithm and return an appropriate
 4168 error.
 4169 5. Let verificationData be the concatenation of (0x00 || rpIdHash
 4170 || clientDataHash || credentialId || publicKeyU2F) (see
 4171 Section 4.3 of [FIDO-U2F-Message-Formats]).
 4172 6. Verify the sig using verificationData and certificate public
 4173 key per [SEC1].
 4174 7. If successful, return attestation type Basic with the
 4175 attestation trust path set to x5c.
 4176
 4177 8.7. None Attestation Statement Format
 4178

3846 9. WebAuthn Extensions
3847
3848 The mechanism for generating public key credentials, as well as
3849 requesting and generating Authentication assertions, as defined in 5
3850 Web Authentication API, can be extended to suit particular use cases.
3851 Each case is addressed by defining a registration extension and/or an
3852 authentication extension.
3853
3854 Every extension is a client extension, meaning that the extension
3855 involves communication with and processing by the client. Client
3856 extensions define the following steps and data:
3857 * navigator.credentials.create() extension request parameters and
3858 response values for registration extensions.
3859 * navigator.credentials.get() extension request parameters and
3860 response values for authentication extensions.
3861 * Client extension processing for registration extensions and
3862 authentication extensions.
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
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

4179 The none attestation statement format is used to replace any
4180 authenticator-provided attestation statement when a Relying Party
4181 indicates it does not wish to receive attestation information, see
4182 5.4.6 Attestation Conveyance Preference enumeration (enum
4183 AttestationConveyancePreference).
4184
4185 Attestation statement format identifier
4186 none
4187
4188 Attestation types supported
4189 None
4190
4191 Syntax
4192 The syntax of a none attestation statement is defined as
4193 follows:
4194
4195 `$$attStmtType ::= (
4196 fmt: "none",
4197 attStmt: emptyMap
4198)`
4199
4200 `emptyMap = {}`
4201
4202 Signing procedure
4203 Return the fixed attestation statement defined above.
4204
4205 Verification procedure
4206 Return attestation type None with an empty trust path.
4207
4208 9. WebAuthn Extensions
4209
4210 The mechanism for generating public key credentials, as well as
4211 requesting and generating Authentication assertions, as defined in 5
4212 Web Authentication API, can be extended to suit particular use cases.
4213 Each case is addressed by defining a registration extension and/or an
4214 authentication extension.
4215
4216 Every extension is a client extension, meaning that the extension
4217 involves communication with and processing by the client. Client
4218 extensions define the following steps and data:
4219 * navigator.credentials.create() extension request parameters and
4220 response values for registration extensions.
4221 * navigator.credentials.get() extension request parameters and
4222 response values for authentication extensions.
4223 * Client extension processing for registration extensions and
4224 authentication extensions.
4225
4226 When creating a public key credential or requesting an authentication
4227 assertion, a Relying Party can request the use of a set of extensions.
4228 These extensions will be invoked during the requested operation if they
4229 are supported by the client and/or the authenticator. The Relying Party
4230 sends the client extension input for each extension in the get() call
4231 (for authentication extensions) or create() call (for registration
4232 extensions) to the client platform. The client platform performs client
4233 extension processing for each extension that it supports, and augments
4234 the client data as specified by each extension, by including the
4235 extension identifier and client extension output values.
4236
4237 An extension can also be an authenticator extension, meaning that the
4238 extension involves communication with and processing by the
4239 authenticator. Authenticator extensions define the following steps and
4240 data:
4241 * authenticatorMakeCredential extension request parameters and
4242 response values for registration extensions.
4243 * authenticatorGetAssertion extension request parameters and response
4244 values for authentication extensions.
4245 * Authenticator extension processing for registration extensions and
4246 authentication extensions.
4247
4248 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

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

4249 processing, the client also creates the CBOR authenticator extension
 4250 input value for each extension (often based on the corresponding client
 4251 extension input value), and passes them to the authenticator in the
 4252 create() call (for registration extensions) or the get() call (for
 4253 authentication extensions). These authenticator extension input values
 4254 are represented in CBOR and passed as name-value pairs, with the
 4255 extension identifier as the name, and the corresponding authenticator
 4256 extension input as the value. The authenticator, in turn, performs
 4257 additional processing for the extensions that it supports, and returns
 4258 the CBOR authenticator extension output for each as specified by the
 4259 extension. Part of the client extension processing for authenticator
 4260 extensions is to use the authenticator extension output as an input to
 4261 creating the client extension output.
 4262

4263 All WebAuthn extensions are **OPTIONAL** for both clients and
 4264 authenticators. Thus, any extensions requested by a Relying Party **MAY**
 4265 be ignored by the client browser or OS and not passed to the
 4266 authenticator at all, or they **MAY** be ignored by the authenticator.
 4267 Ignoring an extension is never considered a failure in WebAuthn API
 4268 processing, so when Relying Parties include extensions with any API
 4269 calls, they **MUST** be prepared to handle cases where some or all of those
 4270 extensions are ignored.
 4271

4272 Clients wishing to support the widest possible range of extensions **MAY**
 4273 choose to pass through any extensions that they do not recognize to
 4274 authenticators, generating the authenticator extension input by simply
 4275 encoding the client extension input in CBOR. All WebAuthn extensions
 4276 MUST be defined in such a way that this implementation choice does not
 4277 endanger the user's security or privacy. For instance, if an extension
 4278 requires client processing, it could be defined in a manner that
 4279 ensures such a naive pass-through will produce a semantically invalid
 4280 authenticator extension input value, resulting in the extension being
 4281 ignored by the authenticator. Since all extensions are **OPTIONAL**, this
 4282 will not cause a functional failure in the API operation. Likewise,
 4283 clients can choose to produce a client extension output value for an
 4284 extension that it does not understand by encoding the authenticator
 4285 extension output value into JSON, provided that the CBOR output uses
 4286 only types present in JSON.
 4287

4288 When clients choose to pass through extensions they do not recognize,
 4289 the JavaScript values in the client extension inputs are converted to
 4290 CBOR values in the authenticator extension inputs. When the JavaScript
 4291 value is an %ArrayBuffer%, it is converted to a CBOR byte array. When
 4292 the JavaScript value is a non-integer number, it is converted to a
 4293 64-bit CBOR floating point number. Otherwise, when the JavaScript type
 4294 corresponds to a JSON type, the conversion is done using the rules
 4295 defined in Section 4.2 of [RFC7049] (Converting from JSON to CBOR), but
 4296 operating on inputs of JavaScript type values rather than inputs of
 4297 JSON type values. Once these conversions are done, canonicalization of
 4298 the resulting CBOR MUST be performed using the CTAP2 canonical CBOR
 4299 encoding form.
 4300

4301 Likewise, when clients receive outputs from extensions they have passed
 4302 through that they do not recognize, the CBOR values in the
 4303 authenticator extension outputs are converted to JavaScript values in
 4304 the client extension outputs. When the CBOR value is a byte string, it
 4305 is converted to a JavaScript %ArrayBuffer% (rather than a
 4306 base64url-encoded string). Otherwise, when the CBOR type corresponds to
 4307 a JSON type, the conversion is done using the rules defined in Section
 4308 4.1 of [RFC7049] (Converting from CBOR to JSON), but producing outputs
 4309 of JavaScript type values rather than outputs of JSON type values.
 4310

4311 Note that some clients may choose to implement this pass-through
 4312 capability under a feature flag. Supporting this capability can
 4313 facilitate innovation, allowing authenticators to experiment with new
 4314 extensions and Relying Parties to use them before there is explicit
 4315 support for them in clients.
 4316

4317 The IANA "WebAuthn Extension Identifier" registry established by
 4318 [WebAuthn-Registries] **can** be consulted for an up-to-date list of

392E registered WebAuthn Extensions.
392F
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3990
3991
3992
3993
3994
3995
3996
3997

9.1. Extension Identifiers

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

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

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

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

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

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

9.2. Defining extensions

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

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

9.3. Extending request parameters

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

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

```
var assertionPromise = navigator.credentials.get({
```

431F registered WebAuthn Extensions.
4320
4321
4322
4323
4324
4325
4326
4327
4328
4329
4330
4331
4332
4333
4334
4335
4336
4337
4338
4339
4340
4341
4342
4343
4344
4345
4346
4347
4348
4349
4350
4351
4352
4353
4354
4355
4356
4357
4358
4359
4360
4361
4362
4363
4364
4365
4366
4367
4368
4369
4370
4371
4372
4373
4374
4375
4376
4377
4378
4379
4380
4381
4382
4383
4384
4385
4386
4387
4388

9.1. Extension Identifiers

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

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

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

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

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

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

9.2. Defining extensions

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

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

9.3. Extending request parameters

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

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

```
var assertionPromise = navigator.credentials.get({
```

```

399E     publicKey: {
399F         // The challenge must be produced by the server, see the Security Consid
4000     erations
4001         challenge: new Uint8Array([4,99,22 /* 29 more random bytes generated by
4002     the server */]),
4003         extensions: {
4004             "webauthnExample_foobar": 42
4005         }
4006     };
4007
4008
4009
4010
4011
4012
4013
4014
4015
4016
4017
4018
4019
4020
4021
4022
4023
4024
4025
4026
4027
4028
4029
4030
4031
4032
4033
4034
4035
4036
4037
4038
4039
4040
4041
4042
4043
4044
4045
4046
4047
4048
4049
4050
4051
4052
4053
4054
4055
4056
4057
4058
4059
4060
4061
4062
4063
4064
4065
4066
4067

```

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 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.

9.5. Authenticator extension processing

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

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

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

```

438E     publicKey: {
438F         // The challenge must be produced by the server, see the Security Consid
4390     erations
4391         challenge: new Uint8Array([4,99,22 /* 29 more random bytes generated by
4392     the server */]),
4393         extensions: {
4394             "webauthnExample_foobar": 42
4395         }
4396     };
4397
4398
4399
4400
4401
4402
4403
4404
4405
4406
4407
4408
4409
4410
4411
4412
4413
4414
4415
4416
4417
4418
4419
4420
4421
4422
4423
4424
4425
4426
4427
4428
4429
4430
4431
4432
4433
4434
4435
4436
4437
4438
4439
4440
4441
4442
4443
4444
4445
4446
4447
4448
4449
4450

```

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 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.

9.5. Authenticator extension processing

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

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

4130 **10. Defined Extensions**

4131

4132

4133 This section defines the initial set of extensions to be registered in

4134 the IANA "WebAuthn Extension Identifier" registry established by

4135 [WebAuthn-Registries]. These are recommended for implementation by user

4136 agents targeting broad interoperability.

4137

4451

4452 Likewise, the extension output is represented in the extensions part of

4453 the authenticator data. The extensions part of the authenticator data

4454 is a CBOR map where each key is an extension identifier and the

4455 corresponding value is the authenticator extension output for that

4456 extension.

4457

4458 For each supported extension, the authenticator extension processing

4459 rule for that extension is used create the authenticator extension

4460 output from the authenticator extension input and possibly also other

4461 inputs.

4462 **10. Defined Extensions**

4463

4464

4465 This section defines the initial set of extensions to be registered in

4466 the IANA "WebAuthn Extension Identifier" registry established by

4467 [WebAuthn-Registries]. These are RECOMMENDED for implementation by user

4468 agents targeting broad interoperability.

4469

413E 10.1. FIDO AppID Extension (appid)
413F
4140 This authentication extension allows Relying Parties that have
4141 previously registered a credential using the legacy FIDO JavaScript
4142 APIs to request an assertion. Specifically, this extension allows
4143 Relying Parties to specify an appid [FIDO-APPID] to overwrite the
4144 otherwise computed rpId. This extension is only valid if used during
4145 the get() call; other usage will result in client error.

4146
4147 Extension identifier
4148 appid

4149
4150 Client extension input
4151 A single JSON string specifying a FIDO appid.

4152
4153 Client extension processing
4154 If rpId is present, return a DOMException whose name is
4155 "NotAllowedError", and terminate this algorithm (5.1.4.1
4156 PublicKeyCredential's [[DiscoverFromExternalSource]](origin,
4157 options, sameOriginWithAncestors) method).
4158
4159 Otherwise, replace the calculation of rpId in Step 6 of 5.1.4.1
4160 PublicKeyCredential's [[DiscoverFromExternalSource]](origin,
4161 options, sameOriginWithAncestors) method with the following
4162 procedure: The client uses the value of appid to perform the
4163 Appid validation procedure (as defined by [FIDO-APPID]). If
4164 valid, the value of rpId for all client processing should be
4165 replaced by the value of appid.

4166
4167 Client extension output
4168 Returns the JSON value true to indicate to the RP that the
4169 extension was acted upon

4170
4171 Authenticator extension input
4172 None.

4173
4174 Authenticator extension processing
4175 None.

4176
4177 Authenticator extension output
4178 None.

4179
4180 10.2. Simple Transaction Authorization Extension (txAuthSimple)
4181
4182 This registration extension and authentication extension allows for a

4470 10.1. FIDO AppID Extension (appid)
4471
4472 This client extension allows Relying Parties that have previously
4473 registered a credential using the legacy FIDO JavaScript APIs to
4474 request an assertion. The FIDO APIs use an alternative identifier for
4475 relying parties called an AppID [FIDO-APPID], and any credentials
4476 created using those APIs will be bound to that identifier. Without this
4477 extension, they would need to be re-registered in order to be bound to
4478 an RP ID.

4479
4480 This extension does not allow FIDO-compatible credentials to be
4481 created. Thus, credentials created with WebAuthn are not backwards
4482 compatible with the FIDO JavaScript APIs.

4483
4484 Extension identifier
4485 appid

4486
4487 Client extension input
4488 A single USVString specifying a FIDO AppID.

4489
4490 partial dictionary AuthenticationExtensionsClientInputs {
4491 USVString appid;
4492 };

4493
4494 Client extension processing

4495
4496 1. If present in a create() call, return a "NotSupportedError"
4497 DOMException--this extension is only valid when requesting an
4498 assertion.
4499 2. Let facetId be the result of passing the caller's origin to
4500 the FIDO algorithm for determining the FacetID of a calling
4501 application.
4502 3. Let appid be the extension input.
4503 4. Pass facetId and appid to the FIDO algorithm for determining
4504 if a caller's FacetID is authorized for an AppID. If that
4505 algorithm rejects appid then return a "SecurityError"
4506 DOMException.
4507 5. When building allowCredentialDescriptorList, if a U2F
4508 authenticator indicates that a credential is inapplicable
4509 (i.e. by returning SW_WRONG_DATA) then the client MUST retry
4510 with the U2F application parameter set to the SHA-256 hash of
4511 appid. If this results in an applicable credential, the client
4512 MUST include the credential in allowCredentialDescriptorList.
4513 The value of appid then replaces the rpId parameter of
4514 authenticatorGetAssertion.

4515
4516 Client extension output
4517 Returns the value true to indicate to the RP that the extension
4518 was acted upon.

4519
4520 partial dictionary AuthenticationExtensionsClientOutputs {
4521 boolean appid;
4522 };

4523
4524 Authenticator extension input
4525 None.

4526
4527 Authenticator extension processing
4528 None.

4529
4530 Authenticator extension output
4531 None.

4532
4533 10.2. Simple Transaction Authorization Extension (txAuthSimple)
4534
4535 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
 4194 Client extension processing
 4195 None, except creating the authenticator extension input from the
 4196 client extension input.
 4197
 4198 Client extension output
 4199 Returns the authenticator extension output string UTF-8 decoded
 4200 into a **JSON** string
 4201
 4202
 4203
 4204
 4205
 4206 Authenticator extension input
 4207 The client extension input encoded as a CBOR text string (major
 4208 type 3).
 4209
 4210
 4211
 4212 Authenticator extension processing
 4213 The authenticator **MUST** display the prompt to the user before
 performing either user verification or test of user presence.
 4214 The authenticator **may** insert line breaks if needed.
 4215
 4216 Authenticator extension output
 4217 A single CBOR string, representing the prompt as displayed
 4218 (including any eventual line breaks).
 4219
 4220
 4221
 4222
 4223
 4224
 4225
 4226
 4227
 4228
 4229
 4230
 4231
 4232
 4233
 4234

10.3. Generic Transaction Authorization Extension (txAuthGeneric)

This registration extension and authentication extension allows images to be used as transaction authorization prompts as well. This allows authenticators without a font rendering engine to be used and also supports a richer visual appearance.

Extension identifier
txAuthGeneric

Client extension input
A **CBOR** map defined as follows:

```
txAuthGenericArg = {
  contentType: text, ; MIME-Type of the content, e.g.
  "image/png"
  content: bytes
}
```

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

4536 simple form of transaction authorization. A Relying Party can specify a
 4537 prompt string, intended for display on a trusted device on the
 4538 authenticator.
 4539
 4540 Extension identifier
 4541 txAuthSimple
 4542
 4543 Client extension input
 4544 A single **USVString** prompt.
 4545
 4546
 4547
 4548
 4549
 4550
 4551 Client extension processing
 4552 None, except creating the authenticator extension input from the
 4553 client extension input.
 4554
 4555 Client extension output
 4556 Returns the authenticator extension output string UTF-8 decoded
 4557 into a **USVString**.
 4558
 4559
 4560
 4561
 4562
 4563
 4564
 4565
 4566
 4567
 4568
 4569
 4570
 4571
 4572
 4573
 4574
 4575
 4576
 4577
 4578
 4579
 4580
 4581
 4582
 4583
 4584
 4585
 4586
 4587
 4588
 4589
 4590
 4591
 4592
 4593
 4594
 4595
 4596
 4597
 4598
 4599
 4600
 4601
 4602
 4603
 4604
 4605

10.3. Generic Transaction Authorization Extension (txAuthGeneric)

This registration extension and authentication extension allows images to be used as transaction authorization prompts as well. This allows authenticators without a font rendering engine to be used and also supports a richer visual appearance.

Extension identifier
txAuthGeneric

Client extension input
A **JavaScript** object defined as follows:

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

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

4235 client extension input.
 4236
 4237 Client extension output
 4238 Returns the **base64url encoding of the authenticator extension**
 4239 **output value as a JSON string**

4240
 4241 Authenticator extension input
 4242 The client extension input encoded as a CBOR map.
 4243
 4244 Authenticator extension processing
 4245 The authenticator **MUST** display the content to the user before
 4246 performing either user verification or test of user presence.
 4247 The authenticator **may** add other information below the content.
 4248 No changes are allowed to the content itself, i.e., inside
 4249 content boundary box.
 4250
 4251 Authenticator extension output
 4252 The hash value of the content which was displayed. The
 4253 authenticator **MUST** use the same hash algorithm as it uses for
 4254 the signature itself.
 4255
 4256 10.4. Authenticator Selection Extension (authnSel)
 4257
 4258 This registration extension allows a Relying Party to guide the
 4259 selection of the authenticator that will be leveraged when creating the
 4260 credential. It is intended primarily for Relying Parties that wish to
 4261 tightly control the experience around credential creation.
 4262
 4263 Extension identifier
 4264 authnSel
 4265
 4266 Client extension input
 4267 A sequence of AAGUIDs:
 4268
 4269 typedef sequence<AAGUID> AuthenticatorSelectionList;
 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
 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
 4289 Client extension output
 4290 Returns the **JSON** value true to indicate to the RP that the
 4291 **extension was acted upon**

4292

4606 client extension input.
 4607
 4608 Client extension output
 4609 Returns the **authenticator extension output value as an**
 4610 **ArrayBuffer**.
 4611
 4612 partial dictionary AuthenticationExtensionsClientOutputs {
 4613 ArrayBuffer txAuthGeneric;
 4614 };
 4615
 4616 Authenticator extension input
 4617 The client extension input encoded as a CBOR map.
 4618
 4619 Authenticator extension processing
 4620 The authenticator **MUST** display the content to the user before
 4621 performing either user verification or test of user presence.
 4622 The authenticator **MAY** add other information below the content.
 4623 No changes are allowed to the content itself, i.e., inside
 4624 content boundary box.
 4625
 4626 Authenticator extension output
 4627 The hash value of the content which was displayed. The
 4628 authenticator **MUST** use the same hash algorithm as it uses for
 4629 the signature itself.
 4630
 4631 10.4. Authenticator Selection Extension (authnSel)
 4632
 4633 This registration extension allows a Relying Party to guide the
 4634 selection of the authenticator that will be leveraged when creating the
 4635 credential. It is intended primarily for Relying Parties that wish to
 4636 tightly control the experience around credential creation.
 4637
 4638 Extension identifier
 4639 authnSel
 4640
 4641 Client extension input
 4642 A sequence of AAGUIDs:
 4643
 4644 typedef sequence<AAGUID> AuthenticatorSelectionList;
 4645
 4646 partial dictionary AuthenticationExtensionsClientInputs {
 4647 AuthenticatorSelectionList authnSel;
 4648 };
 4649
 4650 Each AAGUID corresponds to an authenticator model that is
 4651 acceptable to the Relying Party for this credential creation.
 4652 The list is ordered by decreasing preference.
 4653
 4654 An AAGUID is defined as an array containing the globally unique
 4655 identifier of the authenticator model being sought.
 4656
 4657 typedef BufferSource AAGUID;
 4658
 4659 Client extension processing
 4660 This extension can only be used during create(). If the client
 4661 supports the Authenticator Selection Extension, it **MUST** use the
 4662 first available authenticator whose AAGUID is present in the
 4663 AuthenticatorSelectionList. If none of the available
 4664 authenticators match a provided AAGUID, the client **MUST** select
 4665 an authenticator from among the available authenticators to
 4666 generate the credential.
 4667
 4668 Client extension output
 4669 Returns the value true to indicate to the RP that the **extension**
 4670 **was acted upon**.
 4671
 4672 partial dictionary AuthenticationExtensionsClientOutputs {
 4673 boolean authnSel;
 4674 };
 4675

4293 Authenticator extension input
4294 None.
4295
4296 Authenticator extension processing
4297 None.
4298
4299 Authenticator extension output
4300 None.
4301
4302 10.5. Supported Extensions Extension (exts)
4303
4304 This registration extension enables the Relying Party to determine
4305 which extensions the authenticator supports.
4306
4307 Extension identifier
4308 exts
4309
4310 Client extension input
4311 The Boolean value true to indicate that this extension is
4312 requested by the Relying Party.
4313
4314
4315 Client extension processing
4316 None, except creating the authenticator extension input from the
4317 client extension input.
4318
4319 Client extension output
4320 Returns the list of supported extensions as a JSON array of
extension identifier strings

4321
4322 Authenticator extension input
4323 The Boolean value true, encoded in CBOR (major type 7, value
4324 21).
4325
4326 Authenticator extension processing
4327 The authenticator sets the authenticator extension output to be
4328 a list of extensions that the authenticator supports, as defined
4329 below. This extension can be added to attestation objects.
4330
4331 Authenticator extension output
4332 The SupportedExtensions extension is a list (CBOR array) of
4333 extension identifier (UTF-8 encoded strings).
4334
4335 10.6. User Verification Index Extension (uvi)
4336
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
4348 Client extension processing
None, except creating the authenticator extension input from the

4676 Authenticator extension input
4677 None.
4678
4679 Authenticator extension processing
4680 None.
4681
4682 Authenticator extension output
4683 None.
4684
4685 10.5. Supported Extensions Extension (exts)
4686
4687 This registration extension enables the Relying Party to determine
4688 which extensions the authenticator supports.
4689
4690 Extension identifier
4691 exts
4692
4693 Client extension input
4694 The Boolean value true to indicate that this extension is
4695 requested by the Relying Party.
4696
4697 partial dictionary AuthenticationExtensionsClientInputs {
4698 boolean exts;
4699 };
4700
4701 Client extension processing
4702 None, except creating the authenticator extension input from the
4703 client extension input.
4704
4705 Client extension output
4706 Returns the list of supported extensions as an array of
4707 extension identifier strings.
4708
4709 typedef sequence<USVString> AuthenticationExtensionsSupported;
4710
4711 partial dictionary AuthenticationExtensionsClientOutputs {
4712 AuthenticationExtensionsSupported exts;
4713 };
4714
4715 Authenticator extension input
4716 The Boolean value true, encoded in CBOR (major type 7, value
4717 21).
4718
4719 Authenticator extension processing
4720 The authenticator sets the authenticator extension output to be
4721 a list of extensions that the authenticator supports, as defined
4722 below. This extension can be added to attestation objects.
4723
4724 Authenticator extension output
4725 The SupportedExtensions extension is a list (CBOR array) of
4726 extension identifier (UTF-8 encoded) strings.
4727
4728 10.6. User Verification Index Extension (uvi)
4729
4730 This registration extension and authentication extension enables use of
4731 a user verification index.
4732
4733 Extension identifier
4734 uvi
4735
4736 Client extension input
4737 The Boolean value true to indicate that this extension is
4738 requested by the Relying Party.
4739
4740 partial dictionary AuthenticationExtensionsClientInputs {
4741 boolean uvi;
4742 };
4743
4744 Client extension processing
None, except creating the authenticator extension input from the

434e client extension input.
 4350
 4351 Client extension output
 4352 Returns a JSON string containing the base64url encoding of the
 4353 authenticator extension output

4354
 4355 Authenticator extension input
 4356 The Boolean value true, encoded in CBOR (major type 7, value
 4357 21).
 4358
 4359 Authenticator extension processing
 4360 The authenticator sets the authenticator extension output to be
 4361 a user verification index indicating the method used by the user
 4362 to authorize the operation, as defined below. This extension can
 4363 be added to attestation objects and assertions.
 4364
 4365 Authenticator extension output
 4366 The user verification index (UVI) is a value uniquely
 4367 identifying a user verification data record. The UVI is encoded
 4368 as CBOR byte string (type 0x58). Each UVI value MUST be specific
 4369 to the related key (in order to provide unlinkability). It also
 4370 **must** contain sufficient entropy that makes guessing impractical.
 4371 UVI values MUST NOT be reused by the Authenticator (for other
 4372 biometric data or users).
 4373
 4374 The UVI data can be used by servers to understand whether an
 4375 authentication was authorized by the exact same biometric data
 4376 as the initial key generation. This allows the detection and
 4377 prevention of "friendly fraud".
 4378
 4379 As an example, the UVI could be computed as SHA256(KeyID ||
 4380 SHA256(rawUVI)), where || represents concatenation, and the
 4381 rawUVI reflects (a) the biometric reference data, (b) the
 4382 related OS level user ID and (c) an identifier which changes
 4383 whenever a factory reset is performed for the device, e.g.
 4384 rawUVI = biometricReferenceData || OSLevelUserID ||
 4385 FactoryResetCounter.
 4386
 4387 Servers supporting UVI extensions MUST support a length of up to
 4388 32 bytes for the UVI value.
 4389
 4390 Example for authenticator data containing one UVI extension
 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
 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

474e client extension input.
 474f
 474g Client extension output
 474h Returns the authenticator extension output as an ArrayBuffer.
 474i
 4750 partial dictionary AuthenticationExtensionsClientOutputs {
 4751 ArrayBuffer uvi;
 4752 };
 4753
 4754 Authenticator extension input
 4755 The Boolean value true, encoded in CBOR (major type 7, value
 4756 21).
 4757
 4758 Authenticator extension processing
 4759 The authenticator sets the authenticator extension output to be
 4760 a user verification index indicating the method used by the user
 4761 to authorize the operation, as defined below. This extension can
 4762 be added to attestation objects and assertions.
 4763
 4764 Authenticator extension output
 4765 The user verification index (UVI) is a value uniquely
 4766 identifying a user verification data record. The UVI is encoded
 4767 as CBOR byte string (type 0x58). Each UVI value MUST be specific
 4768 to the related key (in order to provide unlinkability). It also
 4769 **MUST** contain sufficient entropy that makes guessing impractical.
 4770 UVI values MUST NOT be reused by the Authenticator (for other
 4771 biometric data or users).
 4772
 4773 The UVI data can be used by servers to understand whether an
 4774 authentication was authorized by the exact same biometric data
 4775 as the initial key generation. This allows the detection and
 4776 prevention of "friendly fraud".
 4777
 4778 As an example, the UVI could be computed as SHA256(KeyID ||
 4779 SHA256(rawUVI)), where || represents concatenation, and the
 4780 rawUVI reflects (a) the biometric reference data, (b) the
 4781 related OS level user ID and (c) an identifier which changes
 4782 whenever a factory reset is performed for the device, e.g.
 4783 rawUVI = biometricReferenceData || OSLevelUserID ||
 4784 FactoryResetCounter.
 4785
 4786 Servers supporting UVI extensions MUST support a length of up to
 4787 32 bytes for the UVI value.
 4788
 4789 Example for authenticator data containing one UVI extension
 4790
 4791 ... -- [=RP ID=] hash (32 bytes)
 4792 81 -- UP and ED set
 4793 00 00 00 01 -- (initial) signature counter
 4794 ... -- all public key alg etc.
 4795 A1 -- extension: CBOR map of one elemen
 4796 t
 4797 63 -- Key 1: CBOR text string of 3 byte
 4798 s
 4799 75 76 69 -- "uvi" [=UTF-8 encoded=] string
 4800 58 20 -- Value 1: CBOR byte string with 0x
 4801 20 bytes
 4802 43 B8 E3 BE 27 95 8C 28 -- the UVI value itself
 4803 D5 74 BF 46 8A 85 CF 46
 4804 9A 14 F0 E5 16 69 31 DA
 4805 4B CF FF C1 BB 11 32 82

4807
 4808 10.7. Location Extension (loc)
 4809
 4810 The location registration extension and authentication extension
 4811 provides the client device's current location to the WebAuthn Relying
 4812 Party.
 4813
 4814 Extension identifier

```

441E loc
441F
441E Client extension input
441E The Boolean value true to indicate that this extension is
442C requested by the Relying Party.
4421

442Z Client extension processing
442C None, except creating the authenticator extension input from the
4424 client extension input.
442E
442E Client extension output
4427 Returns a JSON object that encodes the location information in
442E the authenticator extension output as a Coordinates value, as
442E defined by The W3C Geolocation API Specification.

443C
4431 Authenticator extension input
4432 The Boolean value true, encoded in CBOR (major type 7, value
4433 21).
4434
443E Authenticator extension processing
443E If the authenticator does not support the extension, then the
4437 authenticator MUST ignore the extension request. If the
443E authenticator accepts the extension, then the authenticator
443E SHOULD only add this extension data to a packed attestation or
444C assertion.

444Z Authenticator extension output
444C If the authenticator accepts the extension request, then
4444 authenticator extension output SHOULD provide location data in
444E the form of a CBOR-encoded map, with the first value being the
444E extension identifier and the second being an array of returned
4447 values. The array elements SHOULD be derived from (key,value)
444E pairings for each location attribute that the authenticator
444E supports. The following is an example of authenticator data
445C where the returned array is comprised of a {longitude, latitude,
4451 altitude} triplet, following the coordinate representation
4452 defined in The W3C Geolocation API Specification.
4453
4454 ... -- [=RP ID=] hash (32 bytes)
4455 81 -- UP and ED set
445E 00 00 00 01 -- (initial) signature counter
4457 ... -- all public key alg etc.
445E A1 -- extension: CBOR map of one elemen
445E t
446C 63 -- Value 1: CBOR text string of 3 by
4461 tes
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
446E 6C 61 74 69 74 75 64 65 -- "latitude" [=UTF-8 encoded=] stri
446E ng
4467 FB ... -- Element 2: Latitude as CBOR encoded double-p
446E recision float
446E 69 -- Element 3: CBOR text string of 9 bytes
447C 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
447E 61 6C 74 69 74 75 64 65 -- "altitude" [=UTF-8 encoded=] stri
447E ng
447E FB ... -- Element 6: Altitude as CBOR encoded double-p

```

```

481E loc
481E
4817 Client extension input
481E The Boolean value true to indicate that this extension is
481E requested by the Relying Party.
482C
4821 partial dictionary AuthenticationExtensionsClientInputs {
4822 boolean loc;
4823 };
4824
482E Client extension processing
482E None, except creating the authenticator extension input from the
4827 client extension input.
482E
482E Client extension output
4830 Returns a JavaScript object that encodes the location
4831 information in the authenticator extension output as a
4832 Coordinates value, as defined by [Geolocation-API].
4833
4834 partial dictionary AuthenticationExtensionsClientOutputs {
4835 Coordinates loc;
4836 };
4837
483E Authenticator extension input
483E The Boolean value true, encoded in CBOR (major type 7, value
484C 21).
4841
484E Authenticator extension processing
484E Determine the Geolocation value.

4844
484E Authenticator extension output
484E A [Geolocation-API] Coordinates record encoded as a CBOR map.
4847 Values represented by the "double" type in JavaScript are
484E represented as 64-bit CBOR floating point numbers. Per the
484E Geolocation specification, the "latitude", "longitude", and
485C "accuracy" values are required and other values such as
4851 "altitude" are optional.

```

447E recision float
 447F
 448C 10.8. User Verification Method Extension (uvm)
 4481
 4482 This registration extension and authentication extension enables use of
 4483 a user verification method.
 4484
 4485 Extension identifier
 4486 uvm
 4487
 4488 Client extension input
 4489 The Boolean value true to indicate that this extension is
 449C requested by the WebAuthn Relying Party.

4491
 4492 Client extension processing
 4493 None, except creating the authenticator extension input from the
 4494 client extension input.
 4495
 4496 Client extension output
 4497 Returns a JSON array of 3-element arrays of numbers that encodes
 449E the factors in the authenticator extension output

449C
 450C Authenticator extension input
 4501 The Boolean value true, encoded in CBOR (major type 7, value
 4502 21).
 4503

4504 Authenticator extension processing
 4505 The authenticator sets the authenticator extension output to be
 4506 one or more user verification methods indicating the method(s)
 4507 used by the user to authorize the operation, as defined below.
 4508 This extension can be added to attestation objects and
 4509 assertions.
 4510

4511 Authenticator extension output
 4512 Authenticators can report up to 3 different user verification
 4513 methods (factors) used in a single authentication instance,
 4514 using the CBOR syntax defined below:
 4515

```
4516 uvmFormat = [ 1*3 uvmEntry ]
4517 uvmEntry = [
4518     userVerificationMethod: uint .size 4,
4519     keyProtectionType: uint .size 2,
4520     matcherProtectionType: uint .size 2
4521 ]
```

4522 The semantics of the fields in each uvmEntry are as follows:
 4523

4524 userVerificationMethod
 4525 The authentication method/factor used by the authenticator
 4526 to verify the user. Available values are defined in
 4527 [FIDOReg], "User Verification Methods" section.
 4528

4529 keyProtectionType
 4530 The method used by the authenticator to protect the FIDO
 4531 registration private key material. Available values are
 4532 defined in [FIDOReg], "Key Protection Types" section.
 4533

4534 matcherProtectionType
 4535 The method used by the authenticator to protect the
 4536

4852 10.8. User Verification Method Extension (uvm)
 4853
 4854 This registration extension and authentication extension enables use of
 4855 a user verification method.
 4856
 4857 Extension identifier
 4858 uvm
 4859
 4860 Client extension input
 4861 The Boolean value true to indicate that this extension is
 4862 requested by the Relying Party.
 4863

```
4864 partial dictionary AuthenticationExtensionsClientInputs {
4865     boolean uvm;
4866 };
```

4867 Client extension processing
 4868 None, except creating the authenticator extension input from the
 4870 client extension input.
 4871

4872 Client extension output
 4873 Returns a JSON array of 3-element arrays of numbers that encodes
 4874 the factors in the authenticator extension output.
 4875

```
4876 typedef sequence<unsigned long> UvmEntry;
4877 typedef sequence<UvmEntry> UvmEntries;
```

```
4878 partial dictionary AuthenticationExtensionsClientOutputs {
4879     UvmEntries uvm;
4880 };
```

4881 Authenticator extension input
 4882 The Boolean value true, encoded in CBOR (major type 7, value
 4883 21).
 4884

4885 Authenticator extension processing
 4886 The authenticator sets the authenticator extension output to be
 4887 one or more user verification methods indicating the method(s)
 4888 used by the user to authorize the operation, as defined below.
 4889 This extension can be added to attestation objects and
 4890 assertions.
 4891

4892 Authenticator extension output
 4893 Authenticators can report up to 3 different user verification
 4894 methods (factors) used in a single authentication instance,
 4895 using the CBOR syntax defined below:
 4896

```
4897 uvmFormat = [ 1*3 uvmEntry ]
4898 uvmEntry = [
4899     userVerificationMethod: uint .size 4,
4900     keyProtectionType: uint .size 2,
4901     matcherProtectionType: uint .size 2
4902 ]
```

4903 The semantics of the fields in each uvmEntry are as follows:
 4904

4905 userVerificationMethod
 4906 The authentication method/factor used by the authenticator
 4907 to verify the user. Available values are defined in
 4908 [FIDOReg], "User Verification Methods" section.
 4909

4910 keyProtectionType
 4911 The method used by the authenticator to protect the FIDO
 4912 registration private key material. Available values are
 4913 defined in [FIDOReg], "Key Protection Types" section.
 4914

4915 matcherProtectionType
 4916 The method used by the authenticator to protect the
 4917
 4918
 4919
 4920

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

4921 matcher that performs user verification. Available values
 4922 are defined in [FIDOReg], "Matcher Protection Types"
 4923 section.
 4924
 4925 If >3 factors can be used in an authentication instance the
 4926 authenticator vendor **MUST** select the 3 factors it believes will
 4927 be most relevant to the Server to include in the UVM.
 4928
 4929 Example for authenticator data containing one UVM extension for
 4930 a multi-factor authentication instance where 2 factors were
 4931 used:
 4932
 4933 ... -- [=RP ID=] hash (32 bytes)
 4934 81 -- UP and ED set
 4935 00 00 00 01 -- (initial) signature counter
 4936 ... -- all public key alg etc.
 4937 A1 -- extension: CBOR map of one element
 4938 63 -- Key 1: CBOR text string of 3 bytes
 4939 75 76 6d -- "uvm" [=UTF-8 encoded=] string
 4940 82 -- Value 1: CBOR array of length 2 indicating two factor
 4941 usage
 4942 83 -- Item 1: CBOR array of length 3
 4943 02 -- Subitem 1: CBOR integer for User Verification Method
 4944 Fingerprint
 4945 04 -- Subitem 2: CBOR short for Key Protection Type TEE
 4946 02 -- Subitem 3: CBOR short for Matcher Protection Type TE
 4947 E
 4948 83 -- Item 2: CBOR array of length 3
 4949 04 -- Subitem 1: CBOR integer for User Verification Method
 4950 Passcode
 4951 01 -- Subitem 2: CBOR short for Key Protection Type Softwa
 4952 re
 4953 01 -- Subitem 3: CBOR short for Matcher Protection Type So
 4954 ftware
 4955

4956 **10.9. Biometric Authenticator Performance Bounds Extension**
 4957 **(biometricPerfBounds)**

4958
 4959 This registration extension allows Relying Parties to specify the
 4960 desired performance bounds for selecting biometric authenticators as
 4961 candidates to be employed in a registration ceremony.
 4962

4963 Extension identifier
 4964 biometricPerfBounds

4965 Client extension input
 4966 Biometric performance bounds:

4967 dictionary authenticatorBiometricPerfBounds{
 4968 float FAR;
 4969 float FRR;
 4970 };

4971 The FAR is the maximum false acceptance rate for a biometric
 4972 authenticator allowed by the Relying Party.

4973 The FAR is the maximum false rejection rate for a biometric
 4974 authenticator allowed by the Relying Party.

4975 Client extension processing
 4976 This extension can only be used during create(). If the client
 4977 supports this extension, it **MUST NOT** use a biometric
 4978 authenticator whose FAR or FRR does not match the bounds as
 4979 provided. The client can obtain information about the biometric
 4980 authenticator's performance from authoritative sources such as
 4981 the FIDO Metadata Service [FIDOMetadataService] (see Sec. 3.2 of
 4982 [FIDOUAFAuthenticatorMetadataStatements]).

4983 Client extension output
 4984 Returns the JSON value true to indicate to the RP that the

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
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

4991 extension was acted upon
4992
4993 Authenticator extension input
4994 None.
4995
4996 Authenticator extension processing
4997 None.
4998
4999 Authenticator extension output
5000 None.

5001 11. IANA Considerations
5002
5003 11.1. WebAuthn Attestation Statement Format Identifier Registrations
5004
5005 This section registers the attestation statement formats defined in
5006 Section 8 Defined Attestation Statement Formats in the IANA "WebAuthn
5007 Attestation Statement Format Identifier" registry established by
5008 [WebAuthn-Registries].
5009 * WebAuthn Attestation Statement Format Identifier: packed
5010 * Description: The "packed" attestation statement format is a
5011 WebAuthn-optimized format for attestation. It uses a very compact
5012 but still extensible encoding method. This format is implementable
5013 by authenticators with limited resources (e.g., secure elements).
5014 * Specification Document: Section 8.2 Packed Attestation Statement
5015 Format of this specification
5016 * WebAuthn Attestation Statement Format Identifier: tpm
5017 * Description: The TPM attestation statement format returns an
5018 attestation statement in the same format as the packed attestation
5019 statement format, although the rawData and signature fields are
5020 computed differently.
5021 * Specification Document: Section 8.3 TPM Attestation Statement
5022 Format of this specification
5023 * WebAuthn Attestation Statement Format Identifier: android-key
5024 * Description: Platform-provided authenticators based on versions
5025 "N", and later, may provide this proprietary "hardware attestation"
5026 statement.
5027 * Specification Document: Section 8.4 Android Key Attestation
5028 Statement Format of this specification
5029 * WebAuthn Attestation Statement Format Identifier: android-safetynet
5030 * Description: Android-based, platform-provided authenticators MAY
5031 produce an attestation statement based on the Android SafetyNet
5032 API.
5033 * Specification Document: Section 8.5 Android SafetyNet Attestation
5034 Statement Format of this specification
5035 * WebAuthn Attestation Statement Format Identifier: fido-u2f
5036 * Description: Used with FIDO U2F authenticators
5037 * Specification Document: Section 8.6 FIDO U2F Attestation Statement
5038 Format of this specification
5039
5040 11.2. WebAuthn Extension Identifier Registrations
5041
5042 This section registers the extension identifier values defined in
5043 Section 9 WebAuthn Extensions in the IANA "WebAuthn Extension
5044 Identifier" registry established by [WebAuthn-Registries].
5045 * WebAuthn Extension Identifier: appid
5046 * Description: This authentication extension allows Relying Parties
5047 that have previously registered a credential using the legacy FIDO
5048 JavaScript APIs to request an assertion.
5049 * Specification Document: Section 10.1 FIDO AppID Extension (appid)
5050 of this specification
5051 * WebAuthn Extension Identifier: txAuthSimple
5052 * Description: This registration extension and authentication
5053 extension allows for a simple form of transaction authorization. A
5054 WebAuthn Relying Party can specify a prompt string, intended for
5055 display on a trusted device on the authenticator
5056 * Specification Document: Section 10.2 Simple Transaction
5057 Authorization Extension (txAuthSimple) of this specification
5058 * WebAuthn Extension Identifier: txAuthGeneric
5059 * 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

4680 This section registers identifiers for RSASSA-PKCS1-v1_5 [RFC8017]
4681 algorithms using SHA-2 and SHA-1 hash functions in the IANA COSE
4682 Algorithms registry [IANA-COSE-ALGS-REG]. It also registers identifiers
4683 for ECDAA algorithms.

- 4684 * Name: RS256
- 4685 * Value: -257
- 4686 * Description: RSASSA-PKCS1-v1_5 w/ SHA-256
- 4687 * Reference: Section 8.2 of [RFC8017]
- 4688 * Recommended: No
- 4689 * Name: RS384
- 4690 * Value: -258
- 4691 * Description: RSASSA-PKCS1-v1_5 w/ SHA-384
- 4692 * Reference: Section 8.2 of [RFC8017]
- 4693 * Recommended: No
- 4694 * Name: RS512
- 4695 * Value: -259
- 4696 * Description: RSASSA-PKCS1-v1_5 w/ SHA-512
- 4697 * Reference: Section 8.2 of [RFC8017]

4700

5061 extension allows images to be used as transaction authorization
5062 prompts as well. This allows authenticators without a font
5063 rendering engine to be used and also supports a richer visual
5064 appearance than accomplished with the webauthn.txauth.simple
5065 extension.
5066 * Specification Document: Section 10.3 Generic Transaction
5067 Authorization Extension (txAuthGeneric) of this specification
5068 * WebAuthn Extension Identifier: authnSel
5069 * Description: This registration extension allows a WebAuthn Relying
5070 Party to guide the selection of the authenticator that will be
5071 leveraged when creating the credential. It is intended primarily
5072 for WebAuthn Relying Parties that wish to tightly control the
5073 experience around credential creation.
5074 * Specification Document: Section 10.4 Authenticator Selection
5075 Extension (authnSel) of this specification
5076 * WebAuthn Extension Identifier: exts
5077 * Description: This registration extension enables the Relying Party
5078 to determine which extensions the authenticator supports. The
5079 extension data is a list (CBOR array) of extension identifiers
5080 encoded as UTF-8 Strings. This extension is added automatically by
5081 the authenticator. This extension can be added to attestation
5082 statements.
5083 * Specification Document: Section 10.5 Supported Extensions
5084 Extension (exts) of this specification
5085 * WebAuthn Extension Identifier: uvi
5086 * Description: This registration extension and authentication
5087 extension enables use of a user verification index. The user
5088 verification index is a value uniquely identifying a user
5089 verification data record. The UVI data can be used by servers to
5090 understand whether an authentication was authorized by the exact
5091 same biometric data as the initial key generation. This allows the
5092 detection and prevention of "friendly fraud".
5093 * Specification Document: Section 10.6 User Verification Index
5094 Extension (uvi) of this specification
5095 * WebAuthn Extension Identifier: loc
5096 * Description: The location registration extension and authentication
5097 extension provides the client device's current location to the
5098 WebAuthn relying party, if supported by the client device and
5099 subject to user consent.
5100 * Specification Document: Section 10.7 Location Extension (loc) of
5101 this specification
5102 * WebAuthn Extension Identifier: uvm
5103 * Description: This registration extension and authentication
5104 extension enables use of a user verification method. The user
5105 verification method extension returns to the Webauthn relying party
5106 which user verification methods (factors) were used for the
5107 WebAuthn operation.
5108 * Specification Document: Section 10.8 User Verification Method
5109 Extension (uvm) of this specification

11.3. COSE Algorithm Registrations

5110 This section registers identifiers for RSASSA-PKCS1-v1_5 [RFC8017]
5111 algorithms using SHA-2 and SHA-1 hash functions in the IANA COSE
5112 Algorithms registry [IANA-COSE-ALGS-REG]. It also registers identifiers
5113 for ECDAA algorithms.

- 5114 * Name: RS256
- 5115 * Value: [TBD \(requested assignment -257\)](#)
- 5116 * Description: RSASSA-PKCS1-v1_5 w/ SHA-256
- 5117 * Reference: Section 8.2 of [RFC8017]
- 5118 * Recommended: No
- 5119 * Name: RS384
- 5120 * Value: [TBD \(requested assignment -258\)](#)
- 5121 * Description: RSASSA-PKCS1-v1_5 w/ SHA-384
- 5122 * Reference: Section 8.2 of [RFC8017]
- 5123 * Recommended: No
- 5124 * Name: RS512
- 5125 * Value: [TBD \(requested assignment -259\)](#)
- 5126 * Description: RSASSA-PKCS1-v1_5 w/ SHA-512
- 5127 * Reference: Section 8.2 of [RFC8017]

5130

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.
5. The authenticator shows appropriate UI for the user to select the authenticator on which the new credential will be created, and obtains a biometric or other authorization gesture from the user.
6. The authenticator returns a response to the client platform, which in turn returns a response to the Relying Party script. If the user declined to select an authenticator or provide authorization, an appropriate error is returned.
7. If a new credential was created,
 - + The Relying Party script sends the newly generated credential public key to the server, along with additional information such as attestation regarding the provenance and characteristics of the authenticator.
 - + The server stores the credential public key in its database and associates it with the user as well as with the characteristics of authentication indicated by attestation, also storing a friendly name for later use.

5131 * Recommended: No
 5132 * Name: ED256
 5133 * Value: TBD (requested assignment -260)
 5134 * Description: TPM_ECC_BN_P256 curve w/ SHA-256
 5135 * Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
 5136 * Recommended: Yes
 5137 * Name: ED512
 5138 * Value: TBD (requested assignment -261)
 5139 * Description: ECC_BN_ISOP512 curve w/ SHA-512
 5140 * Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
 5141 * Recommended: Yes
 5142 * Name: RS1
 5143 * Value: TBD (requested assignment -262)
 5144 * Description: RSASSA-PKCS1-v1_5 w/ SHA-1
 5145 * Reference: Section 8.2 of [RFC8017]
 5146 * 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.
5. The authenticator shows appropriate UI for the user to select the authenticator on which the new credential will be created, and obtains a biometric or other authorization gesture from the user.
6. The authenticator returns a response to the client platform, which in turn returns a response to the Relying Party script. If the user declined to select an authenticator or provide authorization, an appropriate error is returned.
7. If a new credential was created,
 - + The Relying Party script sends the newly generated credential public key to the server, along with additional information such as attestation regarding the provenance and characteristics of the authenticator.
 - + The server stores the credential public key in its database and associates it with the user as well as with the characteristics of authentication indicated by attestation, also storing a friendly name for later use.

```

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/aBjjjppqPb.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 });
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

```

```

5201 + The script may store data such as the credential ID in local
5202 storage, to improve future UX by narrowing the choice of
5203 credential for the user.
5204
5205 The sample code for generating and registering a new key follows:
5206 if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */ }
5207
5208 var publicKey = {
5209 // The challenge must be produced by the server, see the Security Consideratio
5210 ns
5211 challenge: new Uint8Array([21,31,105 /* 29 more random bytes generated by the
5212 server */]),
5213
5214 // Relying Party:
5215 rp: {
5216 name: "ACME Corporation"
5217 },
5218
5219 // User:
5220 user: {
5221 id: Uint8Array.from(window.atob("MIIBkzCCATigAwIBAJCCAAMwggE4oAMCAQIwggGTMII
5222 ="), c=>c.charCodeAt(0)),
5223 name: "alex.p.mueller@example.com",
5224 displayName: "Alex P. Miller",
5225 icon: "https://pics.example.com/00/p/aBjjjppqPb.png"
5226 },
5227
5228 // This Relying Party will accept either an ES256 or RS256 credential, but
5229 // prefers an ES256 credential.
5230 pubKeyCredParams: [
5231 {
5232 type: "public-key",
5233 alg: -7 // "ES256" as registered in the IANA COSE Algorithms registry
5234 },
5235 {
5236 type: "public-key",
5237 alg: -257 // Value registered by this specification for "RS256"
5238 }
5239 ],
5240
5241 timeout: 60000, // 1 minute
5242 excludeCredentials: [], // No exclude list of PKCredDescriptors
5243 extensions: {"loc": true} // Include location information
5244 // in attestation
5245 };
5246
5247 // Note: The following call will cause the authenticator to display UI.
5248 navigator.credentials.create({ publicKey })
5249 .then(function (newCredentialInfo) {
5250 // Send new credential info to server for verification and registration.
5251 }).catch(function (err) {
5252 // No acceptable authenticator or user refused consent. Handle appropriately
5253
5254 });
5255
5256 12.2. Registration Specifically with User Verifying Platform Authenticator
5257
5258 This is flow for when the Relying Party is specifically interested in
5259 creating a public key credential with a user-verifying platform
5260 authenticator.
5261 1. The user visits example.com and clicks on the login button, which
5262 redirects the user to login.example.com.
5263 2. The user enters a username and password to log in. After successful
5264 login, the user is redirected back to example.com.
5265 3. The Relying Party script runs the code snippet below.
5266 4. The user agent asks the user whether they are willing to register
5267 with the Relying Party using an available platform authenticator.
5268 5. If the user is not willing, terminate this flow.
5269 6. The user is shown appropriate UI and guided in creating a
5270 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 ava
4850         ilable platform authenticator
4851         if (userIntent) {
4852             var publicKeyOptions = { /* Public key credential creation options.
4853             */};
4854
4855             // Create and register credentials.
4856             return navigator.credentials.create({ "publicKey": publicKeyOptions
4857             });
4858         } else {
4859
4860             // Record that the user does not intend to use a platform authentica
4861             tor
4862             // and default the user to a password-based flow in the future.
4863             }
4864
4865         }).then(function (newCredentialInfo) {
4866             // Send new credential info to server for verification and registration.
4867         }).catch( function(err) {
4868             // Something went wrong. Handle appropriately.
4869         });

```

12.3. Authentication

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

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

```

5271     successful credential creation, the RP script conveys the new
5272     credential to the server.
5273     if (!window.PublicKeyCredential) { /* Platform not capable of the API. Handle er
5274     ror. */}
5275
5276     PublicKeyCredential.isUserVerifyingPlatformAuthenticatorAvailable()
5277     .then(function (userIntent) {
5278
5279         // If the user has affirmed willingness to register with RP using an ava
5280         ilable platform authenticator
5281         if (userIntent) {
5282             var publicKeyOptions = { /* Public key credential creation options.
5283             */};
5284
5285             // Create and register credentials.
5286             return navigator.credentials.create({ "publicKey": publicKeyOptions
5287             });
5288         } else {
5289
5290             // Record that the user does not intend to use a platform authentica
5291             tor
5292             // and default the user to a password-based flow in the future.
5293             }
5294
5295         }).then(function (newCredentialInfo) {
5296             // Send new credential info to server for verification and registration.
5297         }).catch( function(err) {
5298             // Something went wrong. Handle appropriately.
5299         });

```

12.3. Authentication

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

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

```

4910 If the Relying Party script does not have any hints available (e.g.,
4911 from locally stored data) to help it narrow the list of credentials,
4912 then the sample code for performing such an authentication might look
4913 like this:
4914 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4915
4916 var options = {
4917     // The challenge must be produced by the server, see the Security
4918     // Considerations
4919     challenge: new Uint8Array([4,101,15 /* 29 more random bytes generated
4920     by the server */]),
4921     timeout: 60000, // 1 minute
4922     allowCredentials: [{ type: "public-key" }]
4923 };
4924
4925 navigator.credentials.get({ "publicKey": options })
4926     .then(function (assertion) {
4927         // Send assertion to server for verification
4928     }).catch(function (err) {
4929         // No acceptable credential or user refused consent. Handle appropriately.
4930     });
4931
4932 On the other hand, if the Relying Party script has some hints to help
4933 it narrow the list of credentials, then the sample code for performing
4934 such an authentication might look like the following. Note that this
4935 sample also demonstrates how to use the extension for transaction
4936 authorization.
4937 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4938
4939 var encoder = new TextEncoder();
4940 var acceptableCredential1 = {
4941     type: "public-key",
4942     id: encoder.encode("!!!!!!hi there!!!!!!\n")
4943 };
4944 var acceptableCredential2 = {
4945     type: "public-key",
4946     id: encoder.encode("roses are red, violets are blue\n")
4947 };
4948
4949 var options = {
4950     // The challenge must be produced by the server, see the Security
4951     // Considerations
4952     challenge: new Uint8Array([8,18,33 /* 29 more random bytes generated
4953     by the server */]),
4954     timeout: 60000, // 1 minute
4955     allowCredentials: [acceptableCredential1, acceptableCredential2]
4956 ,
4957     extensions: { 'txAuthSimple':
4958         "Wave your hands in the air like you just don't care" }
4959 };
4960
4961 navigator.credentials.get({ "publicKey": options })
4962     .then(function (assertion) {
4963         // Send assertion to server for verification
4964     }).catch(function (err) {
4965         // No acceptable credential or user refused consent. Handle appropriately.
4966     });
4967
4968 12.4. Aborting Authentication Operations
4969
4970 The below example shows how a developer may use the AbortSignal
4971 parameter to abort a credential registration operation. A similar
4972 procedure applies to an authentication operation.
4973 const authAbortController = new AbortController();
4974 const authAbortSignal = authAbortController.signal;
4975
4976 authAbortSignal.onabort = function () {
4977     // Once the page knows the abort started, inform user it is attempting to abort.
4978 }

```

```

5341 If the Relying Party script does not have any hints available (e.g.,
5342 from locally stored data) to help it narrow the list of credentials,
5343 then the sample code for performing such an authentication might look
5344 like this:
5345 if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */ }
5346
5347 var options = {
5348     // The challenge must be produced by the server, see the Security
5349     // Considerations
5350     challenge: new Uint8Array([4,101,15 /* 29 more random bytes generated
5351     by the server */]),
5352     timeout: 60000, // 1 minute
5353     allowCredentials: [{ type: "public-key" }]
5354 };
5355
5356 navigator.credentials.get({ "publicKey": options })
5357     .then(function (assertion) {
5358         // Send assertion to server for verification
5359     }).catch(function (err) {
5360         // No acceptable credential or user refused consent. Handle appropriately.
5361     });
5362
5363 On the other hand, if the Relying Party script has some hints to help
5364 it narrow the list of credentials, then the sample code for performing
5365 such an authentication might look like the following. Note that this
5366 sample also demonstrates how to use the extension for transaction
5367 authorization.
5368 if (!window.PublicKeyCredential) { /* Platform not capable. Handle error. */ }
5369
5370 var encoder = new TextEncoder();
5371 var acceptableCredential1 = {
5372     type: "public-key",
5373     id: encoder.encode("!!!!!!hi there!!!!!!\n")
5374 };
5375 var acceptableCredential2 = {
5376     type: "public-key",
5377     id: encoder.encode("roses are red, violets are blue\n")
5378 };
5379
5380 var options = {
5381     // The challenge must be produced by the server, see the Security
5382     // Considerations
5383     challenge: new Uint8Array([8,18,33 /* 29 more random bytes generated
5384     by the server */]),
5385     timeout: 60000, // 1 minute
5386     allowCredentials: [acceptableCredential1, acceptableCredential2]
5387 ,
5388     extensions: { 'txAuthSimple':
5389         "Wave your hands in the air like you just don't care" }
5390 };
5391
5392 navigator.credentials.get({ "publicKey": options })
5393     .then(function (assertion) {
5394         // Send assertion to server for verification
5395     }).catch(function (err) {
5396         // No acceptable credential or user refused consent. Handle appropriately.
5397     });
5398
5399 12.4. Aborting Authentication Operations
5400
5401 The below example shows how a developer may use the AbortSignal
5402 parameter to abort a credential registration operation. A similar
5403 procedure applies to an authentication operation.
5404 const authAbortController = new AbortController();
5405 const authAbortSignal = authAbortController.signal;
5406
5407 authAbortSignal.onabort = function () {
5408     // Once the page knows the abort started, inform user it is attempting to abort.
5409 }

```

```

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
5005 12.5. Decommissioning
5006
5007 The following are possible situations in which decommissioning a
5008 credential might be desired. Note that all of these are handled on the
5009 server side and do not need support from the API specified here.
5010 * Possibility #1 -- user reports the credential as lost.
5011   + User goes to server.example.net, authenticates and follows a
5012   link to report a lost/stolen device.
5013   + Server returns a page showing the list of registered
5014   credentials with friendly names as configured during
5015   registration.
5016   + User selects a credential and the server deletes it from its
5017   database.
5018   + In future, the Relying Party script does not specify this
5019   credential in any list of acceptable credentials, and
5020   assertions signed by this credential are rejected.
5021 * Possibility #2 -- server deregisters the credential due to
5022 inactivity.
5023   + Server deletes credential from its database during maintenance
5024   activity.
5025   + In the future, the Relying Party script does not specify this
5026   credential in any list of acceptable credentials, and
5027   assertions signed by this credential are rejected.
5028 * Possibility #3 -- user deletes the credential from the device.
5029   + User employs a device-specific method (e.g., device settings
5030   UI) to delete a credential from their device.
5031   + From this point on, this credential will not appear in any
5032   selection prompts, and no assertions can be generated with it.
5033   + Sometime later, the server deregisters this credential due to
5034   inactivity.
5035
5036 13. Security Considerations

```

```

5411 }
5412
5413 var options = {
5414   // A list of options.
5415 }
5416
5417 navigator.credentials.create({
5418   publicKey: options,
5419   signal: authAbortSignal})
5420   .then(function (attestation) {
5421     // Register the user.
5422   }).catch(function (error) {
5423     if (error == "AbortError") {
5424       // Inform user the credential hasn't been created.
5425       // Let the server know a key hasn't been created.
5426     }
5427   });
5428
5429 // Assume widget shows up whenever authentication occurs.
5430 if (widget == "disappear") {
5431   authAbortSignal.abort();
5432 }
5433
5434 }
5435
5436 12.5. Decommissioning
5437
5438 The following are possible situations in which decommissioning a
5439 credential might be desired. Note that all of these are handled on the
5440 server side and do not need support from the API specified here.
5441 * Possibility #1 -- user reports the credential as lost.
5442   + User goes to server.example.net, authenticates and follows a
5443   link to report a lost/stolen device.
5444   + Server returns a page showing the list of registered
5445   credentials with friendly names as configured during
5446   registration.
5447   + User selects a credential and the server deletes it from its
5448   database.
5449   + In future, the Relying Party script does not specify this
5450   credential in any list of acceptable credentials, and
5451   assertions signed by this credential are rejected.
5452 * Possibility #2 -- server deregisters the credential due to
5453 inactivity.
5454   + Server deletes credential from its database during maintenance
5455   activity.
5456   + In the future, the Relying Party script does not specify this
5457   credential in any list of acceptable credentials, and
5458   assertions signed by this credential are rejected.
5459 * Possibility #3 -- user deletes the credential from the device.
5460   + User employs a device-specific method (e.g., device settings
5461   UI) to delete a credential from their device.
5462   + From this point on, this credential will not appear in any
5463   selection prompts, and no assertions can be generated with it.
5464   + Sometime later, the server deregisters this credential due to
5465   inactivity.
5466
5467 13. Security Considerations
5468
5469 This specification defines a Web API and a cryptographic peer-entity
5470 authentication protocol. The Web Authentication API allows Web
5471 developers (i.e., "authors") to utilize the Web Authentication protocol
5472 in their registration and authentication ceremonies. The entities
5473 comprising the Web Authentication protocol endpoints are
5474 user-controlled authenticators and a Relying Party's computing
5475 environment hosting the Relying Party's web application. In this model,
5476 the user agent, together with the WebAuthn Client, comprise an
5477 intermediary between authenticators and Relying Parties. Additionally,
5478 authenticators can attest to Relying Parties as to their provenance.
5479
5480 At this time, this specification does not feature detailed security
5481 considerations. However, the [FIDOSecRef] document provides a security

```

13.1. Cryptographic Challenges

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.

14. Acknowledgements

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

analysis which is overall applicable to this specification. Also, the [FIDOAuthnrSecReqs] document suite defines authenticator security characteristics which are overall applicable for WebAuthn authenticators.

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

13.1. Cryptographic Challenges

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

13.2. Attestation Security Considerations

13.2.1. Attestation Certificate Hierarchy

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

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

13.2.2. Attestation Certificate and Attestation Certificate CA Compromise

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

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

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

5549
5550
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

If an ECDAAs attestation key has been compromised, it can be added to the RogueList (i.e., the list of revoked authenticators) maintained by the related ECDAAs-Issuer. The Relying Party SHOULD verify whether an authenticator belongs to the RogueList when performing ECDAAs-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 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,

5057 Boris Zbarsky.

5058

5059 Index

5060 Terms defined by this specification

5061 * aaguid, in 6.3.1

5062 * AAGUID, in 10.4

5063 * alg, in 5.3

5619 sameOriginWithAncestors) method need to take care to not leak
5620 information that could enable a malicious Relying Party to distinguish
5621 between these cases, where "excluded" means that at least one of the
5622 credentials listed by the Relying Party in excludeCredentials is bound
5623 to the authenticator:
5624 * No authenticators are present.
5625 * At least one authenticator is present, and at least one present
5626 authenticator is excluded.
5627

5628 If the above cases are distinguishable, information is leaked by which
5629 a malicious Relying Party could identify the user by probing for which
5630 credentials are available. For example, one such information leak is if
5631 the client returns a failure response as soon as an excluded
5632 authenticator becomes available. In this case - especially if the
5633 excluded authenticator is a platform authenticator - the Relying Party
5634 could detect that the ceremony was canceled before the timeout and
5635 before the user could feasibly have canceled it manually, and thus
5636 conclude that at least one of the credentials listed in the
5637 excludeCredentials parameter is available to the user.
5638

5639 The above is not a concern, however, if the user has consented to
5640 create a new credential before a distinguishable error is returned,
5641 because in this case the user has confirmed intent to share the
5642 information that would be leaked.
5643

5644 14.3. Authentication Ceremony Privacy

5645 In order to protect users from being identified without consent,
5646 implementations of the [[DiscoverFromExternalSource]](origin, options,
5647 sameOriginWithAncestors) method need to take care to not leak
5648 information that could enable a malicious Relying Party to distinguish
5649 between these cases, where "named" means that the credential is listed
5650 by the Relying Party in allowCredentials:
5651 * A named credential is not available.
5652 * A named credential is available, but the user does not consent to
5653 use it.
5654

5655 If the above cases are distinguishable, information is leaked by which
5656 a malicious Relying Party could identify the user by probing for which
5657 credentials are available. For example, one such information leak is if
5658 the client returns a failure response as soon as the user denies
5659 consent to proceed with an authentication ceremony. In this case the
5660 Relying Party could detect that the ceremony was canceled by the user
5661 and not the timeout, and thus conclude that at least one of the
5662 credentials listed in the allowCredentials parameter is available to
5663 the user.
5664

5665 15. Acknowledgements

5666 We thank the following people for their reviews of, and contributions
5667 to, this specification: Yuriy Ackermann, James Barclay, Richard Barnes,
5668 Dominic Battr, John Bradley, Domenic Denicola, Rahul Ghosh, Brad Hill,
5669 Jing Jin, Wally Jones, Ian Kilpatrick, Axel Nennker, Yoshikazu Nojima,
5670 Kimberly Paulhamus, Adam Powers, Yaron Sheffer, Anne van Kesteren, and
5671 Boris Zbarsky.
5672

5673 We thank Anthony Nadalin, John Fontana, and Richard Barnes for their
5674 contributions as co-chairs of the Web Authentication Working Group.
5675

5676 We also thank Wendy Seltzer, Samuel Weiler, and Harry Halpin for their
5677 contributions as our W3C Team Contacts.
5678

5679 Index

5680 Terms defined by this specification

5681 * AAGUID, in 10.4

5682 * aaguid, in 6.3.1

5683 * alg, in 5.3

5066 * allowCredentials, in 5.5

5067 * Assertion, in 4
 5068 * assertion signature, in 6
 5069 * attachment modality, in 5.4.5

5070 * Attestation, in 4
 5071 * attestation, in 5.4

5072 * Attestation Certificate, in 4
 5073 * Attestation Conveyance, in 5.4.6
 5074 * AttestationConveyancePreference, in 5.4.6
 5075 * attestationConveyancePreferenceOption, in 5.1.3
 5076 * attestation key pair, in 4
 5077 * attestationObject, in 5.2.1
 5078 * attestation object, in 6.3
 5079 * attestationObjectResult, in 5.1.3
 5080 * attestation private key, in 4
 5081 * attestation public key, in 4
 5082 * attestation signature, in 6
 5083 * attestation statement, in 6.3
 5084 * attestation statement format, in 6.3
 5085 * attestation statement format identifier, in 8.1
 5086 * attestation trust path, in 6.3.2
 5087 * attestation type, in 6.3
 5088 * Attested credential data, in 6.3.1
 5089 * attestedCredentialData, in 6.1
 5090 * authDataExtensions, in 6.1
 5091 * Authentication, in 4
 5092 * Authentication Assertion, in 4
 5093 * authentication extension, in 9
 5094 * AuthenticationExtensions

5095 + definition of, in 5.7
 5096 + (typedef), in 5.7

5097 * Authenticator, in 4
 5098 * AuthenticatorAssertionResponse, in 5.2.2
 5099 * AuthenticatorAttachment, in 5.4.5
 5100 * authenticatorAttachment, in 5.4.4
 5101 * AuthenticatorAttestationResponse, in 5.2.1
 5102 * authenticatorCancel, in 6.2.3

5103 * authenticator data, in 6.1
 5104 * 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

5688 * allowCredentials, in 5.5
 5689 * Anonymization CA, in 14.1
 5690 * appid
 5691 + dict-member for AuthenticationExtensionsClientInputs, in 10.1
 5692 + dict-member for AuthenticationExtensionsClientOutputs, in
 5693 10.1
 5694 * Assertion, in 4
 5695 * assertion signature, in 6
 5696 * attachment modality, in 5.4.5
 5697 * AttCA, in 6.3.3
 5698 * Attestation, in 4
 5699 * attestation, in 5.4
 5700 * Attestation CA, in 6.3.3
 5701 * Attestation Certificate, in 4
 5702 * Attestation Conveyance, in 5.4.6
 5703 * AttestationConveyancePreference, in 5.4.6
 5704 * attestationConveyancePreferenceOption, in 5.1.3
 5705 * attestation key pair, in 4
 5706 * attestationObject, in 5.2.1
 5707 * attestation object, in 6.3
 5708 * attestationObjectResult, in 5.1.3
 5709 * attestation private key, in 4
 5710 * attestation public key, in 4
 5711 * attestation signature, in 6
 5712 * attestation statement, in 6.3
 5713 * attestation statement format, in 6.3
 5714 * attestation statement format identifier, in 8.1
 5715 * attestation trust path, in 6.3.2
 5716 * attestation type, in 6.3
 5717 * Attested credential data, in 6.3.1
 5718 * attestedCredentialData, in 6.1
 5719 * authDataExtensions, in 6.1
 5720 * Authentication, in 4
 5721 * Authentication Assertion, in 4
 5722 * authentication extension, in 9
 5723 * AuthenticationExtensionsAuthenticatorInputs
 5724 + definition of, in 5.9
 5725 + (typedef), in 5.9
 5726 * AuthenticationExtensionsClientInputs
 5727 + definition of, in 5.7
 5728 + (dictionary), in 5.7
 5729 * AuthenticationExtensionsClientOutputs
 5730 + definition of, in 5.8
 5731 + (dictionary), in 5.8
 5732 * AuthenticationExtensionsSupported, in 10.5
 5733 * Authenticator, in 4
 5734 * AuthenticatorAssertionResponse, in 5.2.2
 5735 * AuthenticatorAttachment, in 5.4.5
 5736 * authenticatorAttachment, in 5.4.4
 5737 * AuthenticatorAttestationResponse, in 5.2.1
 5738 * authenticatorBiometricPerfBounds, in 10.9
 5739 * authenticatorCancel, in 6.2.4
 5740 * authenticator data, in 6.1
 5741 * authenticatorData, in 5.2.2
 5742 * authenticator data claimed to have been used for the attestation,
 5743 in 6.3.2
 5744 * authenticator data for the attestation, in 6.3.2
 5745 * authenticatorDataResult, in 5.1.4.1
 5746 * authenticator extension, in 9
 5747 * authenticator extension input, in 9.3
 5748 * authenticator extension output, in 9.5
 5749 * Authenticator extension processing, in 9.5
 5750 * authenticatorGetAssertion, in 6.2.3
 5751 * authenticatorMakeCredential, in 6.2.2
 5752 * Authenticator Model, in 6
 5753 * Authenticator operations, in 6.2
 5754 * AuthenticatorResponse, in 5.2
 5755 * authenticatorSelection, in 5.4
 5756 * AuthenticatorSelectionCriteria, in 5.4.4
 5757 * 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
 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

5758 * authenticator session, in 6.2
 5759 * AuthenticatorTransport, in 5.10.4
 5760 * authnSel
 5761 + dict-member for AuthenticationExtensionsClientInputs, in 10.4
 5762 + dict-member for AuthenticationExtensionsClientOutputs, in
 5763 10.4
 5764 * Authorization Gesture, in 4
 5765 * Base64url Encoding, in 3
 5766 * Basic, in 6.3.3
 5767 * Basic Attestation, in 6.3.3
 5768 * Biometric Authenticator, in 4
 5769 * Biometric Recognition, in 4
 5770 * ble, in 5.10.4
 5771 * CBOR, in 3
 5772 * Ceremony, in 4
 5773 * challenge
 5774 + dict-member for PublicKeyCredentialCreationOptions, in 5.4
 5775 + dict-member for PublicKeyCredentialRequestOptions, in 5.5
 5776 + dict-member for CollectedClientData, in 5.10.1
 5777 * Client, in 4
 5778 * client data, in 5.10.1
 5779 * clientDataJSON, in 5.2
 5780 * clientDataJSONResult
 5781 + dfn for credentialCreationData, in 5.1.3
 5782 + dfn for assertionCreationData, in 5.1.4.1
 5783 * client extension, in 9
 5784 * client extension input, in 9.3
 5785 * client extension output, in 9.4
 5786 * Client extension processing, in 9.4
 5787 * clientExtensionResults
 5788 + dfn for credentialCreationData, in 5.1.3
 5789 + dfn for assertionCreationData, in 5.1.4.1

5790 * [[clientExtensionsResults]], in 5.1
 5791 * Client-Side, in 4
 5792 * client-side credential private key storage, in 4
 5793 * Client-side-resident Credential Private Key, in 4
 5794 * CollectedClientData, in 5.10.1
 5795 * [[CollectFromCredentialStore]](origin, options,
 5796 sameOriginWithAncestors), in 5.1.4
 5797 * Conforming User Agent, in 4
 5798 * content, in 10.3
 5799 * contentType, in 10.3
 5800 * COSEAlgorithmIdentifier
 5801 + definition of, in 5.10.5
 5802 + (typedef), in 5.10.5
 5803 * [[Create]](origin, options, sameOriginWithAncestors), in 5.1.3
 5804 * Credential ID, in 4
 5805 * credentialId, in 6.3.1
 5806 * credentialIdLength, in 6.3.1
 5807 * credentialIdResult, in 5.1.4.1
 5808 * credential key pair, in 4
 5809 * credential private key, in 4
 5810 * Credential Public Key, in 4
 5811 * credentialPublicKey, in 6.3.1
 5812 * credentials map, in 6
 5813 * "cross-platform", in 5.4.5
 5814 * cross-platform, in 5.4.5
 5815 * cross-platform attached, in 5.4.5
 5816 * cross-platform attachment, in 5.4.5
 5817 * DAA, in 6.3.3
 5818 * direct, in 5.4.6
 5819 * "discouraged", in 5.10.6
 5820 * discouraged, in 5.10.6
 5821 * [[DiscoverFromExternalSource]](origin, options,
 5822 sameOriginWithAncestors), in 5.1.4.1
 5823 * [[discovery]], in 5.1
 5824 * displayName, in 5.4.3
 5825 * ECDAAs, in 6.3.3
 5826 * 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 * flags, in 6.1

5191 * getClientExtensionResults(), in 5.1
 5192 * [hashAlgorithm](#), in 5.8.1
 5193 * [Hash of the serialized client data](#), in 5.8.1
 5194 * icon, in 5.4.1
 5195 * id

+ dict-member for [PublicKeyCredentialRpEntity](#), in 5.4.2
 + dict-member for [PublicKeyCredentialUserEntity](#), in 5.4.3
 + dict-member for [PublicKeyCredentialDescriptor](#), in 5.8.3

* [\[\[identifier\]\]](#), in 5.1
 * identifier of the ECDAAs-Issuer public key, in 8.2
 * indirect, in 5.4.6
 * [isUserVerifyingPlatformAuthenticatorAvailable\(\)](#), in 5.1.6
 * JSON-serialized client data, in 5.8.1
 * [MakePublicKeyCredentialOptions](#), in 5.4

* managing authenticator, in 4
 * name, in 5.4.1
 * nfc, in 5.8.4

* none, in 5.4.6
 * [origin](#), in 5.8.1

* platform, in 5.4.5
 * "platform", in 5.4.5
 * platform attachment, in 5.4.5
 * platform authenticators, in 5.4.5
 * "preferred", in 5.8.6
 * preferred, in 5.8.6
 * [Privacy CA](#), in 6.3.3

* pubKeyCredParams, in 5.4
 * publicKey
 + dict-member for [CredentialCreationOptions](#), in 5.1.1
 + dict-member for [CredentialRequestOptions](#), in 5.1.2
 * public-key, in 5.8.2
 * Public Key Credential, in 4
 * [PublicKeyCredential](#), in 5.1
 * [PublicKeyCredentialDescriptor](#), in 5.8.3

* [PublicKeyCredentialEntity](#), in 5.4.1
 * [PublicKeyCredentialParameters](#), in 5.3

5827 * effective user verification requirement for assertion, in 5.1.4.1
 5828 * effective user verification requirement for credential creation, in
 5829 5.1.3
 5830 * Elliptic Curve based Direct Anonymous Attestation, in 6.3.3
 5831 * excludeCredentials, in 5.4
 5832 * extension identifier, in 9.1
 5833 * extensions
 5834 + dict-member for [PublicKeyCredentialCreationOptions](#), in 5.4
 5835 + dict-member for [PublicKeyCredentialRequestOptions](#), in 5.5

* exts
 + dict-member for [AuthenticationExtensionsClientInputs](#), in 10.5
 + dict-member for [AuthenticationExtensionsClientOutputs](#), in 10.5

* FAR, in 10.9
 * flags, in 6.1
 * FRR, in 10.9
 * getClientExtensionResults(), in 5.1
 * [Hash of the serialized client data](#), in 5.10.1
 * [Human Palatability](#), in 4
 * icon, in 5.4.1
 * id

+ dfn for [public key credential source](#), in 4
 + dict-member for [PublicKeyCredentialRpEntity](#), in 5.4.2
 + dict-member for [PublicKeyCredentialUserEntity](#), in 5.4.3
 + dict-member for [TokenBinding](#), in 5.10.1
 + dict-member for [PublicKeyCredentialDescriptor](#), in 5.10.3

* [\[\[identifier\]\]](#), in 5.1
 * identifier of the ECDAAs-Issuer public key, in 8.2
 * indirect, in 5.4.6
 * [isUserVerifyingPlatformAuthenticatorAvailable\(\)](#), in 5.1.7
 * JSON-serialized client data, in 5.10.1
 * loc

+ dict-member for [AuthenticationExtensionsClientInputs](#), in 10.7
 + dict-member for [AuthenticationExtensionsClientOutputs](#), in 10.7

* looking up, in 6.2.1
 * managing authenticator, in 4
 * name, in 5.4.1
 * nfc, in 5.10.4
 * [No attestation statement](#), in 6.3.3
 * None, in 6.3.3
 * none, in 5.4.6
 * none attestation statement format, in 8.7

* "not-supported", in 5.10.1
 * not-supported, in 5.10.1
 * origin, in 5.10.1
 * otherUI, in 4

* platform, in 5.4.5
 * "platform", in 5.4.5
 * platform attachment, in 5.4.5
 * platform authenticators, in 5.4.5
 * [platform credential](#), in 5.4.5
 * "preferred", in 5.10.6
 * preferred, in 5.10.6
 * "present", in 5.10.1
 * present, in 5.10.1

* [\[\[preventSilentAccess\]\]\(credential, sameOriginWithAncestors\)](#), in 5.1.6
 * [privateKey](#), in 4
 * pubKeyCredParams, in 5.4
 * publicKey

+ dict-member for [CredentialCreationOptions](#), in 5.1.1
 + dict-member for [CredentialRequestOptions](#), in 5.1.2
 * public-key, in 5.10.2
 * Public Key Credential, in 4
 * [PublicKeyCredential](#), in 5.1
 * [PublicKeyCredentialCreationOptions](#), in 5.4
 * [PublicKeyCredentialDescriptor](#), in 5.10.3

* [PublicKeyCredentialEntity](#), in 5.4.1
 * [PublicKeyCredentialParameters](#), in 5.3

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
 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

5897 * PublicKeyCredentialRequestOptions, in 5.5
 5898 * PublicKeyCredentialRpEntity, in 5.4.2
 5899 * Public Key Credential Source, in 4
 5900 * PublicKeyCredentialType, in 5.10.2
 5901 * PublicKeyCredentialUserEntity, in 5.4.3
 5902 * Rate Limiting, in 4
 5903 * rawId, in 5.1
 5904 * Registration, in 4
 5905 * registration extension, in 9
 5906 * Relying Party, in 4
 5907 * Relying Party Identifier, in 4
 5908 * "required", in 5.10.6
 5909 * required, in 5.10.6
 5910 * requireResidentKey, in 5.4.4
 5911 * response, in 5.1
 5912 * roaming authenticators, in 5.4.5
 5913 * roaming credential, in 5.4.5
 5914 * rp, in 5.4
 5915 * rpId
 5916 + dfn for public key credential source, in 4
 5917 + dict-member for PublicKeyCredentialRequestOptions, in 5.5
 5918 * RP ID, in 4
 5919 * rpIdHash, in 6.1
 5920 * Self, in 6.3.3
 5921 * Self Attestation, in 6.3.3
 5922 * signature, in 5.2.2
 5923 * Signature Counter, in 6.1.1
 5924 * signatureResult, in 5.1.4.1
 5925 * signCount, in 6.1
 5926 * Signing procedure, in 6.3.2
 5927 * status, in 5.10.1
 5928 * [[Store]](credential, sameOriginWithAncestors), in 5.1.5
 5929 * supported, in 5.10.1
 5930 * "supported", in 5.10.1
 5931 * Test of User Presence, in 4
 5932 * timeout
 5933 + dict-member for PublicKeyCredentialCreationOptions, in 5.4
 5934 + dict-member for PublicKeyCredentialRequestOptions, in 5.5
 5935 * tokenBinding, in 5.10.1
 5936 * TokenBinding, in 5.10.1
 5937 * TokenBindingStatus, in 5.10.1
 5938 * transports, in 5.10.3
 5939 * txAuthGeneric
 5940 + dict-member for AuthenticationExtensionsClientInputs, in 10.3
 5941 + dict-member for AuthenticationExtensionsClientOutputs, in 10.3
 5942 * txAuthGenericArg, in 10.3
 5943 * txAuthSimple
 5944 + dict-member for AuthenticationExtensionsClientInputs, in 10.2
 5945 + dict-member for AuthenticationExtensionsClientOutputs, in 10.2
 5946 * [[type]], in 5.1
 5947 * type
 5948 + dfn for public key credential source, in 4
 5949 + dict-member for PublicKeyCredentialParameters, in 5.3
 5950 + dict-member for CollectedClientData, in 5.10.1
 5951 + dict-member for PublicKeyCredentialDescriptor, in 5.10.3
 5952 * UP, in 4
 5953 * usb, in 5.10.4
 5954 * user, in 5.4
 5955 * User Consent, in 4
 5956 * userHandle
 5957 + dfn for public key credential source, in 4
 5958 + attribute for AuthenticatorAssertionResponse, in 5.2.2
 5959 * User Handle, in 4
 5960 * userHandleResult, in 5.1.4.1
 5961 * User Present, in 4
 5962 * userVerification
 5963 + dict-member for AuthenticatorSelectionCriteria, in 5.4.4
 5964 + 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

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

- + Credential
- + CredentialCreationOptions
- + CredentialRequestOptions
- + CredentialsContainer
- + Request a Credential
- + [[CollectFromCredentialStore]](origin, options, sameOriginWithAncestors)
- + [[Create]](origin, options, sameOriginWithAncestors)
- + [[Store]](credential, sameOriginWithAncestors)
- + [[discovery]]
- + [[type]]
- + create()
- + credential
- + credential source
- + get()
- + id
- + remote
- + same-origin with its ancestors
- + signal (for CredentialCreationOptions)
- + signal (for CredentialRequestOptions)
- + store()
- + type
- + user mediation

* [DOM4] defines the following terms:

- + AbortController
- + aborted flag
- + document

* [ECMAScript] defines the following terms:

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

* [ENCODING] defines the following terms:

- + utf-8 encode

* [FETCH] defines the following terms:

- + window

* [HTML] defines the following terms:

- + ascii serialization of an origin
- + effective domain
- + environment settings object

5324
5325
5326
5327

- 5967 * User Verification, in 4
- 5968 * UserVerificationRequirement, in 5.10.6
- 5969 * User Verified, in 4
- 5970 * UV, in 4
- 5971 * uvi

- + dict-member for AuthenticationExtensionsClientInputs, in 10.6
- + dict-member for AuthenticationExtensionsClientOutputs, in 10.6

* uvm

- + dict-member for AuthenticationExtensionsClientInputs, in 10.8
- + dict-member for AuthenticationExtensionsClientOutputs, in 10.8

* UvmEntries, in 10.8

* UvmEntry, in 10.8

- * Verification procedure, in 6.3.2
- * verification procedure inputs, in 6.3.2
- * Web Authentication API, in 5
- * WebAuthn Client, in 4

Terms defined by reference

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

- + Credential
- + CredentialCreationOptions
- + CredentialRequestOptions
- + CredentialsContainer
- + Request a Credential
- + [[CollectFromCredentialStore]](origin, options, sameOriginWithAncestors)
- + [[Create]](origin, options, sameOriginWithAncestors)
- + [[Store]](credential, sameOriginWithAncestors)
- + [[discovery]]
- + [[type]]
- + create()
- + credential
- + credential source
- + get()
- + id
- + remote
- + same-origin with its ancestors
- + signal (for CredentialCreationOptions)
- + signal (for CredentialRequestOptions)
- + store()
- + type
- + user mediation

* [DOM4] defines the following terms:

- + AbortController
- + aborted flag
- + document

* [ECMAScript] defines the following terms:

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

* [ENCODING] defines the following terms:

- + utf-8 decode

- + utf-8 encode

* [FETCH] defines the following terms:

- + window

* [FIDO-APPID] defines the following terms:

- + determining if a caller's facetid is authorized for an appid
- + determining the facetid of a calling application

* [FIDO-CTAP] defines the following terms:

- + ctap2 canonical cbor encoding form

* [Geolocation-API] defines the following terms:

- + Coordinates

* [HTML] defines the following terms:

- + ascii serialization of an origin
- + effective domain
- + environment settings object

5981
5982
5983
5984
5985
5986
5987
5988
5989
5990
5991
5992
5993
5994
5995
5996
5997
5998
5999
6000
6001
6002
6003
6004
6005
6006
6007
6008
6009
6010
6011
6012
6013
6014
6015
6016
6017
6018
6019
6020
6021
6022
6023
6024
6025
6026
6027
6028
6029
6030
6031
6032
6033
6034
6035
6036

532f + global object
532e + is a registrable domain suffix of or is equal to
533c + 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
533e + opaque origin
5336 + origin
5337 * [INFRA] defines the following terms:
533e + append (for list)
533e + append (for set)
534c + byte sequence
5341 + continue
5342 + empty
5343 + for each (for list)
5344 + for each (for map)
534e + is empty
534e + is not empty
5347 + item (for list)
534e + item (for struct)
534e + list
535c + map
5351 + ordered set
5352 + remove
5353 + set
5354 + size
535e + struct
535e + while
5357 + willful violation
535e * [mixed-content] defines the following terms:
535e + a priori authenticated url
536c * [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:
536e + token binding
536e + token binding id
5367 * [URL] defines the following terms:
536e + domain
536e + empty host
537c + host
5371 + ipv4 address
5372 + ipv6 address
5373 + opaque host
5374 + url serializer
537e + valid domain
537e + valid domain string
5377 * [WebCryptoAPI] defines the following terms:
537e + recognized algorithm name
537e * [WebIDL] defines the following terms:
538c + AbortError
5381 + ArrayBuffer
5382 + BufferSource
5383 + ConstraintError
5384 + DOMException
538e + DOMString
538e + Exposed
5387 + NotAllowedError
538e + NotSupportedError
538e + Promise
539c + SameObject
5391 + SecureContext
5392 + SecurityError
5393 + USVString
5394 + UnknownError
539e + boolean

6037 + global object
603e + is a registrable domain suffix of or is equal to
603e + is not a registrable domain suffix of and is not equal to
604c + origin
6041 + relevant settings object
6042 * [HTML52] defines the following terms:
6043 + document.domain
6044 + opaque origin
604e + origin
604e * [INFRA] defines the following terms:
6047 + append (for list)
604e + append (for set)
604e + byte sequence
605c + continue
6051 + for each (for list)
6052 + for each (for map)
6053 + is empty
6054 + is not empty
605e + item (for list)
605e + item (for struct)
6057 + list
605e + map
605e + ordered set
606c + remove
6061 + set
6062 + set (for map)
6063 + struct
6064 + while
606e + willful violation
606e * [mixed-content] defines the following terms:
606e + a priori authenticated url
606e * [page-visibility] defines the following terms:
606e + visibility states
607c * [secure-contexts] defines the following terms:
6071 + secure contexts
6072 * [TokenBinding] defines the following terms:
6073 + token binding
6074 + token binding id
607e * [URL] defines the following terms:
607e + domain
6077 + empty host
607e + host
607e + ipv4 address
608c + ipv6 address
6081 + opaque host
6082 + url serializer
6083 + valid domain
6084 + valid domain string
608e * [WebIDL] defines the following terms:
608e + AbortError
6087 + ArrayBuffer
608e + BufferSource
608e + ConstraintError
609c + DOMException
6091 + DOMString
6092 + Exposed
6093 + InvalidStateError
6094 + NotAllowedError
609e + NotSupportedError
609e + Promise
6097 + SameObject
609e + SecureContext
609e + SecurityError
610c + USVString
6101 + UnknownError
6102 + boolean
6103 + float

5396 + interface object
5397 + long
5398 + present
5399 + unsigned long
5400 * [whatwg html] defines the following terms:
5401 + focus

5402 References

5403 Normative References

5404 [CDDL]
5405 C. Vigano; H. Birkholz. CBOR data definition language (CDDL): a
5406 notational convention to express CBOR data structures. 21
5407 September 2016. Internet Draft (work in progress). URL:
5408 <https://tools.ietf.org/html/draft-greevenbosch-appsawg-cbor-cddl>

5409 [CREDENTIAL-MANAGEMENT-1]
5410 Mike West. Credential Management Level 1. 4 August 2017. WD.
5411 URL: <https://www.w3.org/TR/credential-management-1/>

5412 [DOM4]
5413 Anne van Kesteren. DOM Standard. Living Standard. URL:
5414 <https://dom.spec.whatwg.org/>

5415 [ECMAScript]
5416 ECMAScript Language Specification. URL:
5417 <https://tc39.github.io/ecma262/>

5418 [ENCODING]
5419 Anne van Kesteren. Encoding Standard. Living Standard. URL:
5420 <https://encoding.spec.whatwg.org/>

5421 [FETCH]
5422 Anne van Kesteren. Fetch Standard. Living Standard. URL:
5423 <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](#). URL:
5436 <https://fidoalliance.org/specs/fido-v2.0-rd-20170927/fido-client-to-authenticator-protocol-v2.0-rd-20170927.html>

5438 [FIDO-U2F-Message-Formats]
5439 D. Balfanz; J. Ehrensvar; 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-u2f-raw-message-formats-v1.1-id-20160915.html>

5442 [FIDOEcdaaAlgorithm]
5443 R. Lindemann; et al. FIDO ECDA A Algorithm. FIDO Alliance
5444 Implementation Draft. URL:
5445 <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ecdaa-algorithm-v1.1-id-20170202.html>

5451 [FIDOReg]
5452 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO UAF Registry of

6104 + interface object
6105 + long
6106 + present
6107 + unsigned long
6108 * [whatwg html] defines the following terms:
6109 + focus
6110 + username

6111 References

6112 Normative References

6113 [CDDL]
6114 C. Vigano; H. Birkholz. CBOR data definition language (CDDL): a
6115 notational convention to express CBOR data structures. 21
6116 September 2016. Internet Draft (work in progress). URL:
6117 <https://tools.ietf.org/html/draft-greevenbosch-appsawg-cbor-cddl>

6118 [CREDENTIAL-MANAGEMENT-1]
6119 Mike West. Credential Management Level 1. 4 August 2017. WD.
6120 URL: <https://www.w3.org/TR/credential-management-1/>

6121 [DOM4]
6122 Anne van Kesteren. DOM Standard. Living Standard. URL:
6123 <https://dom.spec.whatwg.org/>

6124 [ECMAScript]
6125 ECMAScript Language Specification. URL:
6126 <https://tc39.github.io/ecma262/>

6127 [ENCODING]
6128 Anne van Kesteren. Encoding Standard. Living Standard. URL:
6129 <https://encoding.spec.whatwg.org/>

6130 [FETCH]
6131 Anne van Kesteren. Fetch Standard. Living Standard. URL:
6132 <https://fetch.spec.whatwg.org/>

6133 [FIDO-APPID]
6134 D. Balfanz; et al. FIDO AppID and Facets. FIDO Alliance Proposed
6135 Standard. URL:
6136 <https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-appid-and-facets-v2.0-ps-20170927.html>

6137 [FIDO-CTAP]
6138 R. Lindemann; et al. FIDO 2.0: Client to Authenticator Protocol.
6139 FIDO Alliance [Proposed Standard](#). URL:
6140 <https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-client-to-authenticator-protocol-v2.0-ps-20170927.html>

6141 [FIDO-Privacy-Principles]
6142 FIDO Alliance. FIDO Privacy Principles. FIDO Alliance
6143 [Whitepaper](#). URL:
6144 https://fidoalliance.org/wp-content/uploads/2014/12/FIDO_Alliance_Whitepaper_Privacy_Principles.pdf

6145 [FIDO-U2F-Message-Formats]
6146 D. Balfanz; J. Ehrensvar; J. Lang. FIDO U2F Raw Message
6147 Formats. FIDO Alliance Implementation Draft. URL:
6148 <https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-raw-message-formats-v1.1-id-20160915.html>

6149 [FIDOEcdaaAlgorithm]
6150 R. Lindemann; et al. FIDO ECDA A Algorithm. FIDO Alliance
6151 Implementation Draft. URL:
6152 <https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-ecdaa-algorithm-v1.1-id-20170202.html>

6153 [FIDOReg]
6154 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO UAF Registry of

5455 Predefined Values. FIDO Alliance Proposed Standard. URL:
5454 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua>
5455 f-reg-v1.0-ps-20141208.html
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]
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>

6174 Predefined Values. FIDO Alliance Proposed Standard. URL:
6175 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua>
6176 f-reg-v1.0-ps-20141208.html
6177

[Geolocation-API]
6178 Andrei Popescu. Geolocation API Specification 2nd Edition. 8
6179 November 2016. REC. URL: <https://www.w3.org/TR/geolocation-API/>
6180

6181 [HTML]
6182 Anne van Kesteren; et al. HTML Standard. Living Standard. URL:
6183 <https://html.spec.whatwg.org/multipage/>
6184

6185 [HTML52]
6186 Steve Faulkner; et al. HTML 5.2. 14 December 2017. REC. URL:
6187 <https://www.w3.org/TR/html52/>
6188

6189 [IANA-COSE-ALGS-REG]
6190 IANA CBOR Object Signing and Encryption (COSE) Algorithms
6191 Registry. URL:
6192 <https://www.iana.org/assignments/cose/cose.xhtml#algorithms>
6193

6194 [INFRA]
6195 Anne van Kesteren; Domenic Denicola. Infra Standard. Living
6196 Standard. URL: <https://infra.spec.whatwg.org/>
6197

6198 [MIXED-CONTENT]
6199 Mike West. Mixed Content. 2 August 2016. CR. URL:
6200 <https://www.w3.org/TR/mixed-content/>
6201

6202 [PAGE-VISIBILITY]
6203 Jatinder Mann; Arvind Jain. Page Visibility (Second Edition). 29
6204 October 2013. REC. URL: <https://www.w3.org/TR/page-visibility/>
6205

6206 [RFC2119]
6207 S. Bradner. Key words for use in RFCs to Indicate Requirement
6208 Levels. March 1997. Best Current Practice. URL:
6209 <https://tools.ietf.org/html/rfc2119>
6210

6211 [RFC4648]
6212 S. Josefsson. The Base16, Base32, and Base64 Data Encodings.
6213 October 2006. Proposed Standard. URL:
6214 <https://tools.ietf.org/html/rfc4648>
6215

6216 [RFC5234]
6217 D. Crocker, Ed.; P. Overell. Augmented BNF for Syntax
6218 Specifications: ABNF. January 2008. Internet Standard. URL:
6219 <https://tools.ietf.org/html/rfc5234>
6220

6221 [RFC5890]
6222 J. Klensin. Internationalized Domain Names for Applications
6223 (IDNA): Definitions and Document Framework. August 2010.
6224 Proposed Standard. URL: <https://tools.ietf.org/html/rfc5890>
6225

6226 [RFC7049]
6227 C. Bormann; P. Hoffman. Concise Binary Object Representation
6228 (CBOR). October 2013. Proposed Standard. URL:
6229 <https://tools.ietf.org/html/rfc7049>
6230

6231 [RFC8152]
6232 J. Schaad. CBOR Object Signing and Encryption (COSE). July 2017.
6233 Proposed Standard. URL: <https://tools.ietf.org/html/rfc8152>
6234

6235 [RFC8230]
6236 M. Jones. Using RSA Algorithms with CBOR Object Signing and
6237 Encryption (COSE) Messages. September 2017. Proposed Standard.
6238 URL: <https://tools.ietf.org/html/rfc8230>
6239

6240 [SEC1]
6241 SEC1: Elliptic Curve Cryptography, Version 2.0. URL:
6242 <http://www.secg.org/sec1-v2.pdf>
6243

5514
5515
5516
5517
5518

[SECURE-CONTEXTS]

Mike West. Secure Contexts. 15 September 2016. CR. URL:
<https://www.w3.org/TR/secure-contexts/>

[TokenBinding]

A. Popov; et al. The Token Binding Protocol Version 1.0.
February 16, 2017. Internet-Draft. URL:
<https://tools.ietf.org/html/draft-ietf-tokbind-protocol>

[URL]

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

[WebAuthn-Registries]

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

[WebCryptoAPI]

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

[WebIDL]

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

[WebIDL-1]

Cameron McCormack. WebIDL Level 1. 15 December 2016. REC. URL:
<https://www.w3.org/TR/2016/REC-WebIDL-1-20161215/>

Informative References

[Ceremony]

Carl Ellison. Ceremony Design and Analysis. 2007. URL:
<https://eprint.iacr.org/2007/399.pdf>

[Feature-Policy]

Feature Policy. Draft Community Group Report. URL:
<https://wicg.github.io/feature-policy/>

[FIDO-APPID]

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

[FIDO-UAF-AUTHNR-CMDS]

R. Lindemann; J. Kemp. FIDO UAF Authenticator Commands. FIDO
Alliance Implementation Draft. URL:
<https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-authnr-cmds-v1.1-id-20170202.html>

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
5551
5552
5553

5554
5555
5556
5557

5558
5559
5560
5561
5562

5563
5564
5565
5566
5567
5568
5569

6244
6245
6246
6247
6248
6249

[SECURE-CONTEXTS]

Mike West. Secure Contexts. 15 September 2016. CR. URL:
<https://www.w3.org/TR/secure-contexts/>

[TCG-CMCPProfile-AIKCertEnroll]

Scott Kelly; et al. TCG Infrastructure Working Group: A CMC
Profile for AIK Certificate Enrollment. 24 March 2011.
Published. URL:
https://trustedcomputinggroup.org/wp-content/uploads/IWG_CMC_Profile_Cert_Enrollment_v1_r7.pdf

[TokenBinding]

A. Popov; et al. The Token Binding Protocol Version 1.0.
February 16, 2017. Internet-Draft. URL:
<https://tools.ietf.org/html/draft-ietf-tokbind-protocol>

[URL]

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

[WebAuthn-Registries]

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

[WebIDL]

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

[WebIDL-1]

Cameron McCormack. WebIDL Level 1. 15 December 2016. REC. URL:
<https://www.w3.org/TR/2016/REC-WebIDL-1-20161215/>

Informative References

[Ceremony]

Carl Ellison. Ceremony Design and Analysis. 2007. URL:
<https://eprint.iacr.org/2007/399.pdf>

[EduPersonObjectClassSpec]

EduPerson Object Class Specification (200604a). May 15, 2007.
URL:
<https://www.internet2.edu/media/medialibrary/2013/09/04/internet2-mace-dir-eduperson-200604.html>

[Feature-Policy]

Feature Policy. Draft Community Group Report. URL:
<https://wicg.github.io/feature-policy/>

[FIDO-UAF-AUTHNR-CMDS]

R. Lindemann; J. Kemp. FIDO UAF Authenticator Commands. FIDO
Alliance Implementation Draft. URL:
<https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-authnr-cmds-v1.1-id-20170202.html>

[FIDOAuthnrSecReqs]

6250
6251
6252
6253
6254
6255
6256
6257
6258
6259
6260
6261
6262
6263
6264
6265
6266
6267
6268
6269
6270
6271
6272

6273
6274
6275
6276
6277
6278
6279
6280
6281
6282
6283
6284
6285
6286
6287

6288
6289
6290
6291
6292
6293
6294
6295
6296

6297
6298
6299
6300
6301
6302
6303

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-ua>
5574 <f-metadata-service-v1.0-ps-20141208.html>
5575
5576 [FIDOSecRef]
5577 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO Security Reference.
5578 FIDO Alliance Proposed Standard. URL:
5579 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-se>
5580 <curity-ref-v1.0-ps-20141208.html>
5581
5582 [GeoJSON]
5583 The GeoJSON Format Specification. URL:
5584 <http://geojson.org/geojson-spec.html>

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>
5590 _ISOIEC_2382-37_2012.zip
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 [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]

6304 D. Biggs; et al. FIDO Authenticator Security Requirements. FIDO
6305 Alliance Final Documents. URL:
6306 <https://fidoalliance.org/specs/fido-security-requirements-v1.0-f>
6307 <d-20170524/>
6308
6309 [FIDOMetadataService]
6310 R. Lindemann; B. Hill; D. Baghdasaryan. FIDO Metadata Service
6311 v1.0. FIDO Alliance Proposed Standard. URL:
6312 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua>
6313 <f-metadata-service-v1.0-ps-20141208.html>
6314
6315 [FIDOSecRef]
6316 R. Lindemann; D. Baghdasaryan; B. Hill. FIDO Security Reference.
6317 FIDO Alliance Proposed Standard. URL:
6318 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20170411/fido-se>
6319 <curity-ref-v1.2-ps-20170411.html>
6320
6321 [FIDOUAFAuthenticatorMetadataStatements]
6322 B. Hill; D. Baghdasaryan; J. Kemp. FIDO UAF Authenticator
6323 Metadata Statements v1.0. FIDO Alliance Proposed Standard. URL:
6324 <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua>
6325 <f-authnr-metadata-v1.0-ps-20141208.html>
6326
6327 [ISOBiometricVocabulary]
6328 ISO/IEC JTC1/SC37. Information technology -- Vocabulary --
6329 Biometrics. 15 December 2012. International Standard: ISO/IEC
6330 2382-37:2012(E) First Edition. URL:
6331 <http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194>
6332 _ISOIEC_2382-37_2012.zip
6333
6334 [RFC3279]
6335 L. Bassham; W. Polk; R. Housley. Algorithms and Identifiers for
6336 the Internet X.509 Public Key Infrastructure Certificate and
6337 Certificate Revocation List (CRL) Profile. April 2002. Proposed
6338 Standard. URL: <https://tools.ietf.org/html/rfc3279>
6339
6340 [RFC4949]
6341 R. Shirey. Internet Security Glossary, Version 2. August 2007.
6342 Informational. URL: <https://tools.ietf.org/html/rfc4949>
6343
6344 [RFC5280]
6345 D. Cooper; et al. Internet X.509 Public Key Infrastructure
6346 Certificate and Certificate Revocation List (CRL) Profile. May
6347 2008. Proposed Standard. URL:
6348 <https://tools.ietf.org/html/rfc5280>
6349
6350 [RFC6265]
6351 A. Barth. HTTP State Management Mechanism. April 2011. Proposed
6352 Standard. URL: <https://tools.ietf.org/html/rfc6265>
6353
6354 [RFC6454]
6355 A. Barth. The Web Origin Concept. December 2011. Proposed
6356 Standard. URL: <https://tools.ietf.org/html/rfc6454>
6357
6358 [RFC7515]
6359 M. Jones; J. Bradley; N. Sakimura. JSON Web Signature (JWS). May
6360 2015. Proposed Standard. URL:
6361 <https://tools.ietf.org/html/rfc7515>
6362
6363 [RFC8017]
6364 K. Moriarty, Ed.; et al. PKCS #1: RSA Cryptography
6365 Specifications Version 2.2. November 2016. Informational. URL:
6366 <https://tools.ietf.org/html/rfc8017>
6367
6368 [TPMv2-EK-Profile]
6369 TCG EK Credential Profile for TPM Family 2.0. URL:
6370 <http://www.trustedcomputinggroup.org/wp-content/uploads/Credenti>
6371 al_Profile_EK_V2.0_R14_published.pdf
6372
6373 [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;
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

```

```

6374 Trusted Platform Module Library, Part 1: Architecture. URL:
6375 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
6376 2.0-Part-1-Architecture-01.38.pdf
6377
6378 [TPMv2-Part2]
6379 Trusted Platform Module Library, Part 2: Structures. URL:
6380 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
6381 2.0-Part-2-Structures-01.38.pdf
6382
6383 [TPMv2-Part3]
6384 Trusted Platform Module Library, Part 3: Commands. URL:
6385 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
6386 2.0-Part-3-Commands-01.38.pdf
6387
6388 [UAFProtocol]
6389 R. Lindemann; et al. FIDO UAF Protocol Specification v1.0. FIDO
6390 Alliance Proposed Standard. URL:
6391 https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua
6392 f-protocol-v1.0-ps-20141208.html
6393
6394 IDL Index
6395
6396 [SecureContext, Exposed=Window]
6397 interface PublicKeyCredential : Credential {
6398   [SameObject] readonly attribute ArrayBuffer rawId;
6399   [SameObject] readonly attribute AuthenticatorResponse response;
6400   AuthenticationExtensionsClientOutputs getClientExtensionResults();
6401 };
6402
6403 partial dictionary CredentialCreationOptions {
6404   PublicKeyCredentialCreationOptions publicKey;
6405 };
6406
6407 partial dictionary CredentialRequestOptions {
6408   PublicKeyCredentialRequestOptions publicKey;
6409 };
6410
6411 partial interface PublicKeyCredential {
6412   static Promise < boolean > isUserVerifyingPlatformAuthenticatorAvailable();
6413 };
6414
6415 [SecureContext, Exposed=Window]
6416 interface AuthenticatorResponse {
6417   [SameObject] readonly attribute ArrayBuffer clientDataJSON;
6418 };
6419
6420 [SecureContext, Exposed=Window]
6421 interface AuthenticatorAttestationResponse : AuthenticatorResponse {
6422   [SameObject] readonly attribute ArrayBuffer attestationObject;
6423 };
6424
6425 [SecureContext, Exposed=Window]
6426 interface AuthenticatorAssertionResponse : AuthenticatorResponse {
6427   [SameObject] readonly attribute ArrayBuffer authenticatorData;
6428   [SameObject] readonly attribute ArrayBuffer signature;
6429   [SameObject] readonly attribute ArrayBuffer userHandle;
6430 };
6431
6432 dictionary PublicKeyCredentialParameters {
6433   required PublicKeyCredentialType type;
6434   required COSEAlgorithmIdentifier alg;
6435 };
6436
6437 dictionary PublicKeyCredentialCreationOptions {
6438   required PublicKeyCredentialRpEntity rp;
6439   required PublicKeyCredentialUserEntity user;
6440
6441   required BufferSource challenge;
6442   required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
6443

```

```

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;

```

```

5745
5746 dictionary CollectedClientData {
5747   required DOMString type;
5748   required DOMString challenge;
5749   required DOMString origin;
5750   required DOMString hashAlgorithm;
5751   DOMString tokenBindingId;
5752   AuthenticationExtensions clientExtensions;
5753   AuthenticationExtensions authenticatorExtensions;
5754 };
5755

```

```

6444 unsigned long timeout;
6445 sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
6446 AuthenticatorSelectionCriteria authenticatorSelection;
6447 AttestationConveyancePreference attestation = "none";
6448 AuthenticationExtensionsClientInputs extensions;
6449 };
6450
6451 dictionary PublicKeyCredentialEntity {
6452   required DOMString name;
6453   USVString icon;
6454 };
6455
6456 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
6457   DOMString id;
6458 };
6459
6460 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
6461   required BufferSource id;
6462   required DOMString displayName;
6463 };
6464
6465 dictionary AuthenticatorSelectionCriteria {
6466   AuthenticatorAttachment authenticatorAttachment;
6467   boolean requireResidentKey = false;
6468   UserVerificationRequirement userVerification = "preferred";
6469 };
6470
6471 enum AuthenticatorAttachment {
6472   "platform", // Platform attachment
6473   "cross-platform" // Cross-platform attachment
6474 };
6475
6476 enum AttestationConveyancePreference {
6477   "none",
6478   "indirect",
6479   "direct"
6480 };
6481
6482 dictionary PublicKeyCredentialRequestOptions {
6483   required BufferSource challenge;
6484   unsigned long timeout;
6485   USVString rpId;
6486   sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
6487   UserVerificationRequirement userVerification = "preferred";
6488   AuthenticationExtensionsClientInputs extensions;
6489 };
6490
6491 dictionary AuthenticationExtensionsClientInputs {
6492 };
6493
6494 dictionary AuthenticationExtensionsClientOutputs {
6495 };
6496
6497 typedef record<DOMString, DOMString> AuthenticationExtensionsAuthenticatorInputs;
6498
6499
6500 dictionary CollectedClientData {
6501   required DOMString type;
6502   required DOMString challenge;
6503   required DOMString origin;
6504   TokenBinding tokenBinding;
6505 };
6506
6507 dictionary TokenBinding {
6508   required TokenBindingStatus status;
6509   DOMString id;
6510 };
6511
6512 enum TokenBindingStatus { "present", "supported", "not-supported" };
6513

```

```

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
5780
5781
5782
5783
5784
5785
5786
5787
5788
5789
5790
5791
5792
5793
5794
5795
5796
5797
5798
5799
5800
5801
5802
5803
5804
5805
5806
5807
5808
5809
5810
5811
5812
5813
5814
5815
5816
5817
5818
5819
5820
5821
5822
5823
5824
5825
5826
5827
5828
5829
5830
5831
5832
5833
5834
5835
5836
5837
5838
5839
5840
5841
5842
5843
5844
5845
5846
5847
5848
5849
5850
5851
5852
5853
5854
5855
5856
5857
5858
5859
5860
5861
5862
5863
5864
5865
5866
5867
5868
5869
5870
5871
5872
5873
5874
5875
5876
5877
5878
5879
5880
5881
5882
5883
5884
5885
5886
5887
5888
5889
5890
5891
5892
5893
5894
5895
5896
5897
5898
5899
5900
5901
5902
5903
5904
5905
5906
5907
5908
5909
5910
5911
5912
5913
5914
5915
5916
5917
5918
5919
5920
5921
5922
5923
5924
5925
5926
5927
5928
5929
5930
5931
5932
5933
5934
5935
5936
5937
5938
5939
5940
5941
5942
5943
5944
5945
5946
5947
5948
5949
5950
5951
5952
5953
5954
5955
5956
5957
5958
5959
5960
5961
5962
5963
5964
5965
5966
5967
5968
5969
5970
5971
5972
5973
5974
5975
5976
5977
5978
5979
5980
5981
5982
5983
5984
5985
5986
5987
5988
5989
5990
5991
5992
5993
5994
5995
5996
5997
5998
5999
6000

```

```

6514 enum PublicKeyCredentialType {
6515     "public-key"
6516 };
6517
6518 dictionary PublicKeyCredentialDescriptor {
6519     required PublicKeyCredentialType type;
6520     required BufferSource id;
6521     sequence<AuthenticatorTransport> transports;
6522 };
6523
6524 enum AuthenticatorTransport {
6525     "usb",
6526     "nfc",
6527     "ble"
6528 };
6529
6530 typedef long COSEAlgorithmIdentifier;
6531
6532 enum UserVerificationRequirement {
6533     "required",
6534     "preferred",
6535     "discouraged"
6536 };
6537
6538 partial dictionary AuthenticationExtensionsClientInputs {
6539     USVString appId;
6540 };
6541
6542 partial dictionary AuthenticationExtensionsClientOutputs {
6543     boolean appId;
6544 };
6545
6546 partial dictionary AuthenticationExtensionsClientInputs {
6547     USVString txAuthSimple;
6548 };
6549
6550 partial dictionary AuthenticationExtensionsClientOutputs {
6551     USVString txAuthSimple;
6552 };
6553
6554 dictionary txAuthGenericArg {
6555     required USVString contentType; // MIME-Type of the content, e.g., "image
6556     /png"
6557     required ArrayBuffer content;
6558 };
6559
6560 partial dictionary AuthenticationExtensionsClientInputs {
6561     txAuthGenericArg txAuthGeneric;
6562 };
6563
6564 partial dictionary AuthenticationExtensionsClientOutputs {
6565     ArrayBuffer txAuthGeneric;
6566 };
6567
6568 typedef sequence<AAGUID> AuthenticatorSelectionList;
6569
6570
6571
6572
6573
6574
6575
6576
6577
6578
6579
6580
6581
6582
6583
6584
6585
6586
6587
6588
6589
6590
6591
6592
6593
6594
6595
6596
6597
6598
6599
6600
6601
6602
6603
6604
6605
6606
6607
6608
6609
6610
6611
6612
6613
6614
6615
6616
6617
6618
6619
6620
6621
6622
6623
6624
6625
6626
6627
6628
6629
6630
6631
6632
6633
6634
6635
6636
6637
6638
6639
6640
6641
6642
6643
6644
6645
6646
6647
6648
6649
6650
6651
6652
6653
6654
6655
6656
6657
6658
6659
6660
6661
6662
6663
6664
6665
6666
6667
6668
6669
6670
6671
6672
6673
6674
6675
6676
6677
6678
6679
6680
6681
6682
6683
6684
6685
6686
6687
6688
6689
6690
6691
6692
6693
6694
6695
6696
6697
6698
6699
6700
6701
6702
6703
6704
6705
6706
6707
6708
6709
6710
6711
6712
6713
6714
6715
6716
6717
6718
6719
6720
6721
6722
6723
6724
6725
6726
6727
6728
6729
6730
6731
6732
6733
6734
6735
6736
6737
6738
6739
6740
6741
6742
6743
6744
6745
6746
6747
6748
6749
6750
6751
6752
6753
6754
6755
6756
6757
6758
6759
6760
6761
6762
6763
6764
6765
6766
6767
6768
6769
6770
6771
6772
6773
6774
6775
6776
6777
6778
6779
6780
6781
6782
6783
6784
6785
6786
6787
6788
6789
6790
6791
6792
6793
6794
6795
6796
6797
6798
6799
6800

```

5784
5785
5786
5787
5788
5789
5790
5791
5792
5793
5794
5795
5796
5797
5798
5799
5800
5801
5802
5803
5804
5805
5806
5807
5808
5809
5810
5811
5812
5813
5814
5815

Issues Index

The definitions of "lifetime of" and "becomes available" are intended to represent how devices are hotplugged into (USB) or discovered by (NFC) browsers, and are under-specified. Resolving this with good definitions or some other means will be addressed by resolving Issue #613. RET

need to define "blinding". See also #462.
<https://github.com/w3c/webauthn/issues/694> RET

@balfanz wishes to add to the "direct" case: If the authenticator violates the privacy requirements of the attestation type it is using, the client SHOULD terminate this algorithm with a "AttestationNotPrivateError". RET

The definitions of "lifetime of" and "becomes available" are intended to represent how devices are hotplugged into (USB) or discovered by (NFC) browsers, and are under-specified. Resolving this with good definitions or some other means will be addressed by resolving Issue #613. RET

The foregoing step `may` be incorrect, in that we are attempting to create `savedCredentialId` here and use it later below, and we do not have a global in which to allocate a place for it. Perhaps this is good enough? addendum: @jcjones feels the above step is likely good enough. RET

The WHATWG HTML WG is discussing whether to provide a hook when a browsing context gains or loses focuses. If a hook is provided, the above paragraph will be updated to include the hook. See WHATWG HTML WG Issue #2711 for more details. RET

#base64url-encodingReferenced in:
* 5.1. PublicKeyCredential Interface
* 5.1.3. Create a new credential - PublicKeyCredential's

```

6584 typedef sequence<USVString> AuthenticationExtensionsSupported;
6585
6586 partial dictionary AuthenticationExtensionsClientOutputs {
6587   AuthenticationExtensionsSupported exts;
6588 };
6589
6590 partial dictionary AuthenticationExtensionsClientInputs {
6591   boolean uvi;
6592 };
6593
6594 partial dictionary AuthenticationExtensionsClientOutputs {
6595   ArrayBuffer uvi;
6596 };
6597
6598 partial dictionary AuthenticationExtensionsClientInputs {
6599   boolean loc;
6600 };
6601
6602 partial dictionary AuthenticationExtensionsClientOutputs {
6603   Coordinates loc;
6604 };
6605
6606 partial dictionary AuthenticationExtensionsClientInputs {
6607   boolean uvm;
6608 };
6609
6610 typedef sequence<unsigned long> UvmEntry;
6611 typedef sequence<UvmEntry> UvmEntries;
6612
6613 partial dictionary AuthenticationExtensionsClientOutputs {
6614   UvmEntries uvm;
6615 };
6616
6617 dictionary authenticatorBiometricPerfBounds{
6618   float FAR;
6619   float FRR;
6620 };
6621
6622
6623
6624
6625
6626
6627
6628
6629
6630
6631
6632
6633
6634
6635
6636
6637
6638
6639
6640
6641
6642
6643
6644
6645
6646
6647
6648
6649
6650
6651

```

Issues Index

The definitions of "lifetime of" and "becomes available" are intended to represent how devices are hot-plugged into (USB) or discovered by (NFC) browsers, and are underspecified. Resolving this with good definitions or some other means will be addressed by resolving Issue #613. RET

@balfanz wishes to add to the "direct" case: If the authenticator violates the privacy requirements of the attestation type it is using, the client SHOULD terminate this algorithm with an "AttestationNotPrivateError". RET

The definitions of "lifetime of" and "becomes available" are intended to represent how devices are hot-plugged into (USB) or discovered by (NFC) browsers, and are underspecified. Resolving this with good definitions or some other means will be addressed by resolving Issue #613. RET

The foregoing step `may` be incorrect, in that we are attempting to create `savedCredentialId` here and use it later below, and we do not have a global in which to allocate a place for it. Perhaps this is good enough? addendum: @jcjones feels the above step is likely good enough. RET

The WHATWG HTML WG is discussing whether to provide a hook when a browsing context gains or loses focuses. If a hook is provided, the above paragraph will be updated to include the hook. See WHATWG HTML WG Issue #2711 for more details. RET

#base64url-encodingReferenced in:
* 5.1. PublicKeyCredential Interface
* 5.1.3. Create a new credential - PublicKeyCredential's

581f [[Create]](origin, options, sameOriginWithAncestors) method (2)
581f * 5.1.4.1. PublicKeyCredential's
581f [[DiscoverFromExternalSource]](origin, options,
581f sameOriginWithAncestors) method (2)
5820 * 7.2. Verifying an authentication assertion

5821
5822 #cborReferenced in:

5825 * 5.1.3. Create a new credential - PublicKeyCredential's
5825 [[Create]](origin, options, sameOriginWithAncestors) method
5825 * 5.1.4.1. PublicKeyCredential's
5825 [[DiscoverFromExternalSource]](origin, options,
5825 sameOriginWithAncestors) method
5825 * 6.1. Authenticator data (2)
5825 * 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
5835 #attestationReferenced in:
5836 * 4. Terminology (2)
5837 * 5.4.6. Attestation Conveyance Preference enumeration (enum
5838 AttestationConveyancePreference) (2)
5839 * 6. WebAuthn Authenticator model (2)
5840 * 6.3. Attestation (2) (3) (4)

5841 * 11.1. WebAuthn Attestation Statement Format Identifier
5842 Registrations

5843
5844 #attestation-certificateReferenced in:
5845 * 4. Terminology (2)
5846 * 5.1.3. Create a new credential - PublicKeyCredential's
5847 [[Create]](origin, options, sameOriginWithAncestors) method
5848 * 8.3.1. TPM attestation statement certificate requirements

5850 #attestation-key-pairReferenced in:
5851 * 4. Terminology (2)
5852 * 6.3. Attestation

5853
5854 #attestation-private-keyReferenced in:
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:

6652 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6653 * 5.1.4.1. PublicKeyCredential's
6654 [[DiscoverFromExternalSource]](origin, options,
6655 sameOriginWithAncestors) method (2)
6656 * 5.10.1. Client data used in WebAuthn signatures (dictionary
6657 CollectedClientData)
6658 * 7.1. Registering a new credential
6659 * 7.2. Verifying an authentication assertion (2)

6660
6661 #cborReferenced in:
6662 * 2.4. All Conformance Classes
6663 * 3. Dependencies
6664 * 5.1.3. Create a new credential - PublicKeyCredential's
6665 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6666 * 5.1.4.1. PublicKeyCredential's
6667 [[DiscoverFromExternalSource]](origin, options,
6668 sameOriginWithAncestors) method
6669 * 6.1. Authenticator data (2)
6670 * 6.2.2. The authenticatorMakeCredential operation
6671 * 6.2.3. The authenticatorGetAssertion operation
6672 * 9. WebAuthn Extensions (2) (3) (4) (5) (6) (7)
6673 * 9.2. Defining extensions (2)
6674 * 9.3. Extending request parameters
6675 * 9.4. Client extension processing (2)
6676 * 9.5. Authenticator extension processing (2)

6677
6678 #assertionReferenced in:
6679 * 7.1. Registering a new credential
6680 * 10.1. FIDO AppID Extension (appid)
6681 * 13.3. credentialId Unsigned

6682
6683 #attestationReferenced in:
6684 * 4. Terminology (2)
6685 * 5.4.6. Attestation Conveyance Preference enumeration (enum
6686 AttestationConveyancePreference) (2)
6687 * 6. WebAuthn Authenticator Model (2)
6688 * 6.3. Attestation (2) (3) (4)
6689 * 8.2. Packed Attestation Statement Format
6690 * 11.1. WebAuthn Attestation Statement Format Identifier
6691 Registrations
6692 * 13. Security Considerations

6693
6694 #attestation-certificateReferenced in:
6695 * 4. Terminology (2)
6696 * 6.3.3. Attestation Types
6697 * 8.3.1. TPM attestation statement certificate requirements

6698
6699 #attestation-key-pairReferenced in:
6700 * 4. Terminology (2)
6701 * 6.3. Attestation
6702 * 6.3.3. Attestation Types

6703
6704 #attestation-private-keyReferenced in:
6705 * 6. WebAuthn Authenticator Model
6706 * 6.3. Attestation
6707 * 8.2. Packed Attestation Statement Format

6708
6709 #attestation-public-keyReferenced in:
6710 * 6.3. Attestation
6711 * 8.2. Packed Attestation Statement Format

6712
6713 #authenticationReferenced in:
6714 * 1. Introduction (2)
6715 * 4. Terminology (2) (3) (4) (5) (6) (7)
6716 * 7.2. Verifying an authentication assertion (2) (3) (4)
6717 * 13. Security Considerations
6718 * 14.3. Authentication Ceremony Privacy

6719
6720 #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
5881 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
5882 (14) (15) (16) (17)
5883 * 5. Web Authentication API (2) (3)
5884 * 5.1. PublicKeyCredential Interface
5885 * 5.1.3. Create a new credential - PublicKeyCredential's
5886 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
5887
5888 * 5.1.4.1. PublicKeyCredential's
5889 [[DiscoverFromExternalSource]](origin, options,
5890 sameOriginWithAncestors) method (2) (3) (4) (5)
5891 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
5892 * 5.2.1. Information about Public Key Credential (interface
5893 AuthenticatorAttestationResponse) (2)
5894 * 5.2.2. Web Authentication Assertion (interface
5895 AuthenticatorAssertionResponse)
5896 * 5.4.1. Public Key Entity Description (dictionary
5897 PublicKeyCredentialEntity) (2)
5898 * 5.4.3. User Account Parameters for Credential Generation
5899 (dictionary PublicKeyCredentialUserEntity)
5900 * 5.4.5. Authenticator Attachment enumeration (enum
5901 AuthenticatorAttachment)
5902 * 5.4.6. Attestation Conveyance Preference enumeration (enum
5903 AttestationConveyancePreference) (2)
5904 * 5.5. Options for Assertion Generation (dictionary
5905 PublicKeyCredentialRequestOptions)
5906 * 6. WebAuthn Authenticator model (2) (3) (4) (5) (6)
5907 * 6.1. Authenticator data
5908 * 6.2.1. The authenticatorMakeCredential operation (2) (3)
5909 * 6.2.2. The authenticatorGetAssertion operation (2) (3)
5910
5911 * 6.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)
5912 * 6.3.2. Attestation Statement Formats
5913
5914 * 6.3.4. Generating an Attestation Object
5915 * 6.3.5.1. Privacy
5916 * 6.3.5.2. Attestation Certificate and Attestation Certificate CA
5917 Compromise
5918 * 7.1. Registering a new credential
5919 * 8.2. Packed Attestation Statement Format
5920 * 8.4. Android Key Attestation Statement Format
5921 * 8.5. Android SafetyNet Attestation Statement Format
5922
5923 * 10.5. Supported Extensions Extension (exts)
5924 * 10.6. User Verification Index Extension (uvi)
5925 * 10.7. Location Extension (loc) (2) (3) (4)
5926 * 10.8. User Verification Method Extension (uvm)
5927 * 12. Sample scenarios
5928
5929 #authorization-gestureReferenced in:
5930 * 1.1.1. Registration

6721 * 1. Introduction
6722 * 4. Terminology (2) (3) (4) (5) (6) (7) (8)
6723 * 5.1. PublicKeyCredential Interface
6724 * 5.2.2. Web Authentication Assertion (interface
6725 AuthenticatorAssertionResponse)
6726 * 5.5. Options for Assertion Generation (dictionary
6727 PublicKeyCredentialRequestOptions)
6728 * 9. WebAuthn Extensions
6729
6730 #authenticatorReferenced in:
6731 * 1. Introduction (2) (3) (4)
6732 * 1.1. Use Cases
6733 * 2.2. Authenticators
6734 * 2.2.1. Backwards Compatibility with FIDO U2F
6735 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
6736 (14) (15) (16) (17) (18) (19)
6737 * 5. Web Authentication API (2) (3)
6738 * 5.1. PublicKeyCredential Interface
6739 * 5.1.3. Create a new credential - PublicKeyCredential's
6740 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
6741 (4)
6742 * 5.1.4.1. PublicKeyCredential's
6743 [[DiscoverFromExternalSource]](origin, options,
6744 sameOriginWithAncestors) method (2) (3) (4) (5) (6)
6745 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
6746 * 5.2.1. Information about Public Key Credential (interface
6747 AuthenticatorAttestationResponse) (2)
6748 * 5.2.2. Web Authentication Assertion (interface
6749 AuthenticatorAssertionResponse)
6750 * 5.4.1. Public Key Entity Description (dictionary
6751 PublicKeyCredentialEntity) (2)
6752 * 5.4.3. User Account Parameters for Credential Generation
6753 (dictionary PublicKeyCredentialUserEntity)
6754 * 5.4.5. Authenticator Attachment enumeration (enum
6755 AuthenticatorAttachment)
6756 * 5.4.6. Attestation Conveyance Preference enumeration (enum
6757 AttestationConveyancePreference) (2)
6758 * 5.5. Options for Assertion Generation (dictionary
6759 PublicKeyCredentialRequestOptions)
6760 * 6. WebAuthn Authenticator Model (2) (3) (4) (5) (6)
6761 * 6.1. Authenticator data
6762 * 6.2.1. Lookup Credential Source by Credential ID algorithm
6763 * 6.2.2. The authenticatorMakeCredential operation (2)
6764 * 6.2.3. The authenticatorGetAssertion operation (2) (3) (4)
6765 * 6.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)
6766 * 6.3.2. Attestation Statement Formats
6767 * 6.3.3. Attestation Types (2) (3) (4)
6768 * 6.3.4. Generating an Attestation Object
6769 * 7.1. Registering a new credential (2)
6770 * 7.2. Verifying an authentication assertion
6771
6772 * 8.2. Packed Attestation Statement Format
6773 * 8.4. Android Key Attestation Statement Format
6774 * 8.5. Android SafetyNet Attestation Statement Format
6775 * 8.7. None Attestation Statement Format
6776 * 10.5. Supported Extensions Extension (exts)
6777 * 10.6. User Verification Index Extension (uvi)
6778
6779 * 10.8. User Verification Method Extension (uvm)
6780 * 12. Sample scenarios
6781 * 13. Security Considerations (2) (3) (4) (5)
6782 * 13.2.2. Attestation Certificate and Attestation Certificate CA
6783 Compromise
6784 * 13.3. credentialId Unsigned
6785 * 14.1. Attestation Privacy (2) (3)
6786 * 14.2. Registration Ceremony Privacy (2) (3) (4) (5) (6)
6787
6788 #authorization-gestureReferenced in:
6789 * 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
5965 `[[DiscoverFromExternalSource]](origin, options,
5966 sameOriginWithAncestors)` method
5967 * 5.2.1. Information about Public Key Credential (interface
5968 AuthenticatorAttestationResponse)
5969 * 6.2.2. The authenticatorGetAssertion operation (2)

5970
5971 * 6.3.1. Attested credential data
5972 * 7.1. Registering a new credential
5973 * 8.6. FIDO U2F Attestation Statement Format
5974 * 12.1. Registration
5975 * 12.3. Authentication (2) (3)

5976
5977 #credential-public-keyReferenced in:

6788 * 1.1.2. Authentication
6789 * 1.1.3. Other use cases and configurations
6790 * 4. Terminology (2) (3) (4) (5) (6)
6791 * 5.1.4. Use an existing credential to make an assertion -
6792 PublicKeyCredential's `[[Get]](options)` method (2)
6793 * 5.1.6. Preventing silent access to an existing credential -
6794 PublicKeyCredential's `[[preventSilentAccess]](credential,
6795 sameOriginWithAncestors)` method

6796
6797 #biometric-recognitionReferenced in:
6798 * 4. Terminology (2) (3)

6799
6800 #biometric-authenticatorReferenced in:
6801 * 10.9. Biometric Authenticator Performance Bounds Extension
6802 (biometricPerfBounds)

6803
6804 #ceremonyReferenced in:
6805 * 1. Introduction
6806 * 4. Terminology (2) (3) (4) (5) (6) (7)
6807 * 7.1. Registering a new credential (2)
6808 * 7.2. Verifying an authentication assertion (2)
6809 * 13. Security Considerations
6810 * 14.2. Registration Ceremony Privacy
6811 * 14.3. Authentication Ceremony Privacy (2)

6812
6813 #clientReferenced in:
6814 * 4. Terminology
6815 * 5.1.7. Availability of User-Verifying Platform Authenticator -
6816 PublicKeyCredential's
6817 `isUserVerifyingPlatformAuthenticatorAvailable()` method (2) (3) (4)
6818 * 5.4.5. Authenticator Attachment enumeration (enum
6819 AuthenticatorAttachment) (2) (3)
6820 * 7.1. Registering a new credential
6821 * 7.2. Verifying an authentication assertion

6822
6823 #client-side-resident-credential-private-keyReferenced in:
6824 * 4. Terminology (2)
6825 * 5.1.3. Create a new credential - PublicKeyCredential's
6826 `[[Create]](origin, options, sameOriginWithAncestors)` method
6827 * 5.4.4. Authenticator Selection Criteria (dictionary
6828 AuthenticatorSelectionCriteria) (2)
6829 * 6.2.2. The authenticatorMakeCredential operation (2)

6830
6831 #conforming-user-agentReferenced in:
6832 * 1. Introduction
6833 * 2.1. User Agents
6834 * 2.2. Authenticators
6835 * 4. Terminology (2)

6836
6837 #credential-idReferenced in:
6838 * 4. Terminology (2) (3) (4)
6839 * 5.1. PublicKeyCredential Interface (2)
6840 * 5.1.4.1. PublicKeyCredential's
6841 `[[DiscoverFromExternalSource]](origin, options,
6842 sameOriginWithAncestors)` method
6843 * 5.2.1. Information about Public Key Credential (interface
6844 AuthenticatorAttestationResponse)
6845 * 5.10.3. Credential Descriptor (dictionary
6846 PublicKeyCredentialDescriptor)
6847 * 6.2.1. Lookup Credential Source by Credential ID algorithm
6848 * 6.2.2. The authenticatorMakeCredential operation
6849 * 6.2.3. The authenticatorGetAssertion operation
6850 * 6.3.1. Attested credential data
6851 * 7.1. Registering a new credential
6852 * 8.6. FIDO U2F Attestation Statement Format
6853 * 12.1. Registration
6854 * 12.3. Authentication (2) (3)
6855 * 13.3. credentialId Unsigned (2) (3)

6856
6857 #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
6001 [[Create]](origin, options, sameOriginWithAncestors) method

6002
6003
6004
6005 #public-key-credentialReferenced in:
6006 * 1. Introduction (2) (3) (4) (5)
6007 * 4. Terminology (2) (3) (4) (5) (6) (7) (8)
6008 * 5. Web Authentication API (2) (3) (4)
6009 * 5.1. PublicKeyCredential Interface
6010 * 5.1.3. Create a new credential - PublicKeyCredential's

6858 * 4. Terminology (2) (3) (4) (5) (6) (7)
6859 * 5.2.1. Information about Public Key Credential (interface
6860 AuthenticatorAttestationResponse)
6861 * 6. WebAuthn Authenticator Model
6862 * 6.3. Attestation (2) (3)
6863 * 6.3.1. Attested credential data (2) (3)
6864 * 12.1. Registration (2)
6865 * 13.3. credentialId Unsigned
6866

6867 #credential-key-pairReferenced in:
6868 * 4. Terminology (2) (3)
6869

6870 #credential-private-keyReferenced in:
6871 * 4. Terminology (2) (3) (4) (5) (6)
6872 * 5.1. PublicKeyCredential Interface
6873 * 5.2.2. Web Authentication Assertion (interface
6874 AuthenticatorAssertionResponse)
6875 * 6. WebAuthn Authenticator Model
6876
6877 * 6.3. Attestation (2)
6878 * 7.2. Verifying an authentication assertion

6879 #human-palatabilityReferenced in:
6880 * 4. Terminology
6881 * 5.4.1. Public Key Entity Description (dictionary
6882 PublicKeyCredentialEntity) (2)
6883

6884 #public-key-credential-sourceReferenced in:
6885 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11)
6886 * 5.1.3. Create a new credential - PublicKeyCredential's
6887 [[Create]](origin, options, sameOriginWithAncestors) method
6888 * 6. WebAuthn Authenticator Model
6889 * 6.2.1. Lookup Credential Source by Credential ID algorithm (2)
6890 * 6.2.2. The authenticatorMakeCredential operation
6891 * 6.2.3. The authenticatorGetAssertion operation (2)
6892

6893 #public-key-credential-source-typeReferenced in:
6894 * 6.2.2. The authenticatorMakeCredential operation (2)
6895

6896 #public-key-credential-source-idReferenced in:
6897 * 6.2.1. Lookup Credential Source by Credential ID algorithm (2)
6898 * 6.2.2. The authenticatorMakeCredential operation
6899 * 6.2.3. The authenticatorGetAssertion operation
6900

6901 #public-key-credential-source-privatekeyReferenced in:
6902 * 6.2.2. The authenticatorMakeCredential operation
6903 * 6.2.3. The authenticatorGetAssertion operation
6904

6905 #public-key-credential-source-rpidReferenced in:
6906 * 6. WebAuthn Authenticator Model
6907 * 6.2.2. The authenticatorMakeCredential operation
6908 * 6.2.3. The authenticatorGetAssertion operation
6909

6910 #public-key-credential-source-userhandleReferenced in:
6911 * 6. WebAuthn Authenticator Model
6912 * 6.2.2. The authenticatorMakeCredential operation
6913 * 6.2.3. The authenticatorGetAssertion operation (2)
6914

6915 #public-key-credential-source-otheruiReferenced in:
6916 * 6.2.2. The authenticatorMakeCredential operation
6917

6918 #public-key-credential-source-managing-authenticatorReferenced in:
6919 * 4. Terminology
6920

6921 #public-key-credentialReferenced in:
6922 * 1. Introduction (2) (3) (4) (5)
6923 * 4. Terminology (2) (3) (4) (5) (6) (7) (8)
6924 * 5. Web Authentication API (2) (3) (4)
6925 * 5.1. PublicKeyCredential Interface
6926 * 5.1.3. Create a new credential - PublicKeyCredential's

6011 [[Create]](origin, options, sameOriginWithAncestors) method
6012 * 5.1.4. Use an existing credential to make an assertion -
6013 PublicKeyCredential's [[Get]](options) method
6014 * 5.1.4.1. PublicKeyCredential's
6015 [[DiscoverFromExternalSource]](origin, options,
6016 sameOriginWithAncestors) method (2)
6017 * 5.2.1. Information about Public Key Credential (interface
6018 AuthenticatorAttestationResponse)
6019 * 5.4.1. Public Key Entity Description (dictionary
6020 PublicKeyCredentialEntity)
6021 * 5.4.4. Authenticator Selection Criteria (dictionary
6022 AuthenticatorSelectionCriteria)

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
6037 #registrationReferenced in:
6038 * 1. Introduction (2)
6039 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
6040 * 7.1. Registering a new credential

6041
6042 #relying-partyReferenced in:
6043 * 1. Introduction (2) (3) (4) (5) (6) (7)
6044 * 1.1.3. Other use cases and configurations
6045 * 2.3. Relying Parties
6046 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
6047 (14) (15) (16) (17) (18) (19) (20) (21) (22) (23) (24) (25) (26)
6048 (27) (28) (29) (30)
6049 * 5. Web Authentication API (2) (3) (4) (5) (6) (7)
6050 * 5.1. PublicKeyCredential Interface (2)
6051 * 5.1.3. Create a new credential - PublicKeyCredential's
6052 [[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)

6927 [[Create]](origin, options, sameOriginWithAncestors) method
6928 * 5.1.4. Use an existing credential to make an assertion -
6929 PublicKeyCredential's [[Get]](options) method
6930 * 5.1.4.1. PublicKeyCredential's
6931 [[DiscoverFromExternalSource]](origin, options,
6932 sameOriginWithAncestors) method (2)
6933 * 5.2.1. Information about Public Key Credential (interface
6934 AuthenticatorAttestationResponse)
6935 * 5.4.1. Public Key Entity Description (dictionary
6936 PublicKeyCredentialEntity)
6937 * 5.4.4. Authenticator Selection Criteria (dictionary
6938 AuthenticatorSelectionCriteria)
6939 * 5.4.5. Authenticator Attachment enumeration (enum
6940 AuthenticatorAttachment) (2) (3)
6941 * 5.5. Options for Assertion Generation (dictionary
6942 PublicKeyCredentialRequestOptions)
6943 * 5.10. Supporting Data Structures
6944 * 5.10.3. Credential Descriptor (dictionary
6945 PublicKeyCredentialDescriptor) (2) (3)
6946 * 6. WebAuthn Authenticator Model (2)
6947 * 6.2.3. The authenticatorGetAssertion operation
6948 * 6.3. Attestation (2)
6949 * 6.3.2. Attestation Statement Formats
6950 * 6.3.3. Attestation Types

6951 * 7.1. Registering a new credential
6952 * 7.2. Verifying an authentication assertion (2)
6953 * 9. WebAuthn Extensions (2)
6954 * 12. Sample scenarios
6955 * 13.2.2. Attestation Certificate and Attestation Certificate CA
6956 Compromise (2)
6957 * 14.2. Registration Ceremony Privacy (2) (3)
6958 * 14.3. Authentication Ceremony Privacy (2) (3) (4) (5)

6959
6960 #registrationReferenced in:
6961 * 1. Introduction (2)
6962 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
6963 * 7.1. Registering a new credential (2) (3)
6964 * 10.9. Biometric Authenticator Performance Bounds Extension
6965 (biometricPerfBounds)
6966 * 13. Security Considerations

6967
6968 #relying-partyReferenced in:
6969 * 1. Introduction (2) (3) (4) (5) (6) (7)
6970 * 1.1.3. Other use cases and configurations
6971 * 2.3. Relying Parties
6972 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
6973 (14) (15) (16) (17) (18) (19) (20) (21) (22) (23) (24) (25) (26)
6974 (27) (28) (29) (30)
6975 * 5. Web Authentication API (2) (3) (4) (5) (6) (7)
6976 * 5.1. PublicKeyCredential Interface (2)
6977 * 5.1.3. Create a new credential - PublicKeyCredential's
6978 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
6979 (4) (5)
6980 * 5.1.4. Use an existing credential to make an assertion -
6981 PublicKeyCredential's [[Get]](options) method (2)
6982 * 5.1.4.1. PublicKeyCredential's
6983 [[DiscoverFromExternalSource]](origin, options,
6984 sameOriginWithAncestors) method (2) (3) (4)
6985 * 5.1.7. Availability of User-Verifying Platform Authenticator -
6986 PublicKeyCredential's
6987 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3)
6988 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
6989 * 5.2.1. Information about Public Key Credential (interface
6990 AuthenticatorAttestationResponse) (2)
6991 * 5.2.2. Web Authentication Assertion (interface
6992 AuthenticatorAssertionResponse)
6993 * 5.4. Options for Credential Creation (dictionary
6994 PublicKeyCredentialCreationOptions) (2) (3) (4) (5)

606E * 5.4.1. Public Key Entity Description (dictionary
 606F PublicKeyCredentialEntity) (2) (3)
 607C * 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)
 607E * 5.4.6. Attestation Conveyance Preference enumeration (enum
 6077 AttestationConveyancePreference) (2) (3) (4) (5) (6) (7)
 607E * 5.5. Options for Assertion Generation (dictionary
 607E PublicKeyCredentialRequestOptions)
 608C * 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)
 608E * 6. WebAuthn Authenticator Model (2)
 608E * 6.1. Authenticator data (2)
 608E * 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6)
 608E * 6.2.1. The authenticatorMakeCredential operation (2) (3) (4) (5)
 609C (6)
 609E * 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)
 609E * 7. Relying Party Operations (2) (3) (4)
 6097 * 7.1. Registering a new credential (2) (3) (4) (5) (6) (7) (8) (9)
 609E (10) (11) (12)
 609E * 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6) (7)
 610C (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)
 610E * 10.1. FIDO AppID Extension (appid) (2)
 6107 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
 610E * 10.4. Authenticator Selection Extension (authnSel) (2) (3)
 610E * 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)
 611E * 12.3. Authentication (2) (3) (4) (5)
 6117 * 12.5. Decommissioning (2)

6118 * 13.1. Cryptographic Challenges

6119
 612C #relying-party-identifierReferenced in:
 6121 * 4. Terminology
 6122 * 5. Web Authentication API
 6123 * 5.4. Options for Credential Creation (dictionary
 6124 MakePublicKeyCredentialOptions)

699E * 5.4.1. Public Key Entity Description (dictionary
 699E PublicKeyCredentialEntity) (2) (3) (4) (5)
 6997 * 5.4.2. RP Parameters for Credential Generation (dictionary
 699E PublicKeyCredentialRpEntity) (2)
 699E * 5.4.3. User Account Parameters for Credential Generation
 7000 (dictionary PublicKeyCredentialUserEntity)
 7001 * 5.4.4. Authenticator Selection Criteria (dictionary
 7002 AuthenticatorSelectionCriteria) (2) (3)
 7003 * 5.4.5. Authenticator Attachment enumeration (enum
 7004 AuthenticatorAttachment) (2) (3) (4) (5) (6)
 700E * 5.4.6. Attestation Conveyance Preference enumeration (enum
 700E AttestationConveyancePreference) (2) (3) (4) (5) (6) (7)
 7007 * 5.5. Options for Assertion Generation (dictionary
 700E PublicKeyCredentialRequestOptions)
 700E * 5.10.1. Client data used in WebAuthn signatures (dictionary
 7009 CollectedClientData) (2) (3) (4)
 701C * 5.10.4. Authenticator Transport enumeration (enum
 7011 AuthenticatorTransport) (2)
 7012 * 5.10.6. User Verification Requirement enumeration (enum
 7013 UserVerificationRequirement) (2) (3) (4)
 7014 * 6. WebAuthn Authenticator Model (2) (3)
 701E * 6.1. Authenticator data (2)
 701E * 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6)
 7018 * 6.2.2. The authenticatorMakeCredential operation (2) (3) (4) (5)
 701E (6)
 7020 * 6.2.3. The authenticatorGetAssertion operation (2) (3)
 7021 * 6.3. Attestation (2) (3) (4) (5) (6)
 7022 * 6.3.3. Attestation Types

7023 * 7. Relying Party Operations (2) (3) (4)
 7024 * 7.1. Registering a new credential (2) (3) (4) (5) (6) (7) (8) (9)
 702E (10) (11) (12) (13)
 702E * 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6) (7)
 702E (8)
 702E * 8.4. Android Key Attestation Statement Format
 702E * 8.7. None Attestation Statement Format
 702E * 9. WebAuthn Extensions (2) (3) (4) (5)
 7031 * 9.2. Defining extensions (2)
 703E * 9.3. Extending request parameters (2) (3) (4)
 7033 * 10.1. FIDO AppID Extension (appid) (2)

7034 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
 703E * 10.4. Authenticator Selection Extension (authnSel) (2) (3)
 703E * 10.5. Supported Extensions Extension (exts) (2)
 7037 * 10.6. User Verification Index Extension (uvi)
 703E * 10.7. Location Extension (loc) (2)
 703E * 10.8. User Verification Method Extension (uvm)
 7039 * 10.9. Biometric Authenticator Performance Bounds Extension
 7040 (biometricPerfBounds) (2) (3)
 7041 * 11.2. WebAuthn Extension Identifier Registrations (2)
 7042 * 12.1. Registration (2) (3) (4) (5)
 7043 * 12.2. Registration Specifically with User Verifying Platform
 7044 Authenticator (2) (3)
 704E * 12.3. Authentication (2) (3) (4) (5)
 704E * 12.5. Decommissioning (2)
 7047 * 13. Security Considerations (2) (3) (4)
 704E * 13.1. Cryptographic Challenges
 7050 * 13.2.2. Attestation Certificate and Attestation Certificate CA
 7051 Compromise (2) (3) (4) (5) (6)
 7052 * 13.3. credentialId Unsigned
 7053 * 14.1. Attestation Privacy
 7054 * 14.2. Registration Ceremony Privacy (2) (3) (4)
 705E * 14.3. Authentication Ceremony Privacy (2) (3) (4)

705E
 7057
 705E
 705E
 705E
 706C
 706E

#relying-party-identifierReferenced in:
 * 4. Terminology
 * 5. Web Authentication API
 * 5.4. Options for Credential Creation (dictionary
 PublicKeyCredentialCreationOptions)

6125 * 5.5. Options for Assertion Generation (dictionary
6126 PublicKeyCredentialRequestOptions)
6127 * 6. WebAuthn Authenticator model

#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

* 6.1. Authenticator data (2) (3) (4) (5) (6)
6140 * 6.1.1. Signature Counter Considerations
6141 * 6.2.1. The authenticatorMakeCredential operation (2)
6142 * 6.2.2. The authenticatorGetAssertion operation (2)
6143 * 7.1. Registering a new credential (2)
6144 * 7.2. Verifying an authentication assertion
6145 * 8.4. Android Key Attestation Statement Format
6146 * 8.6. FIDO U2F Attestation Statement Format
6147

#test-of-user-presenceReferenced in:
6148 * 4. Terminology (2) (3) (4) (5) (6)
6149 * 6.2.1. The authenticatorMakeCredential operation
6150 * 6.2.2. The authenticatorGetAssertion operation
6151 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
6152 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
6153

#user-consentReferenced in:
6154 * 1. Introduction (2)
6155 * 4. Terminology (2)
6156 * 5. Web Authentication API

* 5.1.4. Use an existing credential to make an assertion -
6160 PublicKeyCredential's [[Get]](options) method
6161

* 5.2.2. Web Authentication Assertion (interface
6162 AuthenticatorAssertionResponse)
6163 * 5.4.6. Attestation Conveyance Preference enumeration (enum
6164 AttestationConveyancePreference)
6165 * 6. WebAuthn Authenticator model (2) (3)
6166 * 6.2.1. The authenticatorMakeCredential operation (2) (3) (4) (5)
6167 (6)
6168 * 6.2.2. The authenticatorGetAssertion operation (2) (3) (4) (5)
6169 * 11.2. WebAuthn Extension Identifier Registrations
6170

#user-handleReferenced in:
6171 * 4. Terminology
6172 * 5.1.4.1. PublicKeyCredential's
6173 [[DiscoverFromExternalSource]](origin, options,
6174 sameOriginWithAncestors) method
6175 * 5.2.2. Web Authentication Assertion (interface
6176 AuthenticatorAssertionResponse)
6177 * 5.4. Options for Credential Creation (dictionary
6178 MakePublicKeyCredentialOptions)
6179 * 5.4.3. User Account Parameters for Credential Generation
6180 (dictionary PublicKeyCredentialUserEntity)
6181 * 6.2.1. The authenticatorMakeCredential operation
6182 * 6.2.2. The authenticatorGetAssertion operation
6183
6184
6185

7062 * 5.5. Options for Assertion Generation (dictionary
7063 PublicKeyCredentialRequestOptions)

#rp-idReferenced in:
7064 * 4. Terminology (2) (3) (4) (5)
7065 * 5. Web Authentication API (2) (3) (4) (5)
7066 * 5.1.3. Create a new credential - PublicKeyCredential's
7067 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7068 * 5.1.4.1. PublicKeyCredential's
7069 [[DiscoverFromExternalSource]](origin, options,
7070 sameOriginWithAncestors) method (2)
7071 * 5.4.2. RP Parameters for Credential Generation (dictionary
7072 PublicKeyCredentialRpEntity)
7073
7074

* 6.1. Authenticator data (2) (3) (4) (5) (6)
7075 * 6.1.1. Signature Counter Considerations
7076 * 6.2.2. The authenticatorMakeCredential operation (2) (3)
7077 * 6.2.3. The authenticatorGetAssertion operation (2)
7078 * 7.1. Registering a new credential (2)
7079 * 7.2. Verifying an authentication assertion
7080 * 8.4. Android Key Attestation Statement Format
7081 * 8.6. FIDO U2F Attestation Statement Format
7082 * 10.1. FIDO AppID Extension (appid)
7083

#test-of-user-presenceReferenced in:
7084 * 4. Terminology (2) (3) (4) (5) (6)
7085 * 6.2.2. The authenticatorMakeCredential operation (2)
7086 * 6.2.3. The authenticatorGetAssertion operation
7087 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
7088 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
7089

#user-consentReferenced in:
7090 * 1. Introduction (2)
7091 * 4. Terminology (2)
7092 * 5. Web Authentication API
7093 * 5.1.3. Create a new credential - PublicKeyCredential's
7094 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
7095 * 5.1.4. Use an existing credential to make an assertion -
7096 PublicKeyCredential's [[Get]](options) method
7097 * 5.1.4.1. PublicKeyCredential's
7098 [[DiscoverFromExternalSource]](origin, options,
7099 sameOriginWithAncestors) method
7100 * 5.2.2. Web Authentication Assertion (interface
7101 AuthenticatorAssertionResponse)
7102 * 5.4.6. Attestation Conveyance Preference enumeration (enum
7103 AttestationConveyancePreference)
7104 * 6. WebAuthn Authenticator Model (2) (3)
7105 * 6.2.2. The authenticatorMakeCredential operation (2) (3) (4) (5)
7106 (6) (7) (8)
7107 * 6.2.3. The authenticatorGetAssertion operation (2) (3) (4) (5)
7108 * 11.2. WebAuthn Extension Identifier Registrations
7109 * 14.2. Registration Ceremony Privacy (2)
7110 * 14.3. Authentication Ceremony Privacy (2) (3)
7111

#user-handleReferenced in:
7112 * 2.2.1. Backwards Compatibility with FIDO U2F
7113 * 4. Terminology
7114 * 5.1.4.1. PublicKeyCredential's
7115 [[DiscoverFromExternalSource]](origin, options,
7116 sameOriginWithAncestors) method (2)
7117 * 5.2.2. Web Authentication Assertion (interface
7118 AuthenticatorAssertionResponse) (2)
7119
7120
7121
7122

* 5.4.3. User Account Parameters for Credential Generation
7123 (dictionary PublicKeyCredentialUserEntity)
7124 * 6.2.2. The authenticatorMakeCredential operation
7125
7126

```

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)

* 10.2. Simple Transaction Authorization Extension (txAuthSimple)
* 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
* 12.2. Registration Specifically with User Verifying Platform
Authenticator

#concept-user-presentReferenced in:
* 4. Terminology
* 6.1. Authenticator data (2) (3)

#upReferenced in:
* 6.1. Authenticator data

#concept-user-verifiedReferenced in:
* 4. Terminology
* 6.1. Authenticator data (2) (3)

#uvReferenced in:
* 5.8.6. User Verification Requirement enumeration (enum
UserVerificationRequirement) (2)
* 6.1. Authenticator data

#webauthn-clientReferenced in:
* 4. Terminology (2)
* 6.2.1. The authenticatorMakeCredential operation
* 6.2.2. The authenticatorGetAssertion operation

#web-authentication-apiReferenced in:
* 1. Introduction (2) (3)
* 4. Terminology (2)

#publickeycredentialReferenced in:
* 1. Introduction
* 5.1. PublicKeyCredential Interface (2) (3) (4) (5) (6) (7) (8)
* 5.1.3. Create a new credential - PublicKeyCredential's
[[Create]](origin, options, sameOriginWithAncestors) method (2)
* 5.1.4.1. PublicKeyCredential's
[[DiscoverFromExternalSource]](origin, options,
sameOriginWithAncestors) method (2)
* 5.1.5. Store an existing credential - PublicKeyCredential's
[[Store]](credential, sameOriginWithAncestors) method (2)
* 5.1.6. Availability of User-Verifying Platform Authenticator -

```

```

7127 #user-verificationReferenced in:
7128 * 1. Introduction
7129 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
7130 * 5.1.3. Create a new credential - PublicKeyCredential's
7131 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
7132 * 5.1.4.1. PublicKeyCredential's
7133 [[DiscoverFromExternalSource]](origin, options,
7134 sameOriginWithAncestors) method (2) (3)
7135 * 5.1.7. Availability of User-Verifying Platform Authenticator -
7136 PublicKeyCredential's
7137 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)
7138 (5)
7139 * 5.4.4. Authenticator Selection Criteria (dictionary
7140 AuthenticatorSelectionCriteria)
7141 * 5.5. Options for Assertion Generation (dictionary
7142 PublicKeyCredentialRequestOptions)
7143 * 5.10.6. User Verification Requirement enumeration (enum
7144 UserVerificationRequirement) (2) (3) (4)
7145 * 6.2.2. The authenticatorMakeCredential operation (2) (3)
7146 * 6.2.3. The authenticatorGetAssertion operation
7147 * 7.1. Registering a new credential (2)
7148 * 7.2. Verifying an authentication assertion (2)
7149 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
7150 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
7151 * 12.2. Registration Specifically with User Verifying Platform
7152 Authenticator

#concept-user-presentReferenced in:
# 4. Terminology
# 6.1. Authenticator data (2) (3)
# 7.1. Registering a new credential
# 7.2. Verifying an authentication assertion

#upReferenced in:
* 6.1. Authenticator data

#concept-user-verifiedReferenced in:
* 4. Terminology
* 6.1. Authenticator data (2) (3)
# 7.1. Registering a new credential
# 7.2. Verifying an authentication assertion

#uvReferenced in:
* 5.10.6. User Verification Requirement enumeration (enum
UserVerificationRequirement) (2)
* 6.1. Authenticator data

#webauthn-clientReferenced in:
* 4. Terminology (2) (3) (4)
* 6.2. Authenticator operations
* 6.2.2. The authenticatorMakeCredential operation
* 6.2.3. The authenticatorGetAssertion operation
* 13. Security Considerations

#web-authentication-apiReferenced in:
* 1. Introduction (2) (3)
* 4. Terminology (2) (3) (4)
# 13. Security Considerations

#publickeycredentialReferenced in:
* 1. Introduction
* 5.1. PublicKeyCredential Interface (2) (3) (4) (5) (6) (7) (8)
* 5.1.3. Create a new credential - PublicKeyCredential's
[[Create]](origin, options, sameOriginWithAncestors) method (2)
* 5.1.4.1. PublicKeyCredential's
[[DiscoverFromExternalSource]](origin, options,
sameOriginWithAncestors) method
* 5.1.5. Store an existing credential - PublicKeyCredential's
[[Store]](credential, sameOriginWithAncestors) method (2)
* 5.1.7. Availability of User-Verifying Platform Authenticator -

```

```

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
6299
6300 * 5.6. Abort operations with AbortSignal (2) (3) (4) (5)
6301 * 6.2.1. The authenticatorMakeCredential operation
6302
6303 #dom-publickeycredential-create-origin-options-sameoriginwithancestors-
6304 originReferenced in:
6305 * 5.1.3. Create a new credential - PublicKeyCredential's
6306 [[Create]](origin, options, sameOriginWithAncestors) method
6307
6308 #dom-publickeycredential-create-origin-options-sameoriginwithancestors-
6309 optionsReferenced in:
6310 * 7.1. Registering a new credential
6311
6312 #effective-user-verification-requirement-for-credential-creationReferen
6313 ced in:
6314 * 6.2.1. The authenticatorMakeCredential operation

```

```

7197 PublicKeyCredential's
7198 isUserVerifyingPlatformAuthenticatorAvailable() method
7199 * 5.10.3. Credential Descriptor (dictionary
7200 PublicKeyCredentialDescriptor)
7201 * 7. Relying Party Operations
7202 * 7.2. Verifying an authentication assertion
7203
7204 #dom-publickeycredential-rawidReferenced in:
7205 * 5.1. PublicKeyCredential Interface
7206 * 7.2. Verifying an authentication assertion
7207
7208 #dom-publickeycredential-getclientextensionresultsReferenced in:
7209 * 5.1. PublicKeyCredential Interface
7210 * 9.4. Client extension processing
7211
7212 #dom-publickeycredential-responseReferenced in:
7213 * 5.1. PublicKeyCredential Interface
7214 * 5.1.3. Create a new credential - PublicKeyCredential's
7215 [[Create]](origin, options, sameOriginWithAncestors) method
7216 * 5.1.4.1. PublicKeyCredential's
7217 [[DiscoverFromExternalSource]](origin, options,
7218 sameOriginWithAncestors) method
7219 * 7.2. Verifying an authentication assertion (2)
7220
7221 #dom-publickeycredential-identifier-slotReferenced in:
7222 * 5.1. PublicKeyCredential Interface (2)
7223 * 5.1.3. Create a new credential - PublicKeyCredential's
7224 [[Create]](origin, options, sameOriginWithAncestors) method
7225 * 5.1.4.1. PublicKeyCredential's
7226 [[DiscoverFromExternalSource]](origin, options,
7227 sameOriginWithAncestors) method
7228
7229 #dom-publickeycredential-clientextensionsresults-slotReferenced in:
7230 * 5.1. PublicKeyCredential Interface
7231 * 5.1.3. Create a new credential - PublicKeyCredential's
7232 [[Create]](origin, options, sameOriginWithAncestors) method
7233 * 5.1.4.1. PublicKeyCredential's
7234 [[DiscoverFromExternalSource]](origin, options,
7235 sameOriginWithAncestors) method
7236
7237 #dom-credentialcreationoptions-publickeyReferenced in:
7238 * 5.1.3. Create a new credential - PublicKeyCredential's
7239 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
7240
7241 #dom-credentialrequestoptions-publickeyReferenced in:
7242 * 5.1.4.1. PublicKeyCredential's
7243 [[DiscoverFromExternalSource]](origin, options,
7244 sameOriginWithAncestors) method (2) (3)
7245
7246 #dom-publickeycredential-create-slotReferenced in:
7247 * 4. Terminology
7248 * 5.1. PublicKeyCredential Interface
7249 * 5.4.5. Authenticator Attachment enumeration (enum
7250 AuthenticatorAttachment)
7251 * 5.6. Abort operations with AbortSignal (2) (3) (4) (5)
7252 * 6.2.2. The authenticatorMakeCredential operation
7253 * 14.2. Registration Ceremony Privacy
7254
7255 #dom-publickeycredential-create-origin-options-sameoriginwithancestors-
7256 originReferenced in:
7257 * 5.1.3. Create a new credential - PublicKeyCredential's
7258 [[Create]](origin, options, sameOriginWithAncestors) method
7259
7260 #dom-publickeycredential-create-origin-options-sameoriginwithancestors-
7261 optionsReferenced in:
7262 * 7.1. Registering a new credential
7263
7264 #effective-user-verification-requirement-for-credential-creationReferen
7265 ced in:
7266 * 6.2.2. The authenticatorMakeCredential operation

```

6313 #credentialcreationdata-attestationobjectresultReferenced in:
6314 * 5.1.3. Create a new credential - PublicKeyCredential's
6315 [[Create]](origin, options, sameOriginWithAncestors) method
6316

6317 #credentialcreationdata-clientdatajsonresultReferenced in:
6318 * 5.1.3. Create a new credential - PublicKeyCredential's
6319 [[Create]](origin, options, sameOriginWithAncestors) method
6320

6321 #credentialcreationdata-attestationconveyancepreferenceoptionReferenced
6322 in:
6323 * 5.1.3. Create a new credential - PublicKeyCredential's
6324 [[Create]](origin, options, sameOriginWithAncestors) method
6325

6326 #credentialcreationdata-clientextensionresultsReferenced in:
6327 * 5.1.3. Create a new credential - PublicKeyCredential's
6328 [[Create]](origin, options, sameOriginWithAncestors) method
6329

6330 #dom-publickeycredential-collectfromcredentialstore-slotReferenced in:
6331 * 5.1.4. Use an existing credential to make an assertion -
6332 PublicKeyCredential's [[Get]](options) method
6333

6334 #dom-publickeycredential-discoverfromexternalsource-slotReferenced in:
6335

6336 * 5.1. PublicKeyCredential Interface
6337 * 5.1.4. Use an existing credential to make an assertion -
6338 PublicKeyCredential's [[Get]](options) method

6339 * 5.6. Abort operations with AbortSignal (2) (3) (4) (5)
6340 * 6.2.2. The authenticatorGetAssertion operation

6341 #dom-publickeycredential-discoverfromexternalsource-origin-options-same
6342 originwithancestors-originReferenced in:
6343 * 5.1.4.1. PublicKeyCredential's
6344 [[DiscoverFromExternalSource]](origin, options,
6345 sameOriginWithAncestors) method
6346

6347 #effective-user-verification-requirement-for-assertionReferenced in:
6348 * 6.2.2. The authenticatorGetAssertion operation
6349

6350 #assertioncreationdata-credentialidresultReferenced in:
6351 * 5.1.4.1. PublicKeyCredential's
6352 [[DiscoverFromExternalSource]](origin, options,
6353 sameOriginWithAncestors) method (2) (3)
6354

6355 #assertioncreationdata-clientdatajsonresultReferenced in:
6356 * 5.1.4.1. PublicKeyCredential's
6357 [[DiscoverFromExternalSource]](origin, options,
6358 sameOriginWithAncestors) method
6359

6360 #assertioncreationdata-authenticatordataresultReferenced in:
6361 * 5.1.4.1. PublicKeyCredential's
6362 [[DiscoverFromExternalSource]](origin, options,
6363 sameOriginWithAncestors) method
6364

6365 #assertioncreationdata-signatureresultReferenced in:
6366 * 5.1.4.1. PublicKeyCredential's
6367 [[DiscoverFromExternalSource]](origin, options,
6368 sameOriginWithAncestors) method
6369

6370 #assertioncreationdata-userhandlerresultReferenced in:
6371 * 5.1.4.1. PublicKeyCredential's
6372 [[DiscoverFromExternalSource]](origin, options,
6373 sameOriginWithAncestors) method
6374

6375 #assertioncreationdata-clientextensionresultsReferenced in:
6376

7267 #credentialcreationdata-attestationobjectresultReferenced in:
7268 * 5.1.3. Create a new credential - PublicKeyCredential's
7269 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
7270 (4) (5)
7271

7272 #credentialcreationdata-clientdatajsonresultReferenced in:
7273 * 5.1.3. Create a new credential - PublicKeyCredential's
7274 [[Create]](origin, options, sameOriginWithAncestors) method
7275

7276 #credentialcreationdata-attestationconveyancepreferenceoptionReferenced
7277 in:
7278 * 5.1.3. Create a new credential - PublicKeyCredential's
7279 [[Create]](origin, options, sameOriginWithAncestors) method
7280

7281 #credentialcreationdata-clientextensionresultsReferenced in:
7282 * 5.1.3. Create a new credential - PublicKeyCredential's
7283 [[Create]](origin, options, sameOriginWithAncestors) method
7284

7285 #dom-publickeycredential-collectfromcredentialstore-slotReferenced in:
7286 * 5.1.4. Use an existing credential to make an assertion -
7287 PublicKeyCredential's [[Get]](options) method
7288

7289 #dom-publickeycredential-discoverfromexternalsource-slotReferenced in:
7290 * 4. Terminology
7291 * 5.1. PublicKeyCredential Interface
7292 * 5.1.4. Use an existing credential to make an assertion -
7293 PublicKeyCredential's [[Get]](options) method
7294 * 5.4.5. Authenticator Attachment enumeration (enum
7295 AuthenticatorAttachment)
7296 * 5.6. Abort operations with AbortSignal (2) (3) (4) (5)
7297 * 6.2.3. The authenticatorGetAssertion operation
7298 * 14.3. Authentication Ceremony Privacy
7299

7300 #dom-publickeycredential-discoverfromexternalsource-origin-options-same
7301 originwithancestors-originReferenced in:
7302 * 5.1.4.1. PublicKeyCredential's
7303 [[DiscoverFromExternalSource]](origin, options,
7304 sameOriginWithAncestors) method
7305

7306 #effective-user-verification-requirement-for-assertionReferenced in:
7307 * 6.2.3. The authenticatorGetAssertion operation
7308

7309 #assertioncreationdata-credentialidresultReferenced in:
7310 * 5.1.4.1. PublicKeyCredential's
7311 [[DiscoverFromExternalSource]](origin, options,
7312 sameOriginWithAncestors) method (2) (3)
7313

7314 #assertioncreationdata-clientdatajsonresultReferenced in:
7315 * 5.1.4.1. PublicKeyCredential's
7316 [[DiscoverFromExternalSource]](origin, options,
7317 sameOriginWithAncestors) method
7318

7319 #assertioncreationdata-authenticatordataresultReferenced in:
7320 * 5.1.4.1. PublicKeyCredential's
7321 [[DiscoverFromExternalSource]](origin, options,
7322 sameOriginWithAncestors) method
7323

7324 #assertioncreationdata-signatureresultReferenced in:
7325 * 5.1.4.1. PublicKeyCredential's
7326 [[DiscoverFromExternalSource]](origin, options,
7327 sameOriginWithAncestors) method
7328

7329 #assertioncreationdata-userhandlerresultReferenced in:
7330 * 5.1.4.1. PublicKeyCredential's
7331 [[DiscoverFromExternalSource]](origin, options,
7332 sameOriginWithAncestors) method (2) (3) (4)
7333 * 6.2.3. The authenticatorGetAssertion operation
7334

7335 #assertioncreationdata-clientextensionresultsReferenced in:
7336

```

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
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:

```

```

7337 * 5.1.4.1. PublicKeyCredential's
7338 [[DiscoverFromExternalSource]](origin, options,
7339 sameOriginWithAncestors) method
7340
7341 #authenticatorresponseReferenced in:
7342 * 5.1. PublicKeyCredential Interface (2)
7343 * 5.2. Authenticator Responses (interface AuthenticatorResponse) (2)
7344 * 5.2.1. Information about Public Key Credential (interface
7345 AuthenticatorAttestationResponse) (2)
7346 * 5.2.2. Web Authentication Assertion (interface
7347 AuthenticatorAssertionResponse) (2)
7348
7349 #dom-authenticatorresponse-clientdatajsonReferenced in:
7350 * 5.1.3. Create a new credential - PublicKeyCredential's
7351 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7352 * 5.1.4.1. PublicKeyCredential's
7353 [[DiscoverFromExternalSource]](origin, options,
7354 sameOriginWithAncestors) method (2)
7355 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
7356 * 5.2.1. Information about Public Key Credential (interface
7357 AuthenticatorAttestationResponse)
7358 * 5.2.2. Web Authentication Assertion (interface
7359 AuthenticatorAssertionResponse)
7360 * 7.1. Registering a new credential (2)
7361 * 7.2. Verifying an authentication assertion
7362
7363 #authenticatorattestationresponseReferenced in:
7364 * 5.1. PublicKeyCredential Interface
7365 * 5.1.3. Create a new credential - PublicKeyCredential's
7366 [[Create]](origin, options, sameOriginWithAncestors) method
7367 * 5.2.1. Information about Public Key Credential (interface
7368 AuthenticatorAttestationResponse) (2)
7369 * 7. Relying Party Operations
7370 * 7.1. Registering a new credential (2)
7371
7372 #dom-authenticatorattestationresponse-attestationobjectReferenced in:
7373 * 5.1.3. Create a new credential - PublicKeyCredential's
7374 [[Create]](origin, options, sameOriginWithAncestors) method
7375 * 5.2.1. Information about Public Key Credential (interface
7376 AuthenticatorAttestationResponse)
7377 * 7.1. Registering a new credential
7378
7379 #authenticatorassertionresponseReferenced in:
7380 * 4. Terminology
7381 * 5.1. PublicKeyCredential Interface
7382 * 5.1.4.1. PublicKeyCredential's
7383 [[DiscoverFromExternalSource]](origin, options,
7384 sameOriginWithAncestors) method
7385 * 5.2.2. Web Authentication Assertion (interface
7386 AuthenticatorAssertionResponse) (2)
7387 * 7. Relying Party Operations
7388
7389 #dom-authenticatorassertionresponse-authenticatordataReferenced in:
7390 * 5.1.4.1. PublicKeyCredential's
7391 [[DiscoverFromExternalSource]](origin, options,
7392 sameOriginWithAncestors) method
7393 * 5.2.2. Web Authentication Assertion (interface
7394 AuthenticatorAssertionResponse)
7395 * 7.2. Verifying an authentication assertion
7396
7397 #dom-authenticatorassertionresponse-signatureReferenced in:
7398 * 5.1.4.1. PublicKeyCredential's
7399 [[DiscoverFromExternalSource]](origin, options,
7400 sameOriginWithAncestors) method
7401 * 5.2.2. Web Authentication Assertion (interface
7402 AuthenticatorAssertionResponse)
7403 * 7.2. Verifying an authentication assertion
7404
7405 #dom-authenticatorassertionresponse-userhandleReferenced in:
7406 * 2.2.1. Backwards Compatibility with FIDO U2F

```

6446 * 5.1.4.1. PublicKeyCredential's
6447 [[DiscoverFromExternalSource]](origin, options,
6448 sameOriginWithAncestors) method
6449 * 5.2.2. Web Authentication Assertion (interface
6450 AuthenticatorAssertionResponse)

6451
6452 #dictdef-publickeycredentialparametersReferenced in:
6453 * 5.3. Parameters for Credential Generation (dictionary
6454 PublicKeyCredentialParameters)
6455 * 5.4. Options for Credential Creation (dictionary
6456 MakePublicKeyCredentialOptions) (2)

6457
6458 #dom-publickeycredentialparameters-typeReferenced in:
6459 * 5.1.3. Create a new credential - PublicKeyCredential's
6460 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6461 * 5.3. Parameters for Credential Generation (dictionary
6462 PublicKeyCredentialParameters)

6463
6464 #dom-publickeycredentialparameters-algReferenced in:
6465 * 5.1.3. Create a new credential - PublicKeyCredential's
6466 [[Create]](origin, options, sameOriginWithAncestors) method
6467 * 5.3. Parameters for Credential Generation (dictionary
6468 PublicKeyCredentialParameters)

6469
6470 #dictdef-makepublickeycredentialoptionsReferenced in:
6471 * 5.1.1. CredentialCreationOptions Extension
6472 * 5.1.3. Create a new credential - PublicKeyCredential's
6473 [[Create]](origin, options, sameOriginWithAncestors) method
6474 * 5.4. Options for Credential Creation (dictionary
6475 MakePublicKeyCredentialOptions)

6476
6477 #dom-makepublickeycredentialoptions-rpReferenced in:
6478 * 5.1.3. Create a new credential - PublicKeyCredential's
6479 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
6480 (4) (5) (6)
6481 * 5.4. Options for Credential Creation (dictionary
6482 MakePublicKeyCredentialOptions)

6483
6484 #dom-makepublickeycredentialoptions-userReferenced in:
6485 * 5.1.3. Create a new credential - PublicKeyCredential's
6486 [[Create]](origin, options, sameOriginWithAncestors) method
6487 * 5.4. Options for Credential Creation (dictionary
6488 MakePublicKeyCredentialOptions)

6489 * 7.1. Registering a new credential

6490
6491 #dom-makepublickeycredentialoptions-challengeReferenced in:
6492 * 5.1.3. Create a new credential - PublicKeyCredential's
6493 [[Create]](origin, options, sameOriginWithAncestors) method
6494 * 5.4. Options for Credential Creation (dictionary
6495 MakePublicKeyCredentialOptions)

6496
6497 #dom-makepublickeycredentialoptions-pubkeycredparamsReferenced in:
6498 * 5.1.3. Create a new credential - PublicKeyCredential's
6499 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6500 * 5.4. Options for Credential Creation (dictionary
6501 MakePublicKeyCredentialOptions)

6502
6503 #dom-makepublickeycredentialoptions-timeoutReferenced in:
6504 * 5.1.3. Create a new credential - PublicKeyCredential's
6505 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6506 * 5.4. Options for Credential Creation (dictionary
6507 MakePublicKeyCredentialOptions)

6508
6509 #dom-makepublickeycredentialoptions-excludecredentialsReferenced in:
6510 * 5.1.3. Create a new credential - PublicKeyCredential's
6511 [[Create]](origin, options, sameOriginWithAncestors) method
6512 * 5.4. Options for Credential Creation (dictionary

7407 * 5.1.4.1. PublicKeyCredential's
7408 [[DiscoverFromExternalSource]](origin, options,
7409 sameOriginWithAncestors) method
7410 * 5.2.2. Web Authentication Assertion (interface
7411 AuthenticatorAssertionResponse)

7412 * 7.2. Verifying an authentication assertion

7413
7414 #dictdef-publickeycredentialparametersReferenced in:
7415 * 5.3. Parameters for Credential Generation (dictionary
7416 PublicKeyCredentialParameters)
7417 * 5.4. Options for Credential Creation (dictionary
7418 PublicKeyCredentialCreationOptions) (2)

7419
7420 #dom-publickeycredentialparameters-typeReferenced in:
7421 * 5.1.3. Create a new credential - PublicKeyCredential's
7422 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7423 * 5.3. Parameters for Credential Generation (dictionary
7424 PublicKeyCredentialParameters)

7425
7426 #dom-publickeycredentialparameters-algReferenced in:
7427 * 5.1.3. Create a new credential - PublicKeyCredential's
7428 [[Create]](origin, options, sameOriginWithAncestors) method
7429 * 5.3. Parameters for Credential Generation (dictionary
7430 PublicKeyCredentialParameters)

7431
7432 #dictdef-publickeycredentialcreationoptionsReferenced in:
7433 * 5.1.1. CredentialCreationOptions Dictionary Extension
7434 * 5.1.3. Create a new credential - PublicKeyCredential's
7435 [[Create]](origin, options, sameOriginWithAncestors) method
7436 * 5.4. Options for Credential Creation (dictionary
7437 PublicKeyCredentialCreationOptions)

7438
7439 #dom-publickeycredentialcreationoptions-rpReferenced in:
7440 * 5.1.3. Create a new credential - PublicKeyCredential's
7441 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
7442 (4) (5) (6)
7443 * 5.4. Options for Credential Creation (dictionary
7444 PublicKeyCredentialCreationOptions)

7445
7446 #dom-publickeycredentialcreationoptions-userReferenced in:
7447 * 5.1.3. Create a new credential - PublicKeyCredential's
7448 [[Create]](origin, options, sameOriginWithAncestors) method
7449 * 5.4. Options for Credential Creation (dictionary
7450 PublicKeyCredentialCreationOptions)

7451 * 7.1. Registering a new credential

7452
7453 #dom-publickeycredentialcreationoptions-challengeReferenced in:
7454 * 5.1.3. Create a new credential - PublicKeyCredential's
7455 [[Create]](origin, options, sameOriginWithAncestors) method
7456 * 5.4. Options for Credential Creation (dictionary
7457 PublicKeyCredentialCreationOptions)

7458 * 13.1. Cryptographic Challenges

7459
7460 #dom-publickeycredentialcreationoptions-pubkeycredparamsReferenced in:
7461 * 5.1.3. Create a new credential - PublicKeyCredential's
7462 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7463 * 5.4. Options for Credential Creation (dictionary
7464 PublicKeyCredentialCreationOptions)

7465
7466 #dom-publickeycredentialcreationoptions-timeoutReferenced in:
7467 * 5.1.3. Create a new credential - PublicKeyCredential's
7468 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7469 * 5.4. Options for Credential Creation (dictionary
7470 PublicKeyCredentialCreationOptions)

7471
7472 #dom-publickeycredentialcreationoptions-excludecredentialsReferenced
7473 in:
7474 * 5.1.3. Create a new credential - PublicKeyCredential's
7475 [[Create]](origin, options, sameOriginWithAncestors) method
7476 * 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
6538 #dictdef-publickeycredentialentityReferenced in:
6539 * 5.4.1. Public Key Entity Description (dictionary
6540 PublicKeyCredentialEntity) (2)
6541 * 5.4.2. RP Parameters for Credential Generation (dictionary
6542 PublicKeyCredentialRpEntity)
6543 * 5.4.3. User Account Parameters for Credential Generation
6544 (dictionary PublicKeyCredentialUserEntity)
6545
6546 #dom-publickeycredentialentity-nameReferenced in:
6547 * 5.4. Options for Credential Creation (dictionary
6548 MakePublicKeyCredentialOptions) (2)
6549 * 5.4.1. Public Key Entity Description (dictionary
6550 PublicKeyCredentialEntity)
6551 * 6.2.1. The authenticatorMakeCredential operation (2)
6552
6553 #dom-publickeycredentialentity-iconReferenced in:
6554 * 5.4.1. Public Key Entity Description (dictionary
6555 PublicKeyCredentialEntity)
6556
6557 #dictdef-publickeycredentialrpentityReferenced in:
6558 * 5.4. Options for Credential Creation (dictionary
6559 MakePublicKeyCredentialOptions) (2)
6560
6561 * 5.4.2. RP Parameters for Credential Generation (dictionary
6562 PublicKeyCredentialRpEntity) (2)
6563 * 6.2.1. The authenticatorMakeCredential operation
6564
6565 #dom-publickeycredentialrpentity-idReferenced in:
6566
6567 * 5.4. Options for Credential Creation (dictionary
6568 MakePublicKeyCredentialOptions)
6569 * 5.4.2. RP Parameters for Credential Generation (dictionary
6570 PublicKeyCredentialRpEntity)
6571 * 6.2.1. The authenticatorMakeCredential operation (2) (3) (4)
6572
6573 #dictdef-publickeycredentialuserentityReferenced in:
6574 * 5.4. Options for Credential Creation (dictionary
6575 MakePublicKeyCredentialOptions) (2)
6576
6577 * 5.4.3. User Account Parameters for Credential Generation

```

```

7477 PublicKeyCredentialCreationOptions)
7478
7479 #dom-publickeycredentialcreationoptions-authenticatorselectionReference
7480 d in:
7481 * 5.1.3. Create a new credential - PublicKeyCredential's
7482 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
7483 (4) (5) (6)
7484 * 5.4. Options for Credential Creation (dictionary
7485 PublicKeyCredentialCreationOptions)
7486 * 6.2.2. The authenticatorMakeCredential operation
7487
7488 #dom-publickeycredentialcreationoptions-attestationReferenced in:
7489 * 5.1.3. Create a new credential - PublicKeyCredential's
7490 [[Create]](origin, options, sameOriginWithAncestors) method
7491 * 5.4. Options for Credential Creation (dictionary
7492 PublicKeyCredentialCreationOptions)
7493
7494 #dom-publickeycredentialcreationoptions-extensionsReferenced in:
7495 * 5.1.3. Create a new credential - PublicKeyCredential's
7496 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7497 * 5.4. Options for Credential Creation (dictionary
7498 PublicKeyCredentialCreationOptions)
7499 * 7.1. Registering a new credential (2)
7500 * 7.2. Verifying an authentication assertion
7501 * 9.3. Extending request parameters
7502
7503 #dictdef-publickeycredentialentityReferenced in:
7504 * 5.4.1. Public Key Entity Description (dictionary
7505 PublicKeyCredentialEntity) (2) (3)
7506 * 5.4.2. RP Parameters for Credential Generation (dictionary
7507 PublicKeyCredentialRpEntity)
7508 * 5.4.3. User Account Parameters for Credential Generation
7509 (dictionary PublicKeyCredentialUserEntity)
7510
7511 #dom-publickeycredentialentity-nameReferenced in:
7512 * 5.4. Options for Credential Creation (dictionary
7513 PublicKeyCredentialCreationOptions) (2)
7514 * 5.4.1. Public Key Entity Description (dictionary
7515 PublicKeyCredentialEntity) (2) (3) (4)
7516 * 6.2.2. The authenticatorMakeCredential operation (2)
7517
7518 #dom-publickeycredentialentity-iconReferenced in:
7519 * 5.4.1. Public Key Entity Description (dictionary
7520 PublicKeyCredentialEntity)
7521
7522 #dictdef-publickeycredentialrpentityReferenced in:
7523 * 5.4. Options for Credential Creation (dictionary
7524 PublicKeyCredentialCreationOptions) (2)
7525 * 5.4.1. Public Key Entity Description (dictionary
7526 PublicKeyCredentialEntity)
7527 * 5.4.2. RP Parameters for Credential Generation (dictionary
7528 PublicKeyCredentialRpEntity) (2)
7529 * 6.2.2. The authenticatorMakeCredential operation
7530
7531 #dom-publickeycredentialrpentity-idReferenced in:
7532 * 5.1.3. Create a new credential - PublicKeyCredential's
7533 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
7534 (4) (5)
7535 * 5.4. Options for Credential Creation (dictionary
7536 PublicKeyCredentialCreationOptions)
7537 * 5.4.2. RP Parameters for Credential Generation (dictionary
7538 PublicKeyCredentialRpEntity)
7539 * 6.2.2. The authenticatorMakeCredential operation (2) (3) (4)
7540
7541 #dictdef-publickeycredentialuserentityReferenced in:
7542 * 5.4. Options for Credential Creation (dictionary
7543 PublicKeyCredentialCreationOptions) (2)
7544 * 5.4.1. Public Key Entity Description (dictionary
7545 PublicKeyCredentialEntity) (2)
7546 * 5.4.3. User Account Parameters for Credential Generation

```

6574 (dictionary PublicKeyCredentialUserEntity) (2)
6575 * 6.2.1. The authenticatorMakeCredential operation
6576
6577 #dom-publickeycredentialuserentity-idReferenced in:
6578 * 5.4. Options for Credential Creation (dictionary
6579 MakePublicKeyCredentialOptions)
6580 * 5.4.3. User Account Parameters for Credential Generation
6581 (dictionary PublicKeyCredentialUserEntity)
6582 * 6.2.1. The authenticatorMakeCredential operation
6583
6584 #dom-publickeycredentialuserentity-displaynameReferenced in:
6585 * 4. Terminology
6586 * 5.4. Options for Credential Creation (dictionary
6587 MakePublicKeyCredentialOptions)

6588 * 5.4.3. User Account Parameters for Credential Generation
6589 (dictionary PublicKeyCredentialUserEntity)
6590 * 6.2.1. The authenticatorMakeCredential operation
6591
6592 #dictdef-authenticatorselectioncriteriaReferenced in:
6593 * 5.4. Options for Credential Creation (dictionary
6594 MakePublicKeyCredentialOptions) (2)
6595 * 5.4.4. Authenticator Selection Criteria (dictionary
6596 AuthenticatorSelectionCriteria) (2)
6597
6598 #dom-authenticatorselectioncriteria-authenticatorattachmentReferenced
6599 in:
6600 * 5.1.3. Create a new credential - PublicKeyCredential's
6601 [[Create]](origin, options, sameOriginWithAncestors) method
6602 * 5.4.4. Authenticator Selection Criteria (dictionary
6603 AuthenticatorSelectionCriteria)
6604
6605 #dom-authenticatorselectioncriteria-requiresresidentkeyReferenced in:
6606 * 5.1.3. Create a new credential - PublicKeyCredential's
6607 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6608 * 5.4.4. Authenticator Selection Criteria (dictionary
6609 AuthenticatorSelectionCriteria)
6610 * 6.2.1. The authenticatorMakeCredential operation
6611
6612 #dom-authenticatorselectioncriteria-userverificationReferenced in:
6613 * 5.1.3. Create a new credential - PublicKeyCredential's
6614 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6615 * 5.4.4. Authenticator Selection Criteria (dictionary
6616 AuthenticatorSelectionCriteria)
6617
6618 #enumdef-authenticatorattachmentReferenced in:
6619 * 5.4.4. Authenticator Selection Criteria (dictionary
6620 AuthenticatorSelectionCriteria) (2)
6621 * 5.4.5. Authenticator Attachment enumeration (enum
6622 AuthenticatorAttachment) (2)
6623
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
6635 #roaming-authenticatorsReferenced in:
6636 * 1.1.3. Other use cases and configurations

7547 (dictionary PublicKeyCredentialUserEntity) (2)
7548 * 6.2.2. The authenticatorMakeCredential operation
7549
7550 #dom-publickeycredentialuserentity-idReferenced in:
7551 * 5.4. Options for Credential Creation (dictionary
7552 PublicKeyCredentialCreationOptions)
7553 * 5.4.3. User Account Parameters for Credential Generation
7554 (dictionary PublicKeyCredentialUserEntity)
7555 * 6.2.2. The authenticatorMakeCredential operation
7556
7557 #dom-publickeycredentialuserentity-displaynameReferenced in:
7558 * 4. Terminology
7559 * 5.4. Options for Credential Creation (dictionary
7560 PublicKeyCredentialCreationOptions)
7561 * 5.4.1. Public Key Entity Description (dictionary
7562 PublicKeyCredentialEntity)
7563 * 5.4.3. User Account Parameters for Credential Generation
7564 (dictionary PublicKeyCredentialUserEntity) (2) (3)
7565 * 6.2.2. The authenticatorMakeCredential operation
7566
7567 #dictdef-authenticatorselectioncriteriaReferenced in:
7568 * 5.4. Options for Credential Creation (dictionary
7569 PublicKeyCredentialCreationOptions) (2)
7570 * 5.4.4. Authenticator Selection Criteria (dictionary
7571 AuthenticatorSelectionCriteria) (2)
7572
7573 #dom-authenticatorselectioncriteria-authenticatorattachmentReferenced
7574 in:
7575 * 5.1.3. Create a new credential - PublicKeyCredential's
7576 [[Create]](origin, options, sameOriginWithAncestors) method
7577 * 5.4.4. Authenticator Selection Criteria (dictionary
7578 AuthenticatorSelectionCriteria)
7579
7580 #dom-authenticatorselectioncriteria-requiresresidentkeyReferenced in:
7581 * 5.1.3. Create a new credential - PublicKeyCredential's
7582 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7583 * 5.4.4. Authenticator Selection Criteria (dictionary
7584 AuthenticatorSelectionCriteria)
7585 * 6.2.2. The authenticatorMakeCredential operation
7586
7587 #dom-authenticatorselectioncriteria-userverificationReferenced in:
7588 * 5.1.3. Create a new credential - PublicKeyCredential's
7589 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7590 * 5.4.4. Authenticator Selection Criteria (dictionary
7591 AuthenticatorSelectionCriteria)
7592
7593 #enumdef-authenticatorattachmentReferenced in:
7594 * 5.4.4. Authenticator Selection Criteria (dictionary
7595 AuthenticatorSelectionCriteria) (2)
7596 * 5.4.5. Authenticator Attachment enumeration (enum
7597 AuthenticatorAttachment) (2)
7598
7599 #attachment-modalityReferenced in:
7600 * 5.4.5. Authenticator Attachment enumeration (enum
7601 AuthenticatorAttachment) (2)
7602
7603 #platform-authenticatorsReferenced in:
7604 * 5.1.7. Availability of User-Verifying Platform Authenticator -
7605 PublicKeyCredential's
7606 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)
7607 (5)
7608 * 5.4.5. Authenticator Attachment enumeration (enum
7609 AuthenticatorAttachment) (2)
7610 * 12.1. Registration
7611 * 12.2. Registration Specifically with User Verifying Platform
7612 Authenticator (2)
7613 * 14.2. Registration Ceremony Privacy
7614
7615 #roaming-authenticatorsReferenced in:
7616 * 1.1.3. Other use cases and configurations

6637 * 5.4.5. Authenticator Attachment enumeration (enum
6638 AuthenticatorAttachment) (2)
6639 * 12.1. Registration
6640
6641 #platform-attachmentReferenced in:
6642 * 5.4.5. Authenticator Attachment enumeration (enum
6643 AuthenticatorAttachment)
6644
6645
6646 #cross-platform-attachedReferenced in:
6647 * 5.4.5. Authenticator Attachment enumeration (enum
6648 AuthenticatorAttachment) (2)
6649
6650
6651 #attestation-conveyanceReferenced in:
6652 * 4. Terminology
6653 * 5.4. Options for Credential Creation (dictionary
6654 MakePublicKeyCredentialOptions)
6655 * 5.4.6. Attestation Conveyance Preference enumeration (enum
6656 AttestationConveyancePreference)
6657
6658 #enumdef-attestationconveyancepreferenceReferenced in:
6659 * 5.4. Options for Credential Creation (dictionary
6660 MakePublicKeyCredentialOptions) (2)
6661 * 5.4.6. Attestation Conveyance Preference enumeration (enum
6662 AttestationConveyancePreference) (2)
6663
6664 #dom-attestationconveyancepreference-noneReferenced in:
6665 * 5.4. Options for Credential Creation (dictionary
6666 MakePublicKeyCredentialOptions)
6667 * 5.4.6. Attestation Conveyance Preference enumeration (enum
6668 AttestationConveyancePreference)
6669
6670 #dom-attestationconveyancepreference-indirectReferenced in:
6671 * 5.4.6. Attestation Conveyance Preference enumeration (enum
6672 AttestationConveyancePreference)
6673
6674 #dom-attestationconveyancepreference-directReferenced in:
6675 * 5.4.6. Attestation Conveyance Preference enumeration (enum
6676 AttestationConveyancePreference)
6677
6678 #dictdef-publickeycredentialrequestoptionsReferenced in:
6679 * 5.1.2. CredentialRequestOptions Extension
6680 * 5.1.4.1. PublicKeyCredential's
6681 [[DiscoverFromExternalSource]](origin, options,
6682 sameOriginWithAncestors) method
6683 * 5.5. Options for Assertion Generation (dictionary
6684 PublicKeyCredentialRequestOptions) (2)
6685 * 7.2. Verifying an authentication assertion
6686
6687 #dom-publickeycredentialrequestoptions-challengeReferenced in:
6688 * 5.1.4.1. PublicKeyCredential's
6689 [[DiscoverFromExternalSource]](origin, options,
6690 sameOriginWithAncestors) method
6691 * 5.5. Options for Assertion Generation (dictionary
6692 PublicKeyCredentialRequestOptions) (2)
6693 * 13.1. Cryptographic Challenges
6694
6695 #dom-publickeycredentialrequestoptions-timeoutReferenced in:
6696 * 5.1.4.1. PublicKeyCredential's
6697 [[DiscoverFromExternalSource]](origin, options,
6698 sameOriginWithAncestors) method (2)
6699 * 5.5. Options for Assertion Generation (dictionary
6700 PublicKeyCredentialRequestOptions)

7617 * 5.4.5. Authenticator Attachment enumeration (enum
7618 AuthenticatorAttachment) (2)
7619 * 12.1. Registration
7620
7621 #platform-attachmentReferenced in:
7622 * 5.4.5. Authenticator Attachment enumeration (enum
7623 AuthenticatorAttachment)
7624
7625 #platform-credentialReferenced in:
7626 * 5.4.5. Authenticator Attachment enumeration (enum
7627 AuthenticatorAttachment) (2)
7628
7629 #cross-platform-attachedReferenced in:
7630 * 5.4.5. Authenticator Attachment enumeration (enum
7631 AuthenticatorAttachment) (2)
7632
7633 #roaming-credentialReferenced in:
7634 * 5.4.5. Authenticator Attachment enumeration (enum
7635 AuthenticatorAttachment) (2)
7636
7637 #attestation-conveyanceReferenced in:
7638 * 4. Terminology
7639 * 5.4. Options for Credential Creation (dictionary
7640 PublicKeyCredentialCreationOptions)
7641 * 5.4.6. Attestation Conveyance Preference enumeration (enum
7642 AttestationConveyancePreference)
7643
7644 #enumdef-attestationconveyancepreferenceReferenced in:
7645 * 5.4. Options for Credential Creation (dictionary
7646 PublicKeyCredentialCreationOptions) (2)
7647 * 5.4.6. Attestation Conveyance Preference enumeration (enum
7648 AttestationConveyancePreference) (2)
7649
7650 #dom-attestationconveyancepreference-noneReferenced in:
7651 * 5.4. Options for Credential Creation (dictionary
7652 PublicKeyCredentialCreationOptions)
7653 * 5.4.6. Attestation Conveyance Preference enumeration (enum
7654 AttestationConveyancePreference)
7655
7656 #dom-attestationconveyancepreference-indirectReferenced in:
7657 * 5.4.6. Attestation Conveyance Preference enumeration (enum
7658 AttestationConveyancePreference)
7659
7660 #dom-attestationconveyancepreference-directReferenced in:
7661 * 5.4.6. Attestation Conveyance Preference enumeration (enum
7662 AttestationConveyancePreference)
7663
7664 #dictdef-publickeycredentialrequestoptionsReferenced in:
7665 * 5.1.2. CredentialRequestOptions Dictionary Extension
7666 * 5.1.4.1. PublicKeyCredential's
7667 [[DiscoverFromExternalSource]](origin, options,
7668 sameOriginWithAncestors) method
7669 * 5.5. Options for Assertion Generation (dictionary
7670 PublicKeyCredentialRequestOptions) (2)
7671 * 7.2. Verifying an authentication assertion
7672
7673 #dom-publickeycredentialrequestoptions-challengeReferenced in:
7674 * 5.1.4.1. PublicKeyCredential's
7675 [[DiscoverFromExternalSource]](origin, options,
7676 sameOriginWithAncestors) method
7677 * 5.5. Options for Assertion Generation (dictionary
7678 PublicKeyCredentialRequestOptions) (2)
7679 * 13.1. Cryptographic Challenges
7680
7681 #dom-publickeycredentialrequestoptions-timeoutReferenced in:
7682 * 5.1.4.1. PublicKeyCredential's
7683 [[DiscoverFromExternalSource]](origin, options,
7684 sameOriginWithAncestors) method (2)
7685 * 5.5. Options for Assertion Generation (dictionary
7686 PublicKeyCredentialRequestOptions)

```

6699 #dom-publickeycredentialrequestoptions-rpidReferenced in:
6700 * 5.1.4.1. PublicKeyCredential's
6701 [[DiscoverFromExternalSource]](origin, options,
6702 sameOriginWithAncestors) method (2) (3) (4)
6703 * 5.5. Options for Assertion Generation (dictionary
6704 PublicKeyCredentialRequestOptions)
6705 * 10.1. FIDO AppID Extension (appid)
6706
6707 #dom-publickeycredentialrequestoptions-allowcredentialsReferenced in:
6708 * 5.1.4.1. PublicKeyCredential's
6709 [[DiscoverFromExternalSource]](origin, options,
6710 sameOriginWithAncestors) method (2) (3) (4)
6711 * 5.5. Options for Assertion Generation (dictionary
6712 PublicKeyCredentialRequestOptions)
6713
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 #typedefdef-authenticationextensionsReferenced in:
6730
6731
6732
6733
6734
6735
6736
6737
6738
6739
6740
6741
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,

```

```

* 5.1. PublicKeyCredential Interface
* 5.1.3. Create a new credential - PublicKeyCredential's
[[Create]](origin, options, sameOriginWithAncestors) method
* 5.1.4.1. PublicKeyCredential's
[[DiscoverFromExternalSource]](origin, options,
sameOriginWithAncestors) method
* 5.4. Options for Credential Creation (dictionary
MakePublicKeyCredentialOptions) (2)
* 5.5. Options for Assertion Generation (dictionary
PublicKeyCredentialRequestOptions) (2)
* 5.8.1. Client data used in WebAuthn signatures (dictionary
CollectedClientData) (2)

```

```

7687 #dom-publickeycredentialrequestoptions-rpidReferenced in:
7688 * 5.1.4.1. PublicKeyCredential's
7689 [[DiscoverFromExternalSource]](origin, options,
7690 sameOriginWithAncestors) method (2) (3) (4)
7691 * 5.5. Options for Assertion Generation (dictionary
7692 PublicKeyCredentialRequestOptions)
7693
7694
7695 #dom-publickeycredentialrequestoptions-allowcredentialsReferenced in:
7696 * 5.1.4.1. PublicKeyCredential's
7697 [[DiscoverFromExternalSource]](origin, options,
7698 sameOriginWithAncestors) method (2) (3) (4)
7699 * 5.5. Options for Assertion Generation (dictionary
7700 PublicKeyCredentialRequestOptions)
7701 * 7.2. Verifying an authentication assertion (2)
7702 * 14.3. Authentication Ceremony Privacy (2)
7703
7704 #dom-publickeycredentialrequestoptions-userverificationReferenced in:
7705 * 5.1.4.1. PublicKeyCredential's
7706 [[DiscoverFromExternalSource]](origin, options,
7707 sameOriginWithAncestors) method (2)
7708 * 5.5. Options for Assertion Generation (dictionary
7709 PublicKeyCredentialRequestOptions)
7710
7711 #dom-publickeycredentialrequestoptions-extensionsReferenced in:
7712 * 5.1.4.1. PublicKeyCredential's
7713 [[DiscoverFromExternalSource]](origin, options,
7714 sameOriginWithAncestors) method (2)
7715 * 5.5. Options for Assertion Generation (dictionary
7716 PublicKeyCredentialRequestOptions)
7717 * 7.2. Verifying an authentication assertion
7718
7719 #dictdef-authenticationextensionsclientinputsReferenced in:
7720 * 5.4. Options for Credential Creation (dictionary
7721 PublicKeyCredentialCreationOptions) (2)
7722 * 5.5. Options for Assertion Generation (dictionary
7723 PublicKeyCredentialRequestOptions) (2)
7724 * 10.1. FIDO AppID Extension (appid)
7725 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
7726 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
7727 * 10.4. Authenticator Selection Extension (authnSel)
7728 * 10.5. Supported Extensions Extension (exts)
7729 * 10.6. User Verification Index Extension (uvi)
7730 * 10.7. Location Extension (loc)
7731 * 10.8. User Verification Method Extension (uvm)
7732
7733 #dictdef-authenticationextensionsclientoutputsReferenced in:
7734 * 5.1. PublicKeyCredential Interface
7735 * 5.1.3. Create a new credential - PublicKeyCredential's
7736 [[Create]](origin, options, sameOriginWithAncestors) method
7737 * 5.1.4.1. PublicKeyCredential's
7738 [[DiscoverFromExternalSource]](origin, options,
7739 sameOriginWithAncestors) method
7740 * 7.1. Registering a new credential
7741 * 7.2. Verifying an authentication assertion
7742 * 10.1. FIDO AppID Extension (appid)
7743 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
7744 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
7745 * 10.4. Authenticator Selection Extension (authnSel)
7746 * 10.5. Supported Extensions Extension (exts)
7747 * 10.6. User Verification Index Extension (uvi)
7748 * 10.7. Location Extension (loc)
7749 * 10.8. User Verification Method Extension (uvm)
7750
7751 #dictdef-collectedclientdataReferenced in:
7752 * 5.1.3. Create a new credential - PublicKeyCredential's
7753 [[Create]](origin, options, sameOriginWithAncestors) method
7754 * 5.1.4.1. PublicKeyCredential's
7755 [[DiscoverFromExternalSource]](origin, options,

```

674E sameOriginWithAncestors) method
 674F * 5.8.1. Client data used in WebAuthn signatures (dictionary
 675C 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
 676C * 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)
 677C * 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
 678C 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
 679C * 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
 6799 [[DiscoverFromExternalSource]](origin, options,
 680C 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
 681C [[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

775E sameOriginWithAncestors) method
 775F * 5.10.1. Client data used in WebAuthn signatures (dictionary
 775E CollectedClientData) (2)
 7759
 776C #client-dataReferenced in:
 7761 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
 7762 * 6. WebAuthn Authenticator Model (2) (3) (4)
 7763 * 6.1. Authenticator data (2)
 7764 * 7.1. Registering a new credential
 7765 * 7.2. Verifying an authentication assertion
 7766 * 9. WebAuthn Extensions
 7767 * 9.4. Client extension processing
 7768
 7769 #dictdef-tokenbindingReferenced in:
 777C * 5.10.1. Client data used in WebAuthn signatures (dictionary
 7771 CollectedClientData)
 7772
 7773 #dom-tokenbinding-statusReferenced in:
 7774 * 7.1. Registering a new credential
 7775 * 7.2. Verifying an authentication assertion
 7776
 7777 #dom-tokenbinding-idReferenced in:
 7778
 7779 * 7.1. Registering a new credential
 777E * 7.2. Verifying an authentication assertion
 778C
 7781 #enumdef-tokenbindingstatusReferenced in:
 7782 * 5.10.1. Client data used in WebAuthn signatures (dictionary
 7783 CollectedClientData)
 7784
 7785 #dom-collectedclientdata-typeReferenced in:
 7786 * 5.1.3. Create a new credential - PublicKeyCredential's
 7787 [[Create]](origin, options, sameOriginWithAncestors) method
 7788 * 5.1.4.1. PublicKeyCredential's
 7789 [[DiscoverFromExternalSource]](origin, options,
 779C sameOriginWithAncestors) method
 7791 * 5.10.1. Client data used in WebAuthn signatures (dictionary
 7792 CollectedClientData)
 7793 * 7.1. Registering a new credential
 7794 * 7.2. Verifying an authentication assertion
 7795
 7796 #dom-collectedclientdata-challengeReferenced in:
 7797 * 5.1.3. Create a new credential - PublicKeyCredential's
 7798 [[Create]](origin, options, sameOriginWithAncestors) method
 7799 * 5.1.4.1. PublicKeyCredential's
 780C [[DiscoverFromExternalSource]](origin, options,
 7801 sameOriginWithAncestors) method
 7802 * 5.10.1. Client data used in WebAuthn signatures (dictionary
 7803 CollectedClientData)
 7804 * 7.1. Registering a new credential
 7805 * 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)
6869 * 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
6878 #enumdef-publickeycredentialtypeReferenced in:
6879
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

```

```

7806
7807 #dom-collectedclientdata-originReferenced in:
7808 * 5.1.3. Create a new credential - PublicKeyCredential's
7809 [[Create]](origin, options, sameOriginWithAncestors) method
7810 * 5.1.4.1. PublicKeyCredential's
7811 [[DiscoverFromExternalSource]](origin, options,
7812 sameOriginWithAncestors) method
7813 * 5.10.1. Client data used in WebAuthn signatures (dictionary
7814 CollectedClientData)
7815 * 7.1. Registering a new credential
7816 * 7.2. Verifying an authentication assertion
7817
7818 #dom-collectedclientdata-tokenbindingReferenced in:
7819 * 5.1.3. Create a new credential - PublicKeyCredential's
7820 [[Create]](origin, options, sameOriginWithAncestors) method
7821 * 5.1.4.1. PublicKeyCredential's
7822 [[DiscoverFromExternalSource]](origin, options,
7823 sameOriginWithAncestors) method
7824 * 5.10.1. Client data used in WebAuthn signatures (dictionary
7825 CollectedClientData)
7826 * 7.1. Registering a new credential (2)
7827 * 7.2. Verifying an authentication assertion (2)
7828
7829 #collectedclientdata-json-serialized-client-dataReferenced in:
7830 * 5.1.3. Create a new credential - PublicKeyCredential's
7831 [[Create]](origin, options, sameOriginWithAncestors) method
7832 * 5.1.4.1. PublicKeyCredential's
7833 [[DiscoverFromExternalSource]](origin, options,
7834 sameOriginWithAncestors) method
7835 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
7836 * 5.2.1. Information about Public Key Credential (interface
7837 AuthenticatorAttestationResponse) (2)
7838 * 5.2.2. Web Authentication Assertion (interface
7839 AuthenticatorAssertionResponse)
7840 * 5.10.1. Client data used in WebAuthn signatures (dictionary
7841 CollectedClientData)
7842
7843 #collectedclientdata-hash-of-the-serialized-client-dataReferenced in:
7844 * 5.1.3. Create a new credential - PublicKeyCredential's
7845 [[Create]](origin, options, sameOriginWithAncestors) method
7846 * 5.1.4.1. PublicKeyCredential's
7847 [[DiscoverFromExternalSource]](origin, options,
7848 sameOriginWithAncestors) method
7849 * 5.2.1. Information about Public Key Credential (interface
7850 AuthenticatorAttestationResponse)
7851 * 5.2.2. Web Authentication Assertion (interface
7852 AuthenticatorAssertionResponse)
7853 * 6. WebAuthn Authenticator Model
7854 * 6.2.2. The authenticatorMakeCredential operation
7855 * 6.2.3. The authenticatorGetAssertion operation (2)
7856
7857 * 6.3.2. Attestation Statement Formats (2)
7858 * 6.3.4. Generating an Attestation Object
7859 * 7.1. Registering a new credential
7860 * 8.2. Packed Attestation Statement Format
7861 * 8.3. TPM Attestation Statement Format
7862 * 8.4. Android Key Attestation Statement Format
7863 * 8.5. Android SafetyNet Attestation Statement Format
7864 * 8.6. FIDO U2F Attestation Statement Format
7865
7866 #enumdef-publickeycredentialtypeReferenced in:
7867
7868 * 4. Terminology
7869 * 5.1.3. Create a new credential - PublicKeyCredential's
7870 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7871 * 5.3. Parameters for Credential Generation (dictionary
7872 PublicKeyCredentialParameters)
7873 * 5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)
7874 * 5.10.3. Credential Descriptor (dictionary

```

```

6885 PublicKeyCredentialDescriptor)
6886 * 6.2.1. The authenticatorMakeCredential operation (2) (3)
6887
6888 #dom-publickeycredentialtype-public-keyReferenced in:
6889 * 5.8.2. Credential Type enumeration (enum PublicKeyCredentialType)
6890
6891 #dictdef-publickeycredentialdescriptorReferenced in:
6892 * 5.1.4.1. PublicKeyCredential's
6893 [[DiscoverFromExternalSource]](origin, options,
6894 sameOriginWithAncestors) method
6895 * 5.4. Options for Credential Creation (dictionary
6896 MakePublicKeyCredentialOptions) (2)
6897 * 5.5. Options for Assertion Generation (dictionary
6898 PublicKeyCredentialRequestOptions) (2) (3)
6899 * 5.8.3. Credential Descriptor (dictionary
6900 PublicKeyCredentialDescriptor)
6901 * 6.2.1. The authenticatorMakeCredential operation
6902 * 6.2.2. The authenticatorGetAssertion operation
6903
6904 #dom-publickeycredentialdescriptor-transportReferenced in:
6905 * 5.1.3. Create a new credential - PublicKeyCredential's
6906 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6907 * 5.1.4.1. PublicKeyCredential's
6908 [[DiscoverFromExternalSource]](origin, options,
6909 sameOriginWithAncestors) method (2)
6910
6911 #dom-publickeycredentialdescriptor-typeReferenced in:
6912 * 5.1.4.1. PublicKeyCredential's
6913 [[DiscoverFromExternalSource]](origin, options,
6914 sameOriginWithAncestors) method
6915 * 5.8.3. Credential Descriptor (dictionary
6916 PublicKeyCredentialDescriptor)
6917 * 6.2.1. The authenticatorMakeCredential operation
6918 * 6.2.2. The authenticatorGetAssertion operation
6919
6920 #dom-publickeycredentialdescriptor-idReferenced in:
6921 * 5.1.4.1. PublicKeyCredential's
6922 [[DiscoverFromExternalSource]](origin, options,
6923 sameOriginWithAncestors) method (2)
6924 * 5.8.3. Credential Descriptor (dictionary
6925 PublicKeyCredentialDescriptor)
6926 * 6.2.1. The authenticatorMakeCredential operation
6927 * 6.2.2. The authenticatorGetAssertion operation
6928
6929 #enumdef-authenticatortransportReferenced in:
6930 * 5.8.3. Credential Descriptor (dictionary
6931 PublicKeyCredentialDescriptor)
6932 * 5.8.4. Authenticator Transport enumeration (enum
6933 AuthenticatorTransport)
6934
6935 #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

```

```

7873 PublicKeyCredentialDescriptor)
7874 * 6.2.2. The authenticatorMakeCredential operation (2) (3)
7875
7876 #dom-publickeycredentialtype-public-keyReferenced in:
7877 * 4. Terminology
7878 * 5.10.2. Credential Type enumeration (enum PublicKeyCredentialType)
7879 * 6.2.2. The authenticatorMakeCredential operation
7880
7881 #dictdef-publickeycredentialdescriptorReferenced in:
7882 * 5.1.4.1. PublicKeyCredential's
7883 [[DiscoverFromExternalSource]](origin, options,
7884 sameOriginWithAncestors) method
7885 * 5.4. Options for Credential Creation (dictionary
7886 PublicKeyCredentialCreationOptions) (2)
7887 * 5.5. Options for Assertion Generation (dictionary
7888 PublicKeyCredentialRequestOptions) (2) (3)
7889 * 5.10.3. Credential Descriptor (dictionary
7890 PublicKeyCredentialDescriptor)
7891 * 6.2.2. The authenticatorMakeCredential operation
7892 * 6.2.3. The authenticatorGetAssertion operation
7893
7894 #dom-publickeycredentialdescriptor-transportReferenced in:
7895 * 5.1.3. Create a new credential - PublicKeyCredential's
7896 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7897 * 5.1.4.1. PublicKeyCredential's
7898 [[DiscoverFromExternalSource]](origin, options,
7899 sameOriginWithAncestors) method (2)
7900
7901 #dom-publickeycredentialdescriptor-typeReferenced in:
7902 * 5.1.4.1. PublicKeyCredential's
7903 [[DiscoverFromExternalSource]](origin, options,
7904 sameOriginWithAncestors) method
7905 * 5.10.3. Credential Descriptor (dictionary
7906 PublicKeyCredentialDescriptor)
7907 * 6.2.2. The authenticatorMakeCredential operation
7908
7909 #dom-publickeycredentialdescriptor-idReferenced in:
7910 * 5.1.4.1. PublicKeyCredential's
7911 [[DiscoverFromExternalSource]](origin, options,
7912 sameOriginWithAncestors) method (2)
7913 * 5.10.3. Credential Descriptor (dictionary
7914 PublicKeyCredentialDescriptor)
7915 * 6.2.2. The authenticatorMakeCredential operation
7916 * 6.2.3. The authenticatorGetAssertion operation
7917
7918 #enumdef-authenticatortransportReferenced in:
7919 * 5.10.3. Credential Descriptor (dictionary
7920 PublicKeyCredentialDescriptor)
7921 * 5.10.4. Authenticator Transport enumeration (enum
7922 AuthenticatorTransport)
7923
7924 #dom-authenticatortransport-usbReferenced in:
7925 * 5.10.4. Authenticator Transport enumeration (enum
7926 AuthenticatorTransport)
7927
7928 #dom-authenticatortransport-nfcReferenced in:
7929 * 5.10.4. Authenticator Transport enumeration (enum
7930 AuthenticatorTransport)
7931
7932 #dom-authenticatortransport-bleReferenced in:
7933 * 5.10.4. Authenticator Transport enumeration (enum
7934 AuthenticatorTransport)
7935
7936 #typedefdef-cosealgorithmidentifierReferenced in:
7937 * 5.1.3. Create a new credential - PublicKeyCredential's
7938 [[Create]](origin, options, sameOriginWithAncestors) method
7939 * 5.3. Parameters for Credential Generation (dictionary
7940 PublicKeyCredentialParameters)
7941 * 5.10.5. 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
6995 #attestation-signatureReferenced in:
6996 * 4. Terminology
6997 * 6. WebAuthn Authenticator model (2) (3)
6998 * 6.3. Attestation
6999 * 7.1. Registering a new credential
7000 * 8.6. FIDO U2F Attestation Statement Format
7001
7002 #assertion-signatureReferenced in:
7003 * 6. WebAuthn Authenticator model (2)
7004 * 6.2.2. The authenticatorGetAssertion operation (2) (3)
7005
7006 #authenticator-dataReferenced in:
7007 * 5.1.4.1. PublicKeyCredential's
7008 [[DiscoverFromExternalSource]](origin, options,
7009 sameOriginWithAncestors) method
7010 * 5.2.1. Information about Public Key Credential (interface
7011 AuthenticatorAttestationResponse) (2)
7012 * 5.2.2. Web Authentication Assertion (interface
7013 AuthenticatorAssertionResponse)
7014 * 6. WebAuthn Authenticator model (2)
7015 * 6.1. Authenticator data (2) (3) (4) (5) (6) (7) (8) (9)

```

```

7942 COSEAlgorithmIdentifier)
7943 * 6.2.2. The authenticatorMakeCredential operation
7944 * 6.3.1. Attested credential data
7945 * 8.2. Packed Attestation Statement Format
7946 * 8.3. TPM Attestation Statement Format
7947
7948 #enumdef-userverificationrequirementReferenced in:
7949 * 5.4.4. Authenticator Selection Criteria (dictionary
7950 AuthenticatorSelectionCriteria) (2)
7951 * 5.5. Options for Assertion Generation (dictionary
7952 PublicKeyCredentialRequestOptions) (2)
7953 * 5.10.6. User Verification Requirement enumeration (enum
7954 UserVerificationRequirement)
7955
7956 #dom-userverificationrequirement-requiredReferenced in:
7957 * 5.1.3. Create a new credential - PublicKeyCredential's
7958 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7959 * 5.1.4.1. PublicKeyCredential's
7960 [[DiscoverFromExternalSource]](origin, options,
7961 sameOriginWithAncestors) method (2)
7962 * 5.10.6. User Verification Requirement enumeration (enum
7963 UserVerificationRequirement)
7964
7965 #dom-userverificationrequirement-preferredReferenced in:
7966 * 5.1.3. Create a new credential - PublicKeyCredential's
7967 [[Create]](origin, options, sameOriginWithAncestors) method
7968 * 5.1.4.1. PublicKeyCredential's
7969 [[DiscoverFromExternalSource]](origin, options,
7970 sameOriginWithAncestors) method
7971 * 5.10.6. User Verification Requirement enumeration (enum
7972 UserVerificationRequirement)
7973
7974 #dom-userverificationrequirement-discouragedReferenced in:
7975 * 5.1.3. Create a new credential - PublicKeyCredential's
7976 [[Create]](origin, options, sameOriginWithAncestors) method
7977 * 5.1.4.1. PublicKeyCredential's
7978 [[DiscoverFromExternalSource]](origin, options,
7979 sameOriginWithAncestors) method
7980 * 5.10.6. User Verification Requirement enumeration (enum
7981 UserVerificationRequirement)
7982
7983 #authenticator-modelReferenced in:
7984 * 6. WebAuthn Authenticator Model
7985
7986 #authenticator-credentials-mapReferenced in:
7987 * 6.2.1. Lookup Credential Source by Credential ID algorithm
7988 * 6.2.2. The authenticatorMakeCredential operation
7989 * 6.2.3. The authenticatorGetAssertion operation
7990
7991 #attestation-signatureReferenced in:
7992 * 4. Terminology
7993 * 6. WebAuthn Authenticator Model (2) (3)
7994 * 6.3. Attestation
7995 * 7.1. Registering a new credential
7996 * 8.6. FIDO U2F Attestation Statement Format
7997
7998 #assertion-signatureReferenced in:
7999 * 6. WebAuthn Authenticator Model (2)
8000 * 6.2.3. The authenticatorGetAssertion operation (2) (3)
8001
8002 #authenticator-dataReferenced in:
8003 * 5.1.4.1. PublicKeyCredential's
8004 [[DiscoverFromExternalSource]](origin, options,
8005 sameOriginWithAncestors) method
8006 * 5.2.1. Information about Public Key Credential (interface
8007 AuthenticatorAttestationResponse) (2)
8008 * 5.2.2. Web Authentication Assertion (interface
8009 AuthenticatorAssertionResponse)
8010 * 6. WebAuthn Authenticator Model (2)
8011 * 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)

7030
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

7038
7039 #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

7058
7059 #signature-counterReferenced in:
7060 * 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)

8012 * 6.1.1. Signature Counter Considerations (2)
8013 * 6.2.2. The authenticatorMakeCredential operation
8014 * 6.2.3. The authenticatorGetAssertion operation
8015 * 6.3. Attestation (2)
8016 * 6.3.1. Attested credential data
8017 * 6.3.2. Attestation Statement Formats (2)
8018 * 6.3.4. Generating an Attestation Object

8019 * 7.1. Registering a new credential
8020 * 8.5. Android SafetyNet Attestation Statement Format
8021 * 9.5. Authenticator extension processing (2)

8022 * 10.6. User Verification Index Extension (uvi)

8023 * 10.8. User Verification Method Extension (uvm)
8024 * 13.2.1. Attestation Certificate Hierarchy
8025 * 13.3. credentialId Unsigned

8026
8027 #rpIdhashReferenced in:
8028 * 7.2. Verifying an authentication assertion
8029
8030 #flagsReferenced in:
8031 * 5.10.6. User Verification Requirement enumeration (enum
8032 UserVerificationRequirement) (2)
8033 * 6.1. Authenticator data
8034 * 7.1. Registering a new credential (2)
8035 * 7.2. Verifying an authentication assertion (2)

8036
8037 #signcountReferenced in:
8038 * 6.1.1. Signature Counter Considerations (2)
8039 * 7.2. Verifying an authentication assertion (2) (3)
8040
8041 #attestedcredentialdataReferenced in:
8042 * 5.1.3. Create a new credential - PublicKeyCredential's
8043 [[Create]](origin, options, sameOriginWithAncestors) method
8044 * 6.1. Authenticator data (2)
8045 * 6.2.2. The authenticatorMakeCredential operation
8046 * 6.2.3. The authenticatorGetAssertion operation
8047 * 7.1. Registering a new credential (2)
8048 * 8.3. TPM Attestation Statement Format
8049 * 8.4. Android Key Attestation Statement Format
8050 * 8.6. FIDO U2F Attestation Statement Format
8051
8052 #authdataextensionsReferenced in:
8053 * 6.1. Authenticator data
8054 * 6.2.2. The authenticatorMakeCredential operation
8055 * 6.2.3. The authenticatorGetAssertion operation
8056 * 7.1. Registering a new credential (2)
8057 * 7.2. Verifying an authentication assertion (2)
8058 * 9.5. Authenticator extension processing (2)

8059
8060 #signature-counterReferenced in:
8061 * 6.1. Authenticator data
8062 * 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6) (7) (8)
8063 (9) (10)
8064 * 6.2.2. The authenticatorMakeCredential operation (2) (3) (4)
8065 * 6.2.3. The authenticatorGetAssertion operation (2)
8066 * 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6)
8067
8068 #authenticator-operationsReferenced in:
8069 * 4. Terminology

8070
8071 #authenticator-sessionReferenced in:
8072 * 5.6. Abort operations with AbortSignal (2)
8073 * 6.2.2. The authenticatorMakeCredential operation
8074 * 6.2.3. The authenticatorGetAssertion operation
8075 * 6.2.4. The authenticatorCancel operation (2)
8076
8077 #credential-id-looking-upReferenced in:
8078 * 6.2.2. The authenticatorMakeCredential operation

```

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 * 5.2.1. Information about Public Key Credential (interface
7121 AuthenticatorAttestationResponse) (2) (3)
7122 * 5.4.6. Attestation Conveyance Preference enumeration (enum
7123 AttestationConveyancePreference) (2) (3) (4) (5) (6) (7)
7124 * 6.3. Attestation (2) (3) (4) (5) (6) (7) (8)
7125 * 6.3.2. Attestation Statement Formats (2) (3) (4)
7126 * 7.1. Registering a new credential
7127
7128 #attestation-statement-formatReferenced in:
7129 * 5.2.1. Information about Public Key Credential (interface
7130 AuthenticatorAttestationResponse)
7131 * 5.8.4. Authenticator Transport enumeration (enum
7132 AuthenticatorTransport)
7133 * 6.2.1. The authenticatorMakeCredential operation
7134 * 6.3. Attestation (2) (3) (4) (5) (6) (7)
7135 * 6.3.2. Attestation Statement Formats (2) (3) (4)
7136

```

```

8079 * 6.2.3. The authenticatorGetAssertion operation
8080
8081 #authenticatormakecredentialReferenced in:
8082 * 4. Terminology (2) (3) (4)
8083 * 5.1.3. Create a new credential - PublicKeyCredential's
8084 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8085 * 6. WebAuthn Authenticator Model
8086 * 6.2.4. The authenticatorCancel operation (2)
8087 * 9. WebAuthn Extensions
8088 * 9.2. Defining extensions
8089 * 9.5. Authenticator extension processing
8090
8091 #authenticatorgetassertionReferenced in:
8092 * 4. Terminology (2) (3)
8093 * 5.1.4.1. PublicKeyCredential's
8094 [[DiscoverFromExternalSource]](origin, options,
8095 sameOriginWithAncestors) method (2) (3) (4)
8096 * 6. WebAuthn Authenticator Model
8097 * 6.1. Authenticator data
8098 * 6.1.1. Signature Counter Considerations (2) (3)
8099 * 6.2.4. The authenticatorCancel operation (2)
8100 * 9. WebAuthn Extensions
8101 * 9.2. Defining extensions
8102 * 9.5. Authenticator extension processing
8103 * 10.1. FIDO AppID Extension (appid)
8104
8105 #authenticatorcancelReferenced in:
8106 * 5.1.3. Create a new credential - PublicKeyCredential's
8107 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
8108 (4) (5)
8109 * 5.1.4.1. PublicKeyCredential's
8110 [[DiscoverFromExternalSource]](origin, options,
8111 sameOriginWithAncestors) method (2) (3) (4)
8112 * 6.2.2. The authenticatorMakeCredential operation
8113 * 6.2.3. The authenticatorGetAssertion operation
8114
8115 #attestation-objectReferenced in:
8116 * 4. Terminology (2) (3)
8117 * 5. Web Authentication API
8118 * 5.2.1. Information about Public Key Credential (interface
8119 AuthenticatorAttestationResponse) (2)
8120 * 5.4. Options for Credential Creation (dictionary
8121 PublicKeyCredentialCreationOptions) (2)
8122 * 6.2.2. The authenticatorMakeCredential operation (2)
8123 * 6.3. Attestation (2) (3)
8124 * 6.3.1. Attested credential data
8125 * 6.3.4. Generating an Attestation Object (2)
8126 * 7.1. Registering a new credential
8127
8128 #attestation-statementReferenced in:
8129 * 4. Terminology (2)
8130 * 5.1.3. Create a new credential - PublicKeyCredential's
8131 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8132 * 5.2.1. Information about Public Key Credential (interface
8133 AuthenticatorAttestationResponse) (2) (3)
8134 * 5.4.6. Attestation Conveyance Preference enumeration (enum
8135 AttestationConveyancePreference) (2) (3) (4) (5) (6)
8136 * 6.3. Attestation (2) (3) (4) (5) (6) (7) (8)
8137 * 6.3.2. Attestation Statement Formats (2) (3) (4)
8138 * 7.1. Registering a new credential
8139 * 8.7. None Attestation Statement Format
8140
8141 #attestation-statement-formatReferenced in:
8142 * 5.2.1. Information about Public Key Credential (interface
8143 AuthenticatorAttestationResponse)
8144 * 5.10.4. Authenticator Transport enumeration (enum
8145 AuthenticatorTransport)
8146 * 6.2.2. The authenticatorMakeCredential operation
8147 * 6.3. Attestation (2) (3) (4) (5) (6) (7)
8148 * 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
7186 #authenticator-data-for-the-attestationReferenced in:
7187 * 8.2. Packed Attestation Statement Format
7188 * 8.3. TPM Attestation Statement Format
7189 * 8.4. Android Key Attestation Statement Format (2)
7190 * 8.5. Android SafetyNet Attestation Statement Format
7191 * 8.6. FIDO U2F Attestation Statement Format
7192
7193 #verification-procedure-inputsReferenced in:
7194 * 8.2. Packed Attestation Statement Format
7195 * 8.3. TPM Attestation Statement Format
7196 * 8.4. Android Key Attestation Statement Format
7197 * 8.5. Android SafetyNet Attestation Statement Format
7198 * 8.6. FIDO U2F Attestation Statement Format
7199
7200 #authenticator-data-claimed-to-have-been-used-for-the-attestationReferenced in:
7201 * 8.4. Android Key Attestation Statement Format
7202
7203 #attestation-trust-pathReferenced in:
7204 * 6.3.2. Attestation Statement Formats

```

```

8149 * 6.3.4. Generating an Attestation Object
8150 * 7.1. Registering a new credential (2)
8151
8152 #attestation-typeReferenced in:
8153 * 5.1.3. Create a new credential - PublicKeyCredential's
8154 [[Create]](origin, options, sameOriginWithAncestors) method
8155 * 6.3. Attestation (2) (3) (4) (5) (6)
8156 * 6.3.2. Attestation Statement Formats (2)
8157
8158 #attested-credential-dataReferenced in:
8159 * 5.1.3. Create a new credential - PublicKeyCredential's
8160 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8161 * 6.1. Authenticator data (2) (3) (4) (5)
8162 * 6.2.2. The authenticatorMakeCredential operation
8163 * 6.3. Attestation (2)
8164 * 6.3.1. Attested credential data
8165 * 6.3.3. Attestation Types
8166
8167 #aaguidReferenced in:
8168 * 4. Terminology
8169 * 5.1.3. Create a new credential - PublicKeyCredential's
8170 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
8171 (4)
8172
8173 * 7.1. Registering a new credential
8174 * 8.2. Packed Attestation Statement Format
8175 * 8.3. TPM Attestation Statement Format
8176
8177 #credentialidlengthReferenced in:
8178 * 6.1. Authenticator data
8179
8180 #credentialidReferenced in:
8181 * 5.1.3. Create a new credential - PublicKeyCredential's
8182 [[Create]](origin, options, sameOriginWithAncestors) method
8183 * 6.1. Authenticator data
8184 * 7.1. Registering a new credential (2)
8185
8186 #credentialpublickeyReferenced in:
8187 * 6.1. Authenticator data
8188 * 6.3.1.1. Examples of credentialPublicKey Values encoded in COSE_Key
8189 format
8190 * 7.1. Registering a new credential
8191 * 8.2. Packed Attestation Statement Format
8192 * 8.3. TPM Attestation Statement Format
8193 * 8.4. Android Key Attestation Statement Format
8194
8195 #signing-procedureReferenced in:
8196 * 6.3.2. Attestation Statement Formats
8197 * 6.3.4. Generating an Attestation Object
8198
8199 #authenticator-data-for-the-attestationReferenced in:
8200 * 8.2. Packed Attestation Statement Format
8201 * 8.3. TPM Attestation Statement Format
8202 * 8.4. Android Key Attestation Statement Format (2)
8203 * 8.5. Android SafetyNet Attestation Statement Format
8204 * 8.6. FIDO U2F Attestation Statement Format
8205
8206 #verification-procedure-inputsReferenced in:
8207 * 8.2. Packed Attestation Statement Format
8208 * 8.3. TPM Attestation Statement Format
8209 * 8.4. Android Key Attestation Statement Format
8210 * 8.5. Android SafetyNet Attestation Statement Format
8211 * 8.6. FIDO U2F Attestation Statement Format
8212
8213 #authenticator-data-claimed-to-have-been-used-for-the-attestationReferenced in:
8214 * 8.4. Android Key Attestation Statement Format
8215
8216 #attestation-trust-pathReferenced in:
8217 * 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
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-ecdaa-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)

8218 * 8.2. Packed Attestation Statement Format (2) (3)
8219 * 8.3. TPM Attestation Statement Format
8220 * 8.4. Android Key Attestation Statement Format
8221 * 8.5. Android SafetyNet Attestation Statement Format
8222 * 8.6. FIDO U2F Attestation Statement Format
8223
8224 #basic-attestationReferenced in:
8225 * 14.1. Attestation Privacy
8226
8227 #basicReferenced in:
8228 * 8.2. Packed Attestation Statement Format (2)
8229 * 8.4. Android Key Attestation Statement Format (2)
8230 * 8.5. Android SafetyNet Attestation Statement Format (2)
8231 * 8.6. FIDO U2F Attestation Statement Format (2)

8232
8233 #self-attestationReferenced in:
8234 * 4. Terminology (2) (3) (4)
8235 * 5.1.3. Create a new credential - PublicKeyCredential's
8236 [[Create]](origin, options, sameOriginWithAncestors) method
8237 * 5.4.6. Attestation Conveyance Preference enumeration (enum
8238 AttestationConveyancePreference)
8239 * 6.3. Attestation (2)
8240 * 6.3.2. Attestation Statement Formats
8241 * 6.3.3. Attestation Types
8242 * 7.1. Registering a new credential (2) (3)
8243 * 8.2. Packed Attestation Statement Format (2)
8244 * 13.2.2. Attestation Certificate and Attestation Certificate CA
8245 Compromise
8246
8247 #selfReferenced in:
8248 * 8.2. Packed Attestation Statement Format
8249
8250 #attestation-caReferenced in:
8251 * 5.4.6. Attestation Conveyance Preference enumeration (enum
8252 AttestationConveyancePreference)
8253 * 6.3.3. Attestation Types (2)
8254 * 14.1. Attestation Privacy (2)
8255
8256 #attcaReferenced in:
8257 * 8.2. Packed Attestation Statement Format
8258 * 8.3. TPM Attestation Statement Format (2)
8259 * 8.6. FIDO U2F Attestation Statement Format

8260
8261 #elliptic-curve-based-direct-anonymous-attestationReferenced in:
8262 * 14.1. Attestation Privacy
8263
8264 #ecdaaReferenced in:
8265 * 6.3.2. Attestation Statement Formats
8266 * 6.3.3. Attestation Types
8267 * 7.1. Registering a new credential
8268 * 8.2. Packed Attestation Statement Format (2) (3) (4)
8269 * 8.3. TPM Attestation Statement Format (2) (3) (4)
8270 * 13.2.2. Attestation Certificate and Attestation Certificate CA
8271 Compromise
8272
8273 #noneReferenced in:
8274 * 8.7. None Attestation Statement Format (2)

8275
8276 #attestation-statement-format-identifierReferenced in:
8277 * 6.3.2. Attestation Statement Formats
8278 * 6.3.4. Generating an Attestation Object
8279
8280 #identifier-of-the-ecdaa-issuer-public-keyReferenced in:
8281 * 7.1. Registering a new credential
8282 * 8.2. Packed Attestation Statement Format
8283 * 8.3. TPM Attestation Statement Format (2)

7255 #ecdaa-issuer-public-keyReferenced in:
7260 * 6.3.2. Attestation Statement Formats
7261 * 6.3.5.1. Privacy
7262 * 7.1. Registering a new credential
7263 * 8.2. Packed Attestation Statement Format (2) (3)
7264

7265 #registration-extensionReferenced in:
7266 * 5.1.3. Create a new credential - PublicKeyCredential's
7267 [[Create]](origin, options, sameOriginWithAncestors) method
7268 * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
7269 * 9.6. Example Extension
7270 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
7271 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
7272 * 10.4. Authenticator Selection Extension (authnSel)
7273 * 10.5. Supported Extensions Extension (exts)
7274 * 10.6. User Verification Index Extension (uvi)
7275 * 10.7. Location Extension (loc)
7276 * 10.8. User Verification Method Extension (uvm)
7277

7278 * 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
7279 (6) (7)

7280 #authentication-extensionReferenced in:
7281 * 5.1.4.1. PublicKeyCredential's
7282 [[DiscoverFromExternalSource]](origin, options,
7283 sameOriginWithAncestors) method
7284 * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
7285 * 9.6. Example Extension
7286 * 10.1. FIDO Appid Extension (appid)
7287 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
7288 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
7289 * 10.6. User Verification Index Extension (uvi)
7290 * 10.7. Location Extension (loc)
7291 * 10.8. User Verification Method Extension (uvm)
7292 * 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
7293 (6)
7294

7295 #client-extensionReferenced in:
7296 * 5.1.3. Create a new credential - PublicKeyCredential's
7297 [[Create]](origin, options, sameOriginWithAncestors) method
7298 * 5.1.4.1. PublicKeyCredential's
7299 [[DiscoverFromExternalSource]](origin, options,
7300 sameOriginWithAncestors) method
7301 * 5.7. Authentication Extensions (typedef AuthenticationExtensions)
7302 * 9. WebAuthn Extensions
7303 * 9.2. Defining extensions
7304 * 9.4. Client extension processing
7305

7306 #authenticator-extensionReferenced in:
7307 * 5.1.3. Create a new credential - PublicKeyCredential's
7308 [[Create]](origin, options, sameOriginWithAncestors) method
7309 * 5.1.4.1. PublicKeyCredential's
7310 [[DiscoverFromExternalSource]](origin, options,
7311 sameOriginWithAncestors) method
7312 * 5.7. Authentication Extensions (typedef AuthenticationExtensions)
7313 * 9. WebAuthn Extensions (2) (3)
7314 * 9.2. Defining extensions (2)
7315 * 9.3. Extending request parameters
7316 * 9.5. Authenticator extension processing
7317

7318 #extension-identifierReferenced in:
7319 * 5.1. PublicKeyCredential Interface
7320 * 5.1.3. Create a new credential - PublicKeyCredential's
7321 [[Create]](origin, options, sameOriginWithAncestors) method
7322 * 5.1.4.1. PublicKeyCredential's
7323 [[DiscoverFromExternalSource]](origin, options,
7324

8284 #ecdaa-issuer-public-keyReferenced in:
8285 * 6.3.2. Attestation Statement Formats
8286 * 7.1. Registering a new credential
8287 * 8.2. Packed Attestation Statement Format (2) (3)
8288 * 14.1. Attestation Privacy
8289

8290 #registration-extensionReferenced in:
8291 * 5.1.3. Create a new credential - PublicKeyCredential's
8292 [[Create]](origin, options, sameOriginWithAncestors) method
8293 * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
8294

8295 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
8296 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
8297 * 10.4. Authenticator Selection Extension (authnSel)
8298 * 10.5. Supported Extensions Extension (exts)
8299 * 10.6. User Verification Index Extension (uvi)
8300 * 10.7. Location Extension (loc)
8301 * 10.8. User Verification Method Extension (uvm)
8302 * 10.9. Biometric Authenticator Performance Bounds Extension
8303 (biometricPerfBounds)
8304 * 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
8305 (6) (7)

8306 #authentication-extensionReferenced in:
8307 * 5.1.4.1. PublicKeyCredential's
8308 [[DiscoverFromExternalSource]](origin, options,
8309 sameOriginWithAncestors) method
8310 * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
8311

8312 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
8313 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
8314 * 10.6. User Verification Index Extension (uvi)
8315 * 10.7. Location Extension (loc)
8316 * 10.8. User Verification Method Extension (uvm)
8317 * 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
8318 (6)
8319

8320 #client-extensionReferenced in:
8321 * 5.1.3. Create a new credential - PublicKeyCredential's
8322 [[Create]](origin, options, sameOriginWithAncestors) method
8323 * 5.1.4.1. PublicKeyCredential's
8324 [[DiscoverFromExternalSource]](origin, options,
8325 sameOriginWithAncestors) method
8326 * 9. WebAuthn Extensions
8327 * 9.2. Defining extensions
8328 * 9.4. Client extension processing
8329 * 10.1. FIDO AppID Extension (appid)
8330

8331 #authenticator-extensionReferenced in:
8332 * 5.1.3. Create a new credential - PublicKeyCredential's
8333 [[Create]](origin, options, sameOriginWithAncestors) method
8334 * 5.1.4.1. PublicKeyCredential's
8335 [[DiscoverFromExternalSource]](origin, options,
8336 sameOriginWithAncestors) method
8337

8337 * 9. WebAuthn Extensions (2) (3)
8338 * 9.2. Defining extensions (2)
8339 * 9.3. Extending request parameters
8340 * 9.5. Authenticator extension processing
8341

8342 #extension-identifierReferenced in:
8343 * 5.1. PublicKeyCredential Interface
8344 * 5.1.3. Create a new credential - PublicKeyCredential's
8345 [[Create]](origin, options, sameOriginWithAncestors) method
8346 * 5.1.4.1. PublicKeyCredential's
8347 [[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
7373 [[DiscoverFromExternalSource]](origin, options,
7374 sameOriginWithAncestors) method (2)
7375 * 9. WebAuthn Extensions (2) (3)

7376
7377 * 9.2. Defining extensions (2) (3)
7378 * 9.4. Client extension processing (2) (3)
7379 * 9.6. Example Extension
7380
7381 #authenticator-extension-processingReferenced in:
7382 * 6.2.1. The authenticatorMakeCredential operation
7383 * 6.2.2. The authenticatorGetAssertion operation
7384 * 9. WebAuthn Extensions
7385 * 9.2. Defining extensions
7386 * 9.5. Authenticator extension processing

```

```

8348 sameOriginWithAncestors) method
8349 * 6.1. Authenticator data
8350 * 6.2.2. The authenticatorMakeCredential operation (2)
8351 * 6.2.3. The authenticatorGetAssertion operation (2)
8352 * 7.1. Registering a new credential (2)
8353 * 7.2. Verifying an authentication assertion (2)
8354 * 9. WebAuthn Extensions (2)
8355 * 9.2. Defining extensions
8356 * 9.3. Extending request parameters
8357 * 9.4. Client extension processing (2)
8358 * 9.5. Authenticator extension processing (2)

8359
8360 * 10.5. Supported Extensions Extension (exts) (2)

8361
8362 * 11.2. WebAuthn Extension Identifier Registrations
8363
8364 #client-extension-inputReferenced in:
8365 * 5.7. Authentication Extensions Client Inputs (typedef
8366 AuthenticationExtensionsClientInputs)
8367 * 7.1. Registering a new credential
8368 * 7.2. Verifying an authentication assertion
8369 * 9. WebAuthn Extensions (2) (3) (4)
8370 * 9.2. Defining extensions
8371 * 9.3. Extending request parameters (2) (3) (4) (5) (6)
8372 * 9.4. Client extension processing (2) (3) (4)

8373
8374 #authenticator-extension-inputReferenced in:
8375 * 5.9. Authentication Extensions Authenticator Inputs (typedef
8376 AuthenticationExtensionsAuthenticatorInputs)
8377 * 6.2.2. The authenticatorMakeCredential operation (2)
8378 * 6.2.3. The authenticatorGetAssertion operation (2)
8379 * 9. WebAuthn Extensions (2) (3) (4) (5) (6)
8380 * 9.2. Defining extensions
8381 * 9.3. Extending request parameters (2) (3)
8382 * 9.4. Client extension processing
8383 * 9.5. Authenticator extension processing (2) (3)

8384
8385 #client-extension-processingReferenced in:
8386 * 5.1. PublicKeyCredential Interface
8387 * 5.1.3. Create a new credential - PublicKeyCredential's
8388 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8389 * 5.1.4.1. PublicKeyCredential's
8390 [[DiscoverFromExternalSource]](origin, options,
8391 sameOriginWithAncestors) method (2)
8392 * 9. WebAuthn Extensions (2) (3) (4)
8393 * 9.2. Defining extensions
8394
8395 #client-extension-outputReferenced in:
8396 * 5.1. PublicKeyCredential Interface
8397 * 5.1.3. Create a new credential - PublicKeyCredential's
8398 [[Create]](origin, options, sameOriginWithAncestors) method (2)
8399 * 5.1.4.1. PublicKeyCredential's
8400 [[DiscoverFromExternalSource]](origin, options,
8401 sameOriginWithAncestors) method (2)
8402 * 5.8. Authentication Extensions Client Outputs (typedef
8403 AuthenticationExtensionsClientOutputs)
8404 * 7.1. Registering a new credential
8405 * 7.2. Verifying an authentication assertion
8406 * 9. WebAuthn Extensions (2) (3) (4)
8407 * 9.2. Defining extensions (2) (3)
8408 * 9.4. Client extension processing (2) (3)

8409
8410 #authenticator-extension-processingReferenced in:
8411 * 6.2.2. The authenticatorMakeCredential operation
8412 * 6.2.3. The authenticatorGetAssertion operation
8413 * 9. WebAuthn Extensions
8414 * 9.2. Defining extensions
8415 * 9.5. Authenticator extension processing

```

```

7383
7384 #authenticator-extension-outputReferenced in:
7385 * 6.1. Authenticator data
7386 * 9. WebAuthn Extensions (2) (3)
7387
7388 * 9.2. Defining extensions (2) (3)
7389 * 9.4. Client extension processing
7390 * 9.5. Authenticator extension processing
7391 * 9.6. Example Extension
7392 * 10.5. Supported Extensions Extension (exts)
7393 * 10.6. User Verification Index Extension (uvi)
7394 * 10.7. Location Extension (loc)
7395 * 10.8. User Verification Method Extension (uvm)
7396
7397 #typedefdef-authenticatorselectionlistReferenced in:
7398 * 10.4. Authenticator Selection Extension (authnSel)
7399
7400 #typedefdef-aaguidReferenced in:
7401 * 10.4. Authenticator Selection Extension (authnSel)

```

```

8414
8415 #authenticator-extension-outputReferenced in:
8416 * 6.1. Authenticator data
8417 * 7.1. Registering a new credential
8418 * 7.2. Verifying an authentication assertion
8419 * 9. WebAuthn Extensions (2) (3) (4)
8420 * 9.2. Defining extensions (2) (3)
8421 * 9.4. Client extension processing
8422 * 9.5. Authenticator extension processing
8423
8424 * 10.5. Supported Extensions Extension (exts)
8425 * 10.6. User Verification Index Extension (uvi)
8426
8427 * 10.8. User Verification Method Extension (uvm)
8428
8429 #dictdef-txauthgenericargReferenced in:
8430 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
8431
8432 #typedefdef-authenticatorselectionlistReferenced in:
8433 * 10.4. Authenticator Selection Extension (authnSel) (2)
8434
8435 #typedefdef-aaguidReferenced in:
8436 * 10.4. Authenticator Selection Extension (authnSel)
8437
8438 #typedefdef-authenticationextensionssupportedReferenced in:
8439 * 10.5. Supported Extensions Extension (exts)
8440
8441 #typedefdef-uvmentryReferenced in:
8442 * 10.8. User Verification Method Extension (uvm)
8443
8444 #typedefdef-uvmentriesReferenced in:
8445 * 10.8. User Verification Method Extension (uvm)
8446
8447 #anonymization-caReferenced in:
8448 * 5.1.3. Create a new credential - PublicKeyCredential's
8449 [[Create]](origin, options, sameOriginWithAncestors) method
8450 * 5.4.6. Attestation Conveyance Preference enumeration (enum
8451 AttestationConveyancePreference)
8452 * 14.1. Attestation Privacy (2) (3)

```