

0001 THE URL:file://localhost/Users/jehodges/documents/work/standards/W3C/WebAuthn/index-master-
0002 tr-598ac41-WD-06.html
0003 THE TITLE:Web Authentication: An API for accessing Public Key Credentials Level 1
0004 ^I Jump to Table of Contents-> Pop Out Sidebar
0005
0006 W3C
0007 Web Authentication: An API for accessing Public Key Credentials Level 1
0008
0009 W3C Working Draft, 11 August 2017
0010
0011 This version:
0012 <https://www.w3.org/TR/2017/WD-webauthn-20170811/>
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
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 Bhavadwaj (Microsoft)
0034 Hubert Le Van Gong (PayPal)
0035 Dirk Balfanz (Google)
0036 Alexei Czeskis (Google)
0037 Arnar Birgisson (Google)
0038 Jeff Hodges (PayPal)
0039 Michael B. Jones (Microsoft)
0040 Rolf Lindemann (Nok Nok Labs)
0041 J.C. Jones (Mozilla)
0042
0043 Tests:
0044 web-platform-tests webauthn/ (ongoing work)
0045
0046 Copyright 2017 W3C^ (MIT, ERCIM, Keio, Beihang). W3C liability,
0047 trademark and document use rules apply.
0048
0049
0050 Abstract
0051 This specification defines an API enabling the creation and use of
0052 strong, attested, scoped, public key-based credentials by web
0053 applications, for the purpose of strongly authenticating users.
0054 Conceptually, one or more public key credentials, each scoped to a
0055 given Relying Party, are created and stored on an authenticator by the
0056 user agent in conjunction with the web application. The user agent
0057 mediates access to public key credentials in order to preserve user
0058 privacy. Authenticators are responsible for ensuring that no operation
0059 is performed without user consent. Authenticators provide cryptographic
0060 proof of their properties to relying parties via attestation. This
0061 specification also describes the functional model for WebAuthn
0062 conformant authenticators, including their signature and attestation
0063 functionality.
0064
0065 Status of this document
0066
0067 This section describes the status of this document at the time of its
0068 publication. Other documents may supersede this document. A list of

0001 THE URL:file://localhost/Users/jehodges/documents/work/standards/W3C/webauthn/index-master-
0002 tr-5e63e57-WD-07.html
0003 THE TITLE:Web Authentication: An API for accessing Public Key Credentials - Level 1
0004 ^I Jump to Table of Contents-> Pop Out Sidebar
0005
0006 W3C
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 Bhavadwaj (Microsoft)
0034 Hubert Le Van Gong (PayPal)
0035 Dirk Balfanz (Google)
0036 Alexei Czeskis (Google)
0037 Arnar Birgisson (Google)
0038 Jeff Hodges (PayPal)
0039 Michael B. Jones (Microsoft)
0040 Rolf Lindemann (Nok Nok Labs)
0041 J.C. Jones (Mozilla)
0042
0043 Tests:
0044 web-platform-tests webauthn/ (ongoing work)
0045
0046 Copyright 2017 W3C^ (MIT, ERCIM, Keio, Beihang). W3C liability,
0047 trademark and document use rules apply.
0048
0049
0050 Abstract
0051 This specification defines an API enabling the creation and use of
0052 strong, attested, scoped, public key-based credentials by web
0053 applications, for the purpose of strongly authenticating users.
0054 Conceptually, one or more public key credentials, each scoped to a
0055 given Relying Party, are created and stored on an authenticator by the
0056 user agent in conjunction with the web application. The user agent
0057 mediates access to public key credentials in order to preserve user
0058 privacy. Authenticators are responsible for ensuring that no operation
0059 is performed without user consent. Authenticators provide cryptographic
0060 proof of their properties to relying parties via attestation. This
0061 specification also describes the functional model for WebAuthn
0062 conformant authenticators, including their signature and attestation
0063 functionality.
0064
0065 Status of this document
0066
0067 This section describes the status of this document at the time of its
0068 publication. Other documents may supersede this document. A list of

current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at <https://www.w3.org/TR/>.

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

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

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

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

Table of Contents

- 1. 1 Introduction
 - 1. 1.1 Use Cases
 - 1. 1.1.1 Registration
 - 2. 1.1.2 Authentication
 - 3. 1.1.3 Other use cases and configurations
- 2. 2 Conformance
 - 1. 2.1 Dependencies
- 3. 3 Terminology
- 4. 4 Web Authentication API
 - 1. 4.1 PublicKeyCredential Interface
 - 1. 4.1.1 CredentialCreationOptions Extension
 - 2. 4.1.2 CredentialRequestOptions Extension
 - 3. 4.1.3 Create a new credential - PublicKeyCredential's `[[Create]](options)` method
 - 4. 4.1.4 Use an existing credential to make an assertion - PublicKeyCredential's `[[DiscoverFromExternalSource]](options)` method
 - 5. 4.1.5 Platform Authenticator Availability - PublicKeyCredential's `isPlatformAuthenticatorAvailable()` method
- 2. 4.2 Authenticator Responses (interface `AuthenticatorResponse`)
 - 1. 4.2.1 Information about Public Key Credential (interface

- AuthenticatorAttestationResponse)
- 2. 4.2.2 Web Authentication Assertion (interface `AuthenticatorAssertionResponse`)
- 3. 4.3 Parameters for Credential Generation (dictionary `PublicKeyCredentialParameters`)
- 4. 4.4 Options for Credential Creation (dictionary `MakePublicKeyCredentialOptions`)
 - 1. 4.4.1 Public Key Entity Description (dictionary `PublicKeyCredentialEntity`)
 - 2. 4.4.2 [User Account](#) Parameters for Credential Generation

current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at <https://www.w3.org/TR/>.

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

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

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

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

Table of Contents

- 1. 1 Introduction
 - 1. 1.1 Use Cases
 - 1. 1.1.1 Registration
 - 2. 1.1.2 Authentication
 - 3. 1.1.3 Other use cases and configurations
- 2. 2 Conformance
 - 1. 2.1 User Agents
- 2. 2.2 Authenticators
- 3. 2.3 Relying Parties
- 3. 3 Dependencies
- 4. 4 Terminology
- 5. 5 Web Authentication API
 - 1. 5.1 PublicKeyCredential Interface
 - 1. 5.1.1 CredentialCreationOptions Extension
 - 2. 5.1.2 CredentialRequestOptions Extension
 - 3. 5.1.3 Create a new credential - PublicKeyCredential's `[[Create]](origin, options, sameOriginWithAncestors)`

- method
- 4. 5.1.4 Use an existing credential to make an assertion - PublicKeyCredential's `[[Get]](options)` method
 - 1. 5.1.4.1 PublicKeyCredential's `[[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors)` method
- 5. 5.1.5 Store an existing credential - PublicKeyCredential's `[[Store]](credential, sameOriginWithAncestors)` method
- 6. 5.1.6 Availability of User-Verifying Platform Authenticator - PublicKeyCredential's `isUserVerifyingPlatformAuthenticatorAvailable()` method
- 2. 5.2 Authenticator Responses (interface `AuthenticatorResponse`)
 - 1. 5.2.1 Information about Public Key Credential (interface `AuthenticatorAttestationResponse`)
 - 2. 5.2.2 Web Authentication Assertion (interface `AuthenticatorAssertionResponse`)
- 3. 5.3 Parameters for Credential Generation (dictionary `PublicKeyCredentialParameters`)
- 4. 5.4 Options for Credential Creation (dictionary `MakePublicKeyCredentialOptions`)
 - 1. 5.4.1 Public Key Entity Description (dictionary `PublicKeyCredentialEntity`)
 - 2. 5.4.2 [RP](#) Parameters for Credential Generation (dictionary

0128 (dictionary PublicKeyCredentialUserEntity)
0129 3. 4.4.3 Authenticator Selection Criteria (dictionary
0130 AuthenticatorSelectionCriteria)
0131 4. 4.4.4 Authenticator Attachment enumeration (enum
0132 AuthenticatorAttachment)
0133 5. 4.5 Options for Assertion Generation (dictionary)

0134 PublicKeyCredentialRequestOptions)
0135 6. 4.6 Authentication Extensions (typedef

0136 AuthenticationExtensions)
0137 7. 4.7 Supporting Data Structures
0138 1. 4.7.1 Client data used in WebAuthn signatures (dictionary
0139 CollectedClientData)
0140 2. 4.7.2 Credential Type enumeration (enum
0141 PublicKeyCredentialType)
0142 3. 4.7.3 Credential Descriptor (dictionary
0143 PublicKeyCredentialDescriptor)
0144 4. 4.7.4 Authenticator Transport enumeration (enum
0145 AuthenticatorTransport)
0146 5. 4.7.5 Cryptographic Algorithm Identifier (typedef
0147 COSEAlgorithmIdentifier)
0148 5. 5 WebAuthn Authenticator model
0149 1. 5.1 Authenticator data
0150 2. 5.2 Authenticator operations
0151 1. 5.2.1 The authenticatorMakeCredential operation
0152 2. 5.2.2 The authenticatorGetAssertion operation
0153 3. 5.2.3 The authenticatorCancel operation
0154 3. 5.3 Attestation
0155 1. 5.3.1 Attestation data
0156 2. 5.3.2 Attestation Statement Formats
0157 3. 5.3.3 Attestation Types
0158 4. 5.3.4 Generating an Attestation Object
0159 5. 5.3.5 Security Considerations
0160 1. 5.3.5.1 Privacy
0161 2. 5.3.5.2 Attestation Certificate and Attestation

0162 Certificate CA Compromise
0163 3. 5.3.5.3 Attestation Certificate Hierarchy
0164 6. 6 Relying Party Operations
0165 1. 6.1 Registering a new credential
0166 2. 6.2 Verifying an authentication assertion
0167 7. 7 Defined Attestation Statement Formats
0168 1. 7.1 Attestation Statement Format Identifiers
0169 2. 7.2 Packed Attestation Statement Format
0170 1. 7.2.1 Packed attestation statement certificate
0171 requirements
0172 3. 7.3 TPM Attestation Statement Format
0173 1. 7.3.1 TPM attestation statement certificate requirements
0174 4. 7.4 Android Key Attestation Statement Format
0175 5. 7.5 Android SafetyNet Attestation Statement Format
0176 6. 7.6 FIDO U2F Attestation Statement Format
0177 8. 8 WebAuthn Extensions
0178 1. 8.1 Extension Identifiers
0179 2. 8.2 Defining extensions
0180 3. 8.3 Extending request parameters
0181 4. 8.4 Client extension processing
0182 5. 8.5 Authenticator extension processing
0183 6. 8.6 Example Extension
0184 9. 9 Defined Extensions
0185 1. 9.1 FIDO Appid Extension (appid)
0186 2. 9.2 Simple Transaction Authorization Extension (txAuthSimple)
0187 3. 9.3 Generic Transaction Authorization Extension
0188 (txAuthGeneric)
0189 4. 9.4 Authenticator Selection Extension (authnSel)

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
0180 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
0189 requirements
0190 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)

0190	5. 9.5 Supported Extensions Extension (exts)
0191	6. 9.6 User Verification Index Extension (uvi)
0192	7. 9.7 Location Extension (loc)
0193	8. 9.8 User Verification Method Extension (uvm)
0194	10. 10 IANA Considerations
0195	1. 10.1 WebAuthn Attestation Statement Format Identifier Registrations
0196	2. 10.2 WebAuthn Extension Identifier Registrations
0197	3. 10.3 COSE Algorithm Registrations
0198	11. 11 Sample scenarios
0199	1. 11.1 Registration
0200	2. 11.2 Registration Specifically with Platform Authenticator
0201	3. 11.3 Authentication
0202	4. 11.4 Decommissioning
0203	12. 12 Acknowledgements
0204	13. Index
0205	
0206	1. Terms defined by this specification
0207	2. Terms defined by reference
0208	14. References
0209	1. Normative References
0210	2. Informative References
0211	15. IDL Index
0212	
0213	1. Introduction
0214	
0215	This section is not normative.
0216	
0217	This specification defines an API enabling the creation and use of 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.
0218	
0219	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.
0220	
0221	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).
0222	
0223	
0224	
0225	
0226	
0227	
0228	
0229	
0230	
0231	
0232	
0233	
0234	
0235	
0236	
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	

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 Registrations
0214	2. 11.2 WebAuthn Extension Identifier Registrations
0215	3. 11.3 COSE Algorithm Registrations
0216	12. 12 Sample scenarios
0217	1. 12.1 Registration
0218	2. 12.2 Registration Specifically with User Verifying Platform Authenticator
0219	3. 12.3 Authentication
0220	4. 12.4 Aborting Authentication Operations
0221	5. 12.5 Decommissioning
0222	13. 13 Security Considerations
0223	1. 13.1 Cryptographic Challenges
0224	14. 14 Acknowledgements
0225	15. Index
0226	1. Terms defined by this specification
0227	2. Terms defined by reference
0228	16. References
0229	1. Normative References
0230	2. Informative References
0231	17. IDL Index
0232	18. Issues Index
0233	
0234	1. Introduction
0235	
0236	This section is not normative.
0237	
0238	
0239	
0240	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.
0241	
0242	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.
0243	
0244	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).
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	

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

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 11 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, "Please complete this action on your phone."
- * Next, on their phone:
 - + User sees a discrete prompt or notification, "Sign in to example.com."
 - + User selects this prompt / notification.
 - + User is shown a list of their example.com identities, e.g., "Sign in as Alice / Sign in as Bob."
 - + User picks an identity, is prompted for an authorization gesture (PIN, biometric, etc.) and provides this.
- * Now, back on the laptop:
 - + Web page shows that the selected user is signed-in, and navigates to the signed-in page.

1.1.3. Other use cases and configurations

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

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

2. Conformance

This specification defines criteria for a Conforming User Agent: A User Agent MUST behave as described in this specification in order to be considered conformant. Conforming User Agents MAY implement algorithms given in this specification in any way desired, so long as the end result is indistinguishable from the result that would be obtained by the specification's algorithms. A conforming User Agent MUST also be a conforming implementation of the IDL fragments of this specification, as described in the "Web IDL" specification. [WebIDL-1]

This specification also defines a model of a conformant authenticator (see 5 WebAuthn Authenticator model). This is a set of functional and security requirements for an authenticator to be usable by a Conforming

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

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, "Please complete this action on your phone."
- * Next, on their phone:
 - + User sees a discrete prompt or notification, "Sign in to example.com."
 - + User selects this prompt / notification.
 - + User is shown a list of their example.com identities, e.g., "Sign in as Alice / Sign in as Bob."
 - + User picks an identity, is prompted for an authorization gesture (PIN, biometric, etc.) and provides this.
- * Now, back on the laptop:
 - + Web page shows that the selected user is signed-in, and navigates to the signed-in page.

1.1.3. Other use cases and configurations

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

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

2. Conformance

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

2.1. User Agents

0325 User Agent. As described in 1.1 Use Cases, an authenticator may be
0326 implemented in the operating system underlying the User Agent, or in
0327 external hardware, or a combination of both.

0328 2.1. Dependencies

0330 This specification relies on several other underlying specifications,
0331 listed below and in Terms defined by reference.
0332
0333 Base64url encoding
0334 The term Base64url Encoding refers to the base64 encoding using
0335 the URL- and filename-safe character set defined in Section 5 of
0336 [RFC4648], with all trailing '=' characters omitted (as
0337 permitted by Section 3.2) and without the inclusion of any line
0338 breaks, whitespace, or other additional characters.
0339
0340 CBOR
0341 A number of structures in this specification, including
0342 attestation statements and extensions, are encoded using the
0343 Compact Binary Object Representation (CBOR) [RFC7049].
0344
0345 CDDL
0346 This specification describes the syntax of all CBOR-encoded data
0347 using the CBOR Data Definition Language (CDDL) [CDDL].
0348
0349 COSE
0350 CBOR Object Signing and Encryption (COSE) [RFC8152]. The IANA
0351 COSE Algorithms registry established by this specification is
0352 also used.
0353
0354 Credential Management
0355 The API described in this document is an extension of the
0356 Credential concept defined in [CREDENTIAL-MANAGEMENT-1].
0357
0358 DOM
0359 DOMException and the DOMException values used in this
0360 specification are defined in [DOM4].
0361
0362 ECMAScript
0363 %ArrayBuffer% is defined in [ECMAScript].
0364
0365 HTML
0366 The concepts of relevant settings object, origin, opaque origin,
0367 and is a registrable domain suffix of or is equal to are defined

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 2.2. Authenticators

0352
0353 An authenticator MUST provide the operations defined by 6 WebAuthn
0354 Authenticator model, and those operations MUST behave as described
0355 there. This is a set of functional and security requirements for an
0356 authenticator to be usable by a Conforming User Agent.

0357
0358 As described in 1.1 Use Cases, an authenticator may be implemented in
0359 the operating system underlying the User Agent, or in external
0360 hardware, or a combination of both.

0361 2.3. Relying Parties

0362
0363 A Relying Party MUST behave as described in 7 Relying Party Operations
0364 to get the security benefits offered by this specification.

0365 3. Dependencies

0366 This specification relies on several other underlying specifications,
0367 listed below and in Terms defined by reference.

0368 Base64url encoding
0369 The term Base64url Encoding refers to the base64 encoding using
0370 the URL- and filename-safe character set defined in Section 5 of
0371 [RFC4648], with all trailing '=' characters omitted (as
0372 permitted by Section 3.2) and without the inclusion of any line
0373 breaks, whitespace, or other additional characters.

0374
0375 CBOR
0376 A number of structures in this specification, including
0377 attestation statements and extensions, are encoded using the
0378 Compact Binary Object Representation (CBOR) [RFC7049].

0379
0380 CDDL
0381 This specification describes the syntax of all CBOR-encoded data
0382 using the CBOR Data Definition Language (CDDL) [CDDL].

0383
0384 COSE
0385 CBOR Object Signing and Encryption (COSE) [RFC8152]. The IANA
0386 COSE Algorithms registry established by this specification is
0387 also used.

0388
0389 Credential Management
0390 The API described in this document is an extension of the
0391 Credential concept defined in [CREDENTIAL-MANAGEMENT-1].

0392
0393 DOM
0394 DOMException and the DOMException values used in this
0395 specification are defined in [DOM4].

0396
0397 ECMAScript
0398 %ArrayBuffer% is defined in [ECMAScript].

0399
0400 HTML
0401 The concepts of relevant settings object, origin, opaque origin,
0402 and is a registrable domain suffix of or is equal to are defined

036f in [HTML52].
037f
0371 Web IDL
0372 Many of the interface definitions and all of the IDL in this
0373 specification depend on [WebIDL-1]. This updated version of the
0374 Web IDL standard adds support for Promises, which are now the
0375 preferred mechanism for asynchronous interaction in all new web
0376 APIs.
0377
0378 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
0379 "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
0380 document are to be interpreted as described in [RFC2119].
0381
0382 3. Terminology
0383
0384 Assertion
0385 See Authentication Assertion.
0386
0387 Attestation
0388 Generally, attestation is a statement serving to bear witness,
0389 confirm, or authenticate. In the WebAuthn context, attestation
0390 is employed to attest to the provenance of an authenticator and
0391 the data it emits; including, for example: credential IDs,
0392 credential key pairs, signature counters, etc. An attestation
0393 statement is conveyed in an attestation object during
0394 registration. See also 5.3 Attestation and Figure 3.

0395
0396 Attestation Certificate
0397 A X.509 Certificate for the attestation key pair used by an
0398 authenticator to attest to its manufacture and capabilities. At
0399 registration time, the authenticator uses the attestation
0400 private key to sign the Relying Party-specific credential public
0401 key (and additional data) that it generates and returns via the
0402 authenticatorMakeCredential operation. Relying Parties use the
0403 attestation public key conveyed in the attestation certificate
0404 to verify the attestation signature. Note that in the case of
0405 self attestation, the authenticator has no distinct attestation
0406 key pair nor attestation certificate, see self attestation for
0407 details.
0408
0409 Authentication
0410 The ceremony where a user, and the user's computing device(s)
0411 (containing at least one authenticator) work in concert to
0412 cryptographically prove to an Relying Party that the user
0413 controls the credential private key associated with a
0414 previously-registered public key credential (see Registration).
0415 Note that this typically includes employing a test of user
0416 presence or user verification.
0417
0418 Authentication Assertion
0419 The cryptographically signed AuthenticatorAssertionResponse
0420 object returned by an authenticator as the result of a
0421 authenticatorGetAssertion operation.
0422

0423 Authenticator
0424 A cryptographic device used by a WebAuthn Client to (i) generate
0425 a public key credential and register it with a Relying Party,
0426 and (ii) subsequently used to cryptographically sign and return,
0427 in the form of an Authentication Assertion, a challenge and
0428 other data presented by a Relying Party (in concert with the
0429 WebAuthn Client) in order to effect authentication.
0430
0431 Authorization Gesture
0432 An authorization gesture is a physical interaction performed by

0408 in [HTML52].
0409
0410 Web IDL
0411 Many of the interface definitions and all of the IDL in this
0412 specification depend on [WebIDL-1]. This updated version of the
0413 Web IDL standard adds support for Promises, which are now the
0414 preferred mechanism for asynchronous interaction in all new web
0415 APIs.
0416
0417 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
0418 "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
0419 document are to be interpreted as described in [RFC2119].
0420
0421 4. Terminology
0422
0423 Assertion
0424 See Authentication Assertion.
0425
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

0433 a user with an authenticator as part of a ceremony, such as
0434 registration or authentication. By making such an authorization
0435 gesture, a user provides consent for (i.e., authorizes) a
0436 ceremony to proceed. This may involve user verification if the
0437 employed authenticator is capable, or it may involve a simple
0438 test of user presence.
0439
0440 **Biometric Recognition**
0441 The automated recognition of individuals based on their
0442 biological and behavioral characteristics
0443 [ISOBiometricVocabulary].
0444
0445 **Ceremony**
0446 The concept of a ceremony [Ceremony] is an extension of the
0447 concept of a network protocol, with human nodes alongside
0448 computer nodes and with communication links that include user
0449 interface(s), human-to-human communication, and transfers of
0450 physical objects that carry data. What is out-of-band to a
0451 protocol is in-band to a ceremony. In this specification,
0452 Registration and Authentication are ceremonies, and an
0453 authorization gesture is often a component of those ceremonies.
0454
0455 **Client**
0456 See Conforming User Agent.
0457
0458 **Client-Side**
0459 This refers in general to the combination of the user's platform
0460 device, user agent, authenticators, and everything gluing it all
0461 together.
0462
0463 **Client-side-resident Credential Private Key**
0464 A Client-side-resident Credential Private Key is stored either
0465 on the client platform, or in some cases on the authenticator
0466 itself, e.g., in the case of a discrete first-factor roaming
0467 authenticator. Such client-side credential private key storage
0468 has the property that the authenticator is able to select the
0469 credential private key given only an RP ID, possibly with user
0470 assistance (e.g., by providing the user a pick list of
0471 credentials associated with the RP ID). By definition, the
0472 private key is always exclusively controlled by the
0473 Authenticator. In the case of a Client-side-resident Credential
0474 Private Key, the Authenticator might offload storage of wrapped
0475 key material to the client platform, but the client platform is
0476 not expected to offload the key storage to remote entities (e.g.
0477 RP Server).
0478
0479 **Conforming User Agent**
0480 A user agent implementing, in conjunction with the underlying
0481 platform, the Web Authentication API and algorithms given in
0482 this specification, and handling communication between
0483 authenticators and Relying Parties.
0484

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
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
0503 **Client-Side**
0504 This refers in general to the combination of the user's platform
0505 device, user agent, authenticators, and everything gluing it all
0506 together.
0507
0508 **Client-side-resident Credential Private Key**
0509 A Client-side-resident Credential Private Key is stored either
0510 on the client platform, or in some cases on the authenticator
0511 itself, e.g., in the case of a discrete first-factor roaming
0512 authenticator. Such client-side credential private key storage
0513 has the property that the authenticator is able to select the
0514 credential private key given only an RP ID, possibly with user
0515 assistance (e.g., by providing the user a pick list of
0516 credentials associated with the RP ID). By definition, the
0517 private key is always exclusively controlled by the
0518 Authenticator. In the case of a Client-side-resident Credential
0519 Private Key, the Authenticator might offload storage of wrapped
0520 key material to the client platform, but the client platform is
0521 not expected to offload the key storage to remote entities (e.g.
0522 RP Server).
0523
0524 **Conforming User Agent**
0525 A user agent implementing, in conjunction with the underlying
0526 platform, the Web Authentication API and algorithms given in
0527 this specification, and handling communication between
0528 authenticators and Relying Parties.
0529
0530 **Credential ID**
0531 A probabilistically-unique byte sequence identifying a public
0532 key credential source and its authentication assertions.
0533
0534 **Credential IDs are generated by authenticators in two forms:**
0535
0536 **1. At least 16 bytes that include at least 100 bits of entropy,**
0537 **or**
0538 **2. The public key credential source, without its Credential ID,**
0539 **encrypted so only its managing authenticator can decrypt it.**
0540 **This form allows the authenticator to be nearly stateless, by**
0541 **having the Relying Party store any necessary state.**
0542 **Note: [FIDO-UAF-AUTHNR-CMDS] includes guidance on encryption**
0543 **techniques under "Security Guidelines".**
0544
0545 **Relying Parties do not need to distinguish these two Credential**
0546 **ID forms.**
0547

0486 Credential Public Key
0486 The public key portion of an Relying Party-specific credential
0487 key pair, generated by an authenticator and returned to an
0488 Relying Party at registration time (see also public key
0488 credential). The private key portion of the credential key pair
0490 is known as the credential private key. Note that in the case of
0491 self attestation, the credential key pair is also used as the
0492 attestation key pair, see self attestation for details.
0493

0548 Credential Public Key
0548 The public key portion of an Relying Party-specific credential
0550 key pair, generated by an authenticator and returned to an
0551 Relying Party at registration time (see also public key
0552 credential). The private key portion of the credential key pair
0553 is known as the credential private key. Note that in the case of
0554 self attestation, the credential key pair is also used as the
0555 attestation key pair, see self attestation for details.
0556
0557 Public Key Credential Source
0558 A credential source ([CREDENTIAL-MANAGEMENT-1]) used by an
0559 authenticator to generate authentication assertions. A public
0560 key credential source has:
0561
0562 + A Credential ID.
0563 + A credential private key.
0564 + The Relying Party Identifier for the Relying Party that
0565 created this credential source.
0566 + An optional user handle for the person who created this
0567 credential source.
0568 + Optional other information used by the authenticator to inform
0569 its UI. For example, this might include the user's
0570 displayName.
0571
0572 The authenticatorMakeCredential operation creates a public key
0573 credential source bound to a managing authenticator and returns
0574 the credential public key associated with its credential private
0575 key. The Relying Party can use this credential public key to
0576 verify the authentication assertions created by this public key
0577 credential source.
0578
0579 Public Key Credential
0580 Generically, a credential is data one entity presents to another
0581 in order to authenticate the former to the latter [RFC4949]. The
0582 term public key credential refers to one of: a public key
0583 credential source, the possibly-attested credential public key
0584 corresponding to a public key credential source, or an
0585 authentication assertion. Which one is generally determined by
0586 context.
0587
0588 Note: This is a willful violation of [RFC4949]. In English, a
0589 "credential" is both a) the thing presented to prove a statement
0590 and b) intended to be used multiple times. It's impossible to
0591 achieve both criteria securely with a single piece of data in a
0592 public key system. [RFC4949] chooses to define a credential as
0593 the thing that can be used multiple times (the public key),
0594 while this specification gives "credential" the English term's
0595 flexibility. This specification uses more specific terms to
0596 identify the data related to an [RFC4949] credential:
0597
0598 "Authentication information" (possibly including a private key)
0599 Public key credential source
0600
0601 "Signed value"
0602 Authentication assertion
0603
0604 [RFC4949] "credential"
0605 Credential public key or attestation object
0606
0607 At registration time, the authenticator creates an asymmetric
0608 key pair, and stores its private key portion and information
0609 from the Relying Party into a public key credential source. The
0610 public key portion is returned to the Relying Party, who then
0611 stores it in conjunction with the present user's account.
0612 Subsequently, only that Relying Party, as identified by its RP
0613 ID, is able to employ the public key credential in
0614 authentication ceremonies, via the get() method. The Relying
0615 Party uses its stored copy of the credential public key to
0616 verify the resultant authentication assertion.
0617

0494 Rate Limiting
0495 The process (also known as throttling) by which an authenticator
0496 implements controls against brute force attacks by limiting the
0497 number of consecutive failed authentication attempts within a
0498 given period of time. If the limit is reached, the authenticator
0499 should impose a delay that increases exponentially with each
0500 successive attempt, or disable the current authentication
0501 modality and offer a different authentication factor if
0502 available. Rate limiting is often implemented as an aspect of
0503 user verification.
0504
0505 Registration
0506 The ceremony where a user, a Relying Party, and the user's
0507 computing device(s) (containing at least one authenticator) work
0508 in concert to create a public key credential and associate it
0509 with the user's Relying Party account. Note that this **typically**
0510 **includes** employing a test of user presence or user verification.
0511
0512 Relying Party
0513 The entity whose web application utilizes the Web Authentication
0514 API to register and authenticate users. See Registration and
0515 Authentication, respectively.
0516
0517 Note: While the term Relying Party is used in other contexts
0518 (e.g., X.509 and OAuth), an entity acting as a Relying Party in
0519 one context is not necessarily a Relying Party in other
0520 contexts.
0521
0522 Relying Party Identifier
0523 RP ID
0524 A valid domain string that identifies the Relying Party on whose
0525 behalf a given registration or authentication ceremony is being
0526 performed. A public key credential can only be used for
0527 authentication with the same entity (as identified by RP ID) it
0528 was registered with. By default, the RP ID for a WebAuthn
0529 operation is set to the caller's origin's effective domain. This
0530 default MAY be overridden by the caller, as long as the
0531 caller-specified RP ID value is a registrable domain suffix of
0532 or is equal to the caller's origin's effective domain. See also
0533 4.1.3 Create a new credential - PublicKeyCredential's
0534 [[Create]](options) method and 4.1.4 Use an existing credential
0535 to make an assertion - PublicKeyCredential's
0536 [[DiscoverFromExternalSource]](options) method.
0537
0538 Note: A Public key credential's scope is for a Relying Party's
0539 origin, with the following restrictions and relaxations:
0540
0541 + The scheme is always https (i.e., a restriction), and,
0542 + the host may be equal to the Relying Party's origin's
0543 effective domain, or it may be equal to a registrable domain
0544 suffix of the Relying Party's origin's effective domain (i.e.,
0545 an available relaxation), and,
0546 + all (TCP) ports on that host (i.e., a relaxation).
0547
0548 This is done in order to match the behavior of pervasively
0549 deployed ambient credentials (e.g., cookies, [RFC6265]). Please
0550 note that this is a greater relaxation of "same-origin"
0551 restrictions than what document.domain's setter provides.
0552
0553 Public Key Credential
0554 Generically, a credential is data one entity presents to another
0555 in order to authenticate the former to the latter [RFC4949]. A
0556 WebAuthn public key credential is a { identifier, type } pair
0557 identifying authentication information established by the
0558 authenticator and the Relying Party, together, at registration
0559 time. The authentication information consists of an asymmetric
0560 key pair, where the public key portion is returned to the
0561 Relying Party, who then stores it in conjunction with the
0562 present user's account. The authenticator maps the private key
0563 portion to the Relying Party's RP ID and stores it.

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

0564 Subsequently, only that Relying Party, as identified by its RP
0565 ID, is able to employ the public key credential in
0566 authentication ceremonies, via the get() method. The Relying
0567 Party uses its stored copy of the credential public key to
0568 verify the resultant authentication assertion.
0569
0570
0571 Test of User Presence
0572 A test of user presence is a simple form of authorization
0573 gesture and technical process where a user interacts with an
0574 authenticator by (typically) simply touching it (other
0575 modalities may also exist), yielding a boolean result. Note that
0576 this does not constitute user verification because a user
0577 presence test, by definition, is not capable of biometric
0578 recognition, nor does it involve the presentation of a shared
0579 secret such as a password or PIN.
0580
0581 User Consent
0582 User consent means the user agrees with what they are being
0583 asked, i.e., it encompasses reading and understanding prompts.
0584 An authorization gesture is a ceremony component often employed
0585 to indicate user consent.

0586
0587
0588 User Verification
0589 The technical process by which an authenticator locally
0590 authorizes the invocation of the authenticatorMakeCredential and
0591 authenticatorGetAssertion operations. User verification may be
0592 instigated through various authorization gesture modalities; for
0593 example, through a touch plus pin code, password entry, or
0594 biometric recognition (e.g., presenting a fingerprint)
0595 [ISOBiometricVocabulary]. The intent is to be able to
0596 distinguish individual users. Note that invocation of the
0597 authenticatorMakeCredential and authenticatorGetAssertion
0598 operations implies use of key material managed by the
0599 authenticator. Note that for security, user verification and use
0600 of credential private keys must occur within a single logical
0601 security boundary defining the authenticator.

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

0605
0606 User Verified
0607 UV
0608 Upon successful completion of a user verification process, the
0609 user is said to be "verified".

0610
0611 WebAuthn Client
0612 Also referred to herein as simply a client. See also Conforming
0613 User Agent.

0614
0615 4. Web Authentication API
0616

0617 This section normatively specifies the API for creating and using
0618 public key credentials. The basic idea is that the credentials belong
0619 to the user and are managed by an authenticator, with which the Relying
0620 Party interacts through the client (consisting of the browser and
0621 underlying OS platform). Scripts can (with the user's consent) request
0622 the browser to create a new credential for future use by the Relying

0677
0678 Test of User Presence
0679 A test of user presence is a simple form of authorization
0680 gesture and technical process where a user interacts with an
0681 authenticator by (typically) simply touching it (other
0682 modalities may also exist), yielding a boolean result. Note that
0683 this does not constitute user verification because a user
0684 presence test, by definition, is not capable of biometric
0685 recognition, nor does it involve the presentation of a shared
0686 secret such as a password or PIN.
0687
0688 User Consent
0689 User consent means the user agrees with what they are being
0690 asked, i.e., it encompasses reading and understanding prompts.
0691 An authorization gesture is a ceremony component often employed
0692 to indicate user consent.

0693
0694 User Handle
0695 The user handle is specified by a Relying Party and is a unique
0696 identifier for a user account with that Relying Party. A user
0697 handle is an opaque byte sequence with a maximum size of 64
0698 bytes.

0699 The user handle is not meant to be displayed to the user, but is
0700 used by the Relying Party to control the number of credentials -
0701 an authenticator will never contain more than one credential for
0702 a given Relying Party under the same user handle.
0703

0704
0705 User Verification
0706 The technical process by which an authenticator locally
0707 authorizes the invocation of the authenticatorMakeCredential and
0708 authenticatorGetAssertion operations. User verification may be
0709 instigated through various authorization gesture modalities; for
0710 example, through a touch plus pin code, password entry, or
0711 biometric recognition (e.g., presenting a fingerprint)
0712 [ISOBiometricVocabulary]. The intent is to be able to
0713 distinguish individual users. Note that invocation of the
0714 authenticatorMakeCredential and authenticatorGetAssertion
0715 operations implies use of key material managed by the
0716 authenticator. Note that for security, user verification and use
0717 of credential private keys must occur within a single logical
0718 security boundary defining the authenticator.

0719
0720 User Present
0721 UP
0722 Upon successful completion of a user presence test, the user is
0723 said to be "present".

0724
0725 User Verified
0726 UV
0727 Upon successful completion of a user verification process, the
0728 user is said to be "verified".

0729
0730 WebAuthn Client
0731 Also referred to herein as simply a client. See also Conforming
0732 User Agent.

0733
0734 5. Web Authentication API
0735

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

0623 Party. Scripts can also request the user's permission to perform
0624 authentication operations with an existing credential. All such
0625 operations are performed in the authenticator and are mediated by the
0626 browser and/or platform on the user's behalf. At no point does the
0627 script get access to the credentials themselves; it only gets
0628 information about the credentials in the form of objects.
0629
0630 In addition to the above script interface, the authenticator may
0631 implement (or come with client software that implements) a user
0632 interface for management. Such an interface may be used, for example,
0633 to reset the authenticator to a clean state or to inspect the current
0634 state of the authenticator. In other words, such an interface is
0635 similar to the user interfaces provided by browsers for managing user
0636 state such as history, saved passwords and cookies. Authenticator
0637 management actions such as credential deletion are considered to be the
0638 responsibility of such a user interface and are deliberately omitted
0639 from the API exposed to scripts.
0640
0641 The security properties of this API are provided by the client and the
0642 authenticator working together. The authenticator, which holds and
0643 manages credentials, ensures that all operations are scoped to a
0644 particular origin, and cannot be replayed against a different origin,
0645 by incorporating the origin in its responses. Specifically, as defined
0646 in 5.2 Authenticator operations, the full origin of the requester is
0647 included, and signed over, in the attestation object produced when a
0648 new credential is created as well as in all assertions produced by
0649 WebAuthn credentials.
0650
0651 Additionally, to maintain user privacy and prevent malicious Relying
0652 Parties from probing for the presence of public key credentials
0653 belonging to other Relying Parties, each credential is also associated
0654 with a Relying Party Identifier, or RP ID. This RP ID is provided by
0655 the client to the authenticator for all operations, and the
0656 authenticator ensures that credentials created by a Relying Party can
0657 only be used in operations requested by the same RP ID. Separating the
0658 origin from the RP ID in this way allows the API to be used in cases
0659 where a single Relying Party maintains multiple origins.
0660
0661 The client facilitates these security measures by providing the Relying
0662 Party's origin and RP ID to the authenticator for each operation. Since
0663 this is an integral part of the WebAuthn security model, user agents
0664 only expose this API to callers in secure contexts.
0665
0666 The Web Authentication API is defined by the union of the Web IDL
0667 fragments presented in the following sections. A combined IDL listing
0668 is given in the IDL Index.
0669
0670 **4.1. PublicKeyCredential Interface**
0671
0672 The PublicKeyCredential interface inherits from Credential
0673 [CREDENTIAL-MANAGEMENT-1], and contains the attributes that are
0674 returned to the caller when a new credential is created, or a new
0675 assertion is requested.
0676 [SecureContext]
0677 interface PublicKeyCredential : Credential {
0678 [SameObject] readonly attribute ArrayBuffer rawId;
0679 [SameObject] readonly attribute AuthenticatorResponse response;
0680 [SameObject] readonly attribute AuthenticationExtensions clientExtensionResu
0681 its;
0682 };
0683
0684 id
0685 This attribute is inherited from Credential, though
0686 PublicKeyCredential overrides Credential's getter, instead
0687 returning the base64url encoding of the data contained in the
0688 object's [[identifier]] internal slot.
0689
0690 rawId
0691 This attribute returns the ArrayBuffer contained in the
0692 [[identifier]] internal slot.

0741 Party. Scripts can also request the user's permission to perform
0742 authentication operations with an existing credential. All such
0743 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.

0693
0694 response, of type AuthenticatorResponse, readonly
0695 This attribute contains the authenticator's response to the
0696 client's request to either create a public key credential, or
0697 generate an authentication assertion. If the PublicKeyCredential
0698 is created in response to create(), this attribute's value will
0699 be an AuthenticatorAttestationResponse, otherwise, the
0700 PublicKeyCredential was created in response to get(), and this
0701 attribute's value will be an AuthenticatorAssertionResponse.
0702
0703 clientExtensionResults, of type AuthenticationExtensions, readonly
0704 This attribute contains a map containing extension identifier ->
0705 client extension output entries produced by the extension's
0706 client extension processing.

0707
0708 [[type]]
0709 The PublicKeyCredential interface object's [[type]] internal
0710 slot's value is the string "public-key".

0711
0712 Note: This is reflected via the type attribute getter inherited
0713 from Credential.

0714
0715 [[discovery]]
0716 The PublicKeyCredential interface object's [[discovery]]
0717 internal slot's value is "remote".
0718
0719 [[identifier]]
0720 This internal slot contains an identifier for the credential,
0721 chosen by the platform with help from the authenticator. This
0722 identifier is used to look up credentials for use, and is
0723 therefore expected to be globally unique with high probability
0724 across all credentials of the same type, across all
0725 authenticators. This API does not constrain the format or length
0726 of this identifier, except that it must be sufficient for the
0727 platform to uniquely select a key. For example, an authenticator
0728 without on-board storage may create identifiers containing a
0729 credential private key wrapped with a symmetric key that is
0730 burned into the authenticator.
0731

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

0736
0737 4.1.1. CredentialCreationOptions Extension
0738
0739 To support registration via navigator.credentials.create(), this
0740 document extends the CredentialCreationOptions dictionary as follows:
0741 partial dictionary CredentialCreationOptions {
0742 MakePublicKeyCredentialOptions publicKey;
0743 };
0744
0745 4.1.2. CredentialRequestOptions Extension
0746
0747 To support obtaining assertions via navigator.credentials.get(), this
0748 document extends the CredentialRequestOptions dictionary as follows:
0749 partial dictionary CredentialRequestOptions {
0750 PublicKeyCredentialRequestOptions publicKey;
0751 };
0752

0810 response, of type AuthenticatorResponse, readonly
0811 This attribute contains the authenticator's response to the
0812 client's request to either create a public key credential, or
0813 generate an authentication assertion. If the PublicKeyCredential
0814 is created in response to create(), this attribute's value will
0815 be an AuthenticatorAttestationResponse, otherwise, the
0816 PublicKeyCredential was created in response to get(), and this
0817 attribute's value will be an AuthenticatorAssertionResponse.
0818
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
0857 PublicKeyCredential's interface object inherits Credential's
0858 implementation of [[CollectFromCredentialStore]](origin, options,
0859 sameOriginWithAncestors), and defines its own implementation of
0860 [[Create]](origin, options, sameOriginWithAncestors),
0861 [[DiscoverFromExternalSource]](origin, options,
0862 sameOriginWithAncestors), and [[Store]](credential,
0863 sameOriginWithAncestors).
0864
0865 5.1.1. CredentialCreationOptions Extension
0866
0867 To support registration via navigator.credentials.create(), this
0868 document extends the CredentialCreationOptions dictionary as follows:
0869 partial dictionary CredentialCreationOptions {
0870 MakePublicKeyCredentialOptions publicKey;
0871 };
0872
0873 5.1.2. CredentialRequestOptions Extension
0874
0875 To support obtaining assertions via navigator.credentials.get(), this
0876 document extends the CredentialRequestOptions dictionary as follows:
0877 partial dictionary CredentialRequestOptions {
0878 PublicKeyCredentialRequestOptions publicKey;
0879 };
0880

0753 4.1.3. Create a new credential - PublicKeyCredential's `[[Create]](options)`
0754 method
0755
0756 PublicKeyCredential's interface object's implementation of the
0757 `[[Create]](options)` method allows scripts to call
0758 `navigator.credentials.create()` to request the creation of a new
0759 credential key pair and PublicKeyCredential, managed by an
0760 authenticator. The user agent will prompt the user for consent. On
0761 success, the returned promise will be resolved with a
0762 PublicKeyCredential containing an AuthenticatorAttestationResponse
0763 object.
0764
0765 Note: This algorithm is synchronous; the Promise resolution/rejection
0766 is handled by `navigator.credentials.create()`.

0767
0768 This method accepts a single argument:

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

When this method is invoked, the user agent MUST execute the following
algorithm:
1. Assert: options.publicKey is present.
2. Let options be the value of options.publicKey.
3. If any of the name member of options.rp, the name member of
options.user, the displayName member of options.user, or the id
member of options.user are not present, return a TypeError simple
exception.

4. If the timeout member of options is present, check if its value
lies within a reasonable range as defined by the platform and if
not, correct it to the closest value lying within that range. Set
adjustedTimeout to this adjusted value. If the timeout member of
options is not present, then set adjustedTimeout to a
platform-specific default.
5. Let global be the PublicKeyCredential's interface object's
environment settings object's global object.
6. Let callerOrigin be the origin specified by this
PublicKeyCredential interface object's relevant settings object. If
callerOrigin is an opaque origin, return a DOMException whose name
is "NotAllowedError", and terminate this algorithm.

0880 5.1.3. Create a new credential - PublicKeyCredential's `[[Create]](origin,`
0881 `options, sameOriginWithAncestors)` method
0882
0883 PublicKeyCredential's interface object's implementation of the

0884
0885 `[[Create]](origin, options, sameOriginWithAncestors)` internal method
0886 [CREDENTIAL-MANAGEMENT-1] allows Relying Party scripts to call
0887 `navigator.credentials.create()` to request the creation of a new public
0888 key credential source, bound to an authenticator. This
0889 `navigator.credentials.create()` operation can be aborted by leveraging
0890 the AbortController; see DOM 3.3 Using AbortController and AbortSignal
0891 objects in APIs for detailed instructions.
0892
0893 This internal method accepts three arguments:
0894
origin
This argument is the relevant settings object's origin, as
determined by the calling create() implementation.

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

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

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

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

0796 7. Let effectiveDomain be the callerOrigin's effective domain. If
0797 effective domain is not a valid domain, then return a DOMException
0798 whose name is "SecurityError" and terminate this algorithm.
0799 Note: An effective domain may resolve to a host, which can be
0800 represented in various manners, such as domain, ipv4 address, ipv6
0801 address, opaque host, or empty host. Only the domain format of host
0802 is allowed here.
0803 8. Let rpId be effectiveDomain.
0804 9. If options.rp.id is present:
0805 1. If options.rp.id is not a registrable domain suffix of and is
0806 not equal to effectiveDomain, return a DOMException whose name
0807 is "SecurityError", and terminate this algorithm.
0808 2. Set rpId to options.rp.id.
0809 Note: rpId represents the caller's RP ID. The RP ID defaults
0810 to being the caller's origin's effective domain unless the
0811 caller has explicitly set options.rp.id when calling create().
0812 10. Let credTypesAndPubKeyAlgs be a new list whose items are pairs of
0813 PublicKeyCredentialType and a COSEAlgorithmIdentifier.
0814 11. For each current of options.pubKeyCredParams:
0815 1. If current.type does not contain a PublicKeyCredentialType
0816 supported by this implementation, then continue.
0817 2. Let alg be current.alg.
0818 3. Append the pair of current.type and alg to
0819 credTypesAndPubKeyAlgs.
0820 12. If credTypesAndPubKeyAlgs is empty and options.pubKeyCredParams is
0821 not empty, cancel the timer started in step 2, return a
0822 DOMException whose name is "NotSupportedError", and terminate this
0823 algorithm.
0824 13. Let clientExtensions be a new map and let authenticatorExtensions
0825 be a new map.
0826 14. If the extensions member of options is present, then for each
0827 extensionId -> clientExtensionInput of options.extensions:
0828 1. If extensionId is not supported by this client platform or is
0829 not a registration extension, then continue.
0830 2. Set clientExtensions[extensionId] to clientExtensionInput.
0831 3. If extensionId is not an authenticator extension, then
0832 continue.
0833 4. Let authenticatorExtensionInput be the (CBOR) result of
0834 running extensionId's client extension processing algorithm on
0835 clientExtensionInput. If the algorithm returned an error,
0836 continue.
0837 5. Set authenticatorExtensions[extensionId] to the base64url
0838 encoding of authenticatorExtensionInput.
0839 15. Let collectedClientData be a new CollectedClientData instance whose
0840 fields are:
0841
0842 challenge
0843 The base64url encoding of options.challenge.
0844
0845 origin
0846 The serialization of callerOrigin.
0847
0848 hashAlgorithm
0849 The recognized algorithm name of the hash algorithm
0850 selected by the client for generating the hash of the
0851 serialized client data.
0852
0853 tokenBindingId
0854 The Token Binding ID associated with callerOrigin, if one
0855 is available.
0856
0857 clientExtensions

0941 effective domain is not a valid domain, then return a DOMException
0942 whose name is "SecurityError" and terminate this algorithm.
0943 Note: An effective domain may resolve to a host, which can be
0944 represented in various manners, such as domain, ipv4 address, ipv6
0945 address, opaque host, or empty host. Only the domain format of host
0946 is allowed here.
0947 7. If options.rp.id
0948
0949 Is present
0950 If options.rp.id is not a registrable domain suffix of and
0951 is not equal to effectiveDomain, return a DOMException
0952 whose name is "SecurityError", and terminate this
0953 algorithm.
0954
0955 Is not present
0956 Set options.rp.id to effectiveDomain.
0957
0958 Note: options.rp.id represents the caller's RP ID. The RP ID
0959 defaults to being the caller's origin's effective domain unless the
0960 caller has explicitly set options.rp.id when calling create().
0961 8. Let credTypesAndPubKeyAlgs be a new list whose items are pairs of
0962 PublicKeyCredentialType and a COSEAlgorithmIdentifier.
0963 9. For each current of options.pubKeyCredParams:
0964 1. If current.type does not contain a PublicKeyCredentialType
0965 supported by this implementation, then continue.
0966 2. Let alg be current.alg.
0967 3. Append the pair of current.type and alg to
0968 credTypesAndPubKeyAlgs.
0969 10. If credTypesAndPubKeyAlgs is empty and options.pubKeyCredParams is
0970 not empty, return a DOMException whose name is "NotSupportedError",
0971 and terminate this algorithm.
0972 11. Let clientExtensions be a new map and let authenticatorExtensions
0973 be a new map.
0974 12. If the extensions member of options is present, then for each
0975 extensionId -> clientExtensionInput of options.extensions:
0976 1. If extensionId is not supported by this client platform or is
0977 not a registration extension, then continue.
0978 2. Set clientExtensions[extensionId] to clientExtensionInput.
0979 3. If extensionId is not an authenticator extension, then
0980 continue.
0981 4. Let authenticatorExtensionInput be the (CBOR) result of
0982 running extensionId's client extension processing algorithm on
0983 clientExtensionInput. If the algorithm returned an error,
0984 continue.
0985 5. Set authenticatorExtensions[extensionId] to the base64url
0986 encoding of authenticatorExtensionInput.
0987 13. Let collectedClientData be a new CollectedClientData instance whose
0988 fields are:
0989
0990 type
0991 The string "webauthn.create".
0992
0993 challenge
0994 The base64url encoding of options.challenge.
0995
0996 origin
0997 The serialization of callerOrigin.
0998
0999 hashAlgorithm
1000 The recognized algorithm name of the hash algorithm
1001 selected by the client for generating the hash of the
1002 serialized client data.
1003
1004 tokenBindingId
1005 The Token Binding ID associated with callerOrigin, if one
1006 is available.
1007
1008 clientExtensions

```
085f clientExtensions
085f
086f authenticatorExtensions
086f authenticatorExtensions
086f
086f 16. Let clientDataJSON be the JSON-serialized client data constructed
086f from collectedClientData.
086f 17. Let clientDataHash be the hash of the serialized client data
086f represented by clientDataJSON.
086f 18. Let currentlyAvailableAuthenticators be a new ordered set
086f consisting of all authenticators currently available on this
086f platform.
087f 19. Let selectedAuthenticators be a new ordered set.
087f 20. If currentlyAvailableAuthenticators is empty, return a DOMException
087f whose name is "NotFoundError", and terminate this algorithm.
087f 21. If options.authenticatorSelection is present, iterate through
087f currentlyAvailableAuthenticators and do the following for each
087f authenticator:
087f 1. If aa is present and its value is not equal to authenticator's
087f
0877 attachment modality, continue.
087f 2. If rk is set to true and the authenticator is not capable of
087f
0879 storing a Client-Side-Resident Credential Private Key,
0880 continue.
088f 3. If uv is set to true and the authenticator is not capable of
088f
0882 performing user verification, continue.
088f 4. Append authenticator to selectedAuthenticators.
088f 22. If selectedAuthenticators is empty, return a DOMException whose
088f name is "ConstraintError", and terminate this algorithm.
088f 23. Let issuedRequests be a new ordered set.
088f 24. For each authenticator in currentlyAvailableAuthenticators:
088f 1. Let excludeCredentialDescriptorList be a new list.
088f 2. For each credential descriptor C in
088f
089f options.excludeCredentials:
089f 1. If C.transports is not empty, and authenticator is
089f connected over a transport not mentioned in C.transports,
089f the client MAY continue.
089f 2. Otherwise, Append C to excludeCredentialDescriptorList.
089f 3. In parallel, invoke the authenticatorMakeCredential operation
089f on authenticator with rpId, clientDataHash, options.rp,
089f options.user, options.authenticatorSelection.rk,
089f credTypesAndPubKeyAlgs, excludeCredentialDescriptorList, and
089f authenticatorExtensions as parameters.
090f 4. Append authenticator to issuedRequests.
090f 25. Start a timer for adjustedTimeout milliseconds. Then execute the
090f following steps in parallel. The task source for these tasks is the
090f dom manipulation task source.
090f 26. While issuedRequests is not empty, perform the following actions
```

```
100f clientExtensions
101f
101f authenticatorExtensions
101f authenticatorExtensions
101f
101f 14. Let clientDataJSON be the JSON-serialized client data constructed
101f from collectedClientData.
101f 15. Let clientDataHash be the hash of the serialized client data
101f represented by clientDataJSON.
101f 16. If the options.signal is present and its aborted flag is set to
101f true, return a DOMException whose name is "AbortError" and
101f terminate this algorithm.
101f 17. Start lifetimeTimer.
101f 18. Let issuedRequests be a new ordered set.
101f 19. For each authenticator that becomes available on this platform
101f during the lifetime of lifetimeTimer, do the following:
101f The definitions of "lifetime of" and "becomes available" are
101f intended to represent how devices are hotplugged into (USB) or
101f discovered by (NFC) browsers, and are under-specified. Resolving
101f this with good definitions or some other means will be addressed by
101f resolving Issue #613.
101f 1. If options.authenticatorSelection is present:
101f 1. If options.authenticatorSelection.authenticatorAttachment
101f is present and its value is not equal to authenticator's
101f attachment modality, continue.
101f 2. If options.authenticatorSelection.requireResidentKey is
101f set to true and the authenticator is not capable of
101f storing a Client-Side-Resident Credential Private Key,
101f continue.
101f 3. If options.authenticatorSelection.userVerification is set
101f to required and the authenticator is not capable of
101f performing user verification, continue.
101f 2. Let userVerification be the effective user verification
101f requirement for credential creation, a Boolean value, as
101f follows. If options.authenticatorSelection.userVerification
101f
101f is set to required
101f Let userVerification be true.
101f
101f is set to preferred
101f If the authenticator
101f
101f is capable of user verification
101f Let userVerification be true.
101f
101f is not capable of user verification
101f Let userVerification be false.
101f
101f is set to discouraged
101f Let userVerification be false.
101f
101f 3. Let userPresence be a Boolean value set to the inverse of
101f userVerification.
101f 4. Let excludeCredentialDescriptorList be a new list.
101f 5. For each credential descriptor C in
101f options.excludeCredentials:
101f 1. If C.transports is not empty, and authenticator is
101f connected over a transport not mentioned in C.transports,
101f the client MAY continue.
101f 2. Otherwise, Append C to excludeCredentialDescriptorList.
101f 6. Invoke the authenticatorMakeCredential operation on
101f authenticator with clientDataHash, options.rp, options.user,
101f options.authenticatorSelection.requireResidentKey,
101f userPresence, userVerification, credTypesAndPubKeyAlgs,
101f excludeCredentialDescriptorList, and authenticatorExtensions
101f as parameters.
101f 7. Append authenticator to issuedRequests.
101f 20. While issuedRequests is not empty, perform the following actions
101f depending upon lifetimeTimer and responses from the authenticators:
```

0905 depending upon the adjustedTimeout timer and responses from the
0906 authenticators;
0907
0908 If the adjustedTimeout timer expires,
0909 For each authenticator in issuedRequests invoke the
0910 authenticatorCancel operation on authenticator and remove
0911 authenticator from issuedRequests.
0912

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

1078
1079 If lifetimeTimer expires,
1080 For each authenticator in issuedRequests invoke the
1081 authenticatorCancel operation on authenticator and remove
1082 authenticator from issuedRequests.
1083
1084 If the options.signal is present and its aborted flag is set to
1085 true,
1086 For each authenticator in issuedRequests invoke the
1087 authenticatorCancel operation on authenticator and remove
1088 authenticator from issuedRequests. Then return a
1089 DOMException whose name is "AbortError" and terminate this
1090 algorithm.
1091
1092 If any authenticator returns a status indicating that the user
1093 cancelled the operation,
1094
1095 1. Remove authenticator from issuedRequests.
1096 2. For each remaining authenticator in issuedRequests invoke
1097 the authenticatorCancel operation on authenticator and
1098 remove it from issuedRequests.
1099
1100 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
1108 attestationObjectResult
1109 whose value is the bytes returned from the
1110 successful authenticatorMakeCredential
1111 operation.
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:
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
1142 "none"
1143 Replace potentially uniquely identifying
1144 information (such as AAGUID and
1145 attestation certificates) in the

0937
0938
0939
0940
0941
0942
0943

0944
0945
0946
0947
0948

0949
0950
0951
0952
0953
0954
0955
0956
0957
0958
0959
0960
0961
0962

0963
0964
0965

0966
0967
0968

[[identifier]]
id

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

clientDataJSON
A new ArrayBuffer, created using
global's %ArrayBuffer%, containing the
bytes of clientDataJSON.

attestationObject
attestationObject

clientExtensionResults
A new AuthenticationExtensions object
containing the extension identifier -> client
extension output entries created by running
each extension's client extension processing
algorithm to create the client extension
outputs, for each client extension in
clientDataJSON.clientExtensions.

5. For each remaining authenticator in issuedRequests invoke
the authenticatorCancel operation on authenticator and
remove it from issuedRequests.
6. Return value and terminate this algorithm.

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

1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203

1204
1205
1206
1207
1208
1209
1210
1211
1212
1213

attested credential data and attestation
statement, respectively, with blinded
versions of the same data.

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

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

"direct"
Convey the authenticator's AAGUID and
attestation statement, unaltered, to the
RP.

@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".

3. Let id be
attestationObject.authData.attestedCredentialData.c
redentialId.
4. Let pubKeyCred be a new PublicKeyCredential object
associated with global whose fields are:

[[identifier]]
id

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

clientDataJSON
A new ArrayBuffer, created using
global's %ArrayBuffer%, containing
the bytes of
credentialCreationData.clientDataJ
SONResult.

attestationObject
attestationObject

[[clientExtensionResults]]
A new ArrayBuffer, created using
global's %ArrayBuffer%, containing the
bytes of
credentialCreationData.clientExtensionRe
sults.

5. Return pubKeyCred.
4. For each remaining authenticator in issuedRequests invoke
the authenticatorCancel operation on authenticator and
remove it from issuedRequests.
5. Return constructCredentialAlg and terminate this
algorithm.

21. Return a DOMException whose name is "NotAllowedError".

0965	During the above process, the user agent SHOULD show some UI to the
0970	user to guide them in the process of selecting and authorizing an
0971	authenticator.
0972	
0973	4.1.4. Use an existing credential to make an assertion -
0974	PublicKeyCredential's <code>[[DiscoverFromExternalSource]](options)</code> method
0975	
0976	The <code>[[DiscoverFromExternalSource]](options)</code> method is used to discover
0977	and use an existing public key credential, with the user's consent. The
0978	script optionally specifies some criteria to indicate what credentials
0979	are acceptable to it. The user agent and/or platform locates
0980	credentials matching the specified criteria, and guides the user to
0981	pick one that the script will be allowed to use. The user may choose
0982	not to provide a credential even if one is present, for example to
0983	maintain privacy.
0984	
0985	Note: This algorithm is synchronous; the Promise resolution/rejection
0986	is handled by <code>navigator.credentials.get()</code> .
0987	
0988	This method accepts a single argument:
0989	
0990	
0991	
0992	options
0993	This argument is a <code>CredentialRequestOptions</code> object whose
0994	<code>options.publicKey</code> member contains a challenge and additional
0995	options as described in 4.5 Options for Assertion Generation
0996	(dictionary <code>PublicKeyCredentialRequestOptions</code>). The selected
0997	authenticator signs the challenge along with other collected
0998	data in order to produce an assertion. See 5.2.2 The
0999	<code>authenticatorGetAssertion</code> operation.
1000	
1001	When this method is invoked, the user agent MUST execute the following
1002	algorithm:
1003	1. Assert: <code>options.publicKey</code> is present.
	2. Let <code>options</code> be the value of <code>options.publicKey</code> .
	3. If the <code>timeout</code> member of <code>options</code> is present, check if its value

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 <code>[[Get]](options)</code> method
1220	
1221	Relying Parties call <code>navigator.credentials.get({publicKey:..., ...})</code> 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 <code>authenticatorGetAssertion</code> 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 <code>get()</code> implementation [CREDENTIAL-MANAGEMENT-1] calls
1235	<code>PublicKeyCredential.[[CollectFromCredentialStore]]()</code> 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	<code>PublicKeyCredential.[[DiscoverFromExternalSource]]()</code> 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	<code>PublicKeyCredential.[[CollectFromCredentialStore]](origin, options,</code>
1245	<code>sameOriginWithAncestors)</code> internal method inherits the default behavior
1246	of <code>Credential.[[CollectFromCredentialStore]]()</code> , of returning an empty
1247	set.
1248	
1249	5.1.4.1. <code>PublicKeyCredential's [[DiscoverFromExternalSource]](origin,</code>
1250	<code>options, sameOriginWithAncestors)</code> 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 <code>get()</code> implementation, i.e.,
1257	<code>CredentialsContainer's Request a Credential</code> abstract operation.
1258	
1259	options
1260	This argument is a <code>CredentialRequestOptions</code> object whose
1261	<code>options.publicKey</code> member contains a
1262	<code>PublicKeyCredentialRequestOptions</code> 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 <code>navigator.credentials.get()</code> .
1272	
1273	When this method is invoked, the user agent MUST execute the following
1274	algorithm:
1275	1. Assert: <code>options.publicKey</code> is present.
1276	2. If <code>sameOriginWithAncestors</code> is false, return a "NotAllowedError"
1277	<code>DOMException</code> .
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

1004 lies within a reasonable range as defined by the platform and if
1005 not, correct it to the closest value lying within that range. Set
1006 **adjustedTimeout** to this adjusted value. If the timeout member of
1007 options is not present, then set **adjustedTimeout** to a
1008 platform-specific default.
1009 4. Let **global** be the **PublicKeyCredential**'s interface object's
1010 environment settings object's global object.
1011 5. Let **callerOrigin** be the origin specified by this
1012 **PublicKeyCredential** interface object's relevant settings object. If
1013 **callerOrigin** is an opaque origin, return a **DOMException** whose name
1014 is "NotAllowedError", and terminate this algorithm.
1015 6. Let **effectiveDomain** be the **callerOrigin**'s effective domain. If
1016 effective domain is not a valid domain, then return a **DOMException**
1017 whose name is "SecurityError" and terminate this algorithm.
1018 Note: An effective domain may resolve to a host, which can be
1019 represented in various manners, such as domain, ipv4 address, ipv6
1020 address, opaque host, or empty host. Only the domain format of host
1021 is allowed here.
1022 7. If options.**rpId** is not present, then set **rpId** to **effectiveDomain**.
1023 Otherwise:
1024 1. If options.**rpId** is not a registrable domain suffix of and is
1025 not equal to **effectiveDomain**, return a **DOMException** whose name
1026 is "SecurityError", and terminate this algorithm.
1027 2. Set **rpId** to options.**rpId**.
1028 Note: **rpId** represents the caller's RP ID. The RP ID defaults
1029 to being the caller's origin's effective domain unless the
1030 caller has explicitly set options.**rpId** when calling **get()**.
1031 8. Let **clientExtensions** be a new map and let **authenticatorExtensions**
1032 be a new map.
1033 9. If the extensions member of options is present, then for each
1034 extensionId -> **clientExtensionInput** of options.extensions:
1035 1. If extensionId is not supported by this client platform or is
1036 not an authentication extension, then continue.
1037 2. Set **clientExtensions**[extensionId] to **clientExtensionInput**.
1038 3. If extensionId is not an authenticator extension, then
1039 continue.
1040 4. Let **authenticatorExtensionInput** be the (CBOR) result of
1041 running extensionId's client extension processing algorithm on
1042 **clientExtensionInput**. If the algorithm returned an error,
1043 continue.
1044 5. Set **authenticatorExtensions**[extensionId] to the base64url
1045 encoding of **authenticatorExtensionInput**.
1046 10. Let **collectedClientData** be a new **CollectedClientData** instance whose
1047 fields are:
1048
1049 challenge
1050 The base64url encoding of options.challenge
1051
1052 origin
1053 The serialization of **callerOrigin**.
1054
1055 hashAlgorithm
1056 The recognized algorithm name of the hash algorithm
1057 selected by the client for generating the hash of the
1058 serialized client data
1059
1060 tokenBindingId
1061 The Token Binding ID associated with **callerOrigin**, if one
1062 is available.

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
1301 6. Let **effectiveDomain** be the **callerOrigin**'s effective domain. If
1302 effective domain is not a valid domain, then return a **DOMException**
1303 whose name is "SecurityError" and terminate this algorithm.
1304 Note: An effective domain may resolve to a host, which can be
1305 represented in various manners, such as domain, ipv4 address, ipv6
1306 address, opaque host, or empty host. Only the domain format of host
1307 is allowed here.
1308 7. If options.**rpId** is not present, then set **rpId** to **effectiveDomain**.
1309 Otherwise:
1310 1. If options.**rpId** is not a registrable domain suffix of and is
1311 not equal to **effectiveDomain**, return a **DOMException** whose name
1312 is "SecurityError", and terminate this algorithm.
1313 2. Set **rpId** to options.**rpId**.
1314 Note: **rpId** represents the caller's RP ID. The RP ID defaults
1315 to being the caller's origin's effective domain unless the
1316 caller has explicitly set options.**rpId** when calling **get()**.
1317 8. Let **clientExtensions** be a new map and let **authenticatorExtensions**
1318 be a new map.
1319 9. If the extensions member of options is present, then for each
1320 extensionId -> **clientExtensionInput** of options.extensions:
1321 1. If extensionId is not supported by this client platform or is
1322 not an authentication extension, then continue.
1323 2. Set **clientExtensions**[extensionId] to **clientExtensionInput**.
1324 3. If extensionId is not an authenticator extension, then
1325 continue.
1326 4. Let **authenticatorExtensionInput** be the (CBOR) result of
1327 running extensionId's client extension processing algorithm on
1328 **clientExtensionInput**. If the algorithm returned an error,
1329 continue.
1330 5. Set **authenticatorExtensions**[extensionId] to the base64url
1331 encoding of **authenticatorExtensionInput**.
1332 10. Let **collectedClientData** be a new **CollectedClientData** instance whose
1333 fields are:
1334
1335 type
1336 The string "webauthn.get".
1337
1338 challenge
1339 The base64url encoding of options.challenge
1340
1341 origin
1342 The serialization of **callerOrigin**.
1343
1344 hashAlgorithm
1345 The recognized algorithm name of the hash algorithm
1346 selected by the client for generating the hash of the
1347 serialized client data
1348
1349 tokenBindingId
1350 The Token Binding ID associated with **callerOrigin**, if one
1351 is available.


```

1351 clientExtensions
1352 clientExtensions
1353
1354 authenticatorExtensions
1355 authenticatorExtensions
1356
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
1365 terminate this algorithm.
1366 14. Let issuedRequests be a new ordered set.
1367 15. Let authenticator be a platform-specific handle whose value
1368 identifies an authenticator.
1369 16. Start lifetimeTimer.
1370 17. For each authenticator that becomes available on this platform
1371 during the lifetime of lifetimeTimer, perform the following steps:
1372 The definitions of "lifetime of" and "becomes available" are
1373 intended to represent how devices are hotplugged into (USB) or
1374 discovered by (NFC) browsers, and are under-specified. Resolving
1375 this with good definitions or some other means will be addressed by
1376 resolving Issue #613.
1377 1. If options.userVerification is set to required and the
1378 authenticator is not capable of performing user verification,
1379 continue.
1380 2. Let userVerification be the effective user verification
1381 requirement for assertion, a Boolean value, as follows. If
1382 options.userVerification
1383 is set to required
1384 Let userVerification be true.
1385
1386 is set to preferred
1387 If the authenticator
1388 is capable of user verification
1389 Let userVerification be true.
1390
1391 is not capable of user verification
1392 Let userVerification be false.
1393
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 is not empty
1410
1411 1. Let distinctTransports be a new ordered set.
1412 2. If allowCredentialDescriptorList has exactly one
1413 value, let savedCredentialId be a new
1414 PublicKeyCredentialDescriptor.id and set its value
1415 to allowCredentialDescriptorList[0].id's value (see
1416 here in 6.2.2 The authenticatorGetAssertion
1417 operation for more information).
1418
1419

```

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.

3. 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 in parallel the authenticatorGetAssertion operation on authenticator, with rpId, clientDataHash, allowCredentialDescriptorList, and authenticatorExtensions as parameters.

is empty

Using local configuration knowledge of the appropriate transport to use with authenticator, invoke in parallel the authenticatorGetAssertion operation on authenticator with rpId, clientDataHash, allowCredentialDescriptorList, and clientExtensions as parameters.

is empty

Using local configuration knowledge of the appropriate transport to use with authenticator, invoke in parallel the authenticatorGetAssertion operation on authenticator with rpId, clientDataHash, and clientExtensions as parameters.

Note: In this case, the Relying Party did not supply a list of acceptable credential descriptors. Thus the authenticator is being asked to exercise any credential it may possess that is bound to the Relying Party, as identified by rpId.

4. Append authenticator to issuedRequests.

17. Start a timer for adjustedTimeout milliseconds. Then execute the following steps in parallel. The task source for these tasks is the dom manipulation task source.

18. While issuedRequests is not empty, perform the following actions depending upon the adjustedTimeout timer and responses from the authenticators:

If the adjustedTimeout timer expires,

For each authenticator in issuedRequests invoke the authenticatorCancel operation on authenticator and remove authenticator from issuedRequests.

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.

is empty

Using local configuration knowledge of the appropriate transport to use with authenticator, invoke the authenticatorGetAssertion operation on authenticator with rpId, clientDataHash, allowCredentialDescriptorList, userPresence, userVerification, and clientExtensions as parameters.

is empty

Using local configuration knowledge of the appropriate transport to use with authenticator, invoke the authenticatorGetAssertion operation on authenticator with rpId, clientDataHash, userPresence, userVerification and clientExtensions as parameters.

Note: In this case, the Relying Party did not supply a list of acceptable credential descriptors. Thus the authenticator is being asked to exercise any credential it may possess that is bound to the Relying Party, as identified by rpId.

7. Append authenticator to issuedRequests.

18. While issuedRequests is not empty, perform the following actions depending upon lifetimeTimer and responses from the authenticators:

If lifetimeTimer expires,

For each authenticator in issuedRequests invoke the authenticatorCancel operation on authenticator and remove authenticator from issuedRequests.

If the signal member is present and the aborted flag is set to

1154 If any authenticator returns a status indicating that the user
1155 cancelled the operation,
1156
1157 1. Remove authenticator from issuedRequests.
1158 2. For each remaining authenticator in issuedRequests invoke
1159 the authenticatorCancel operation on authenticator and
1160 remove it from issuedRequests.
1161
1162 If any authenticator returns an error status,
1163 Remove authenticator from issuedRequests.
1164
1165 If any authenticator indicates success,
1166
1167 1. Remove authenticator from issuedRequests.
1168 2. Let **value** be a new **PublicKeyCredential** associated with
1169 global whose fields are:
1170
1171 **[[identifier]]**
1172 A new **ArrayBuffer**, created using global's
1173 **%ArrayBuffer%**, containing the bytes of the

1174 credential ID returned from the successful
1175 authenticatorGetAssertion operation, as
1176 defined in 5.2.2 The
1177 authenticatorGetAssertion operation.
1178
1179 **response**
1180 A new **AuthenticatorAssertionResponse** object

1181 associated with global whose fields are:
1182
1183 **clientDataJSON**
1184 A new **ArrayBuffer**, created using
1185 global's **%ArrayBuffer%**, containing the
1186 bytes of **clientDataJSON**

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 **credentialIdResult**
1509 If **savedCredentialId** exists, set the value of
1510 **credentialIdResult** to be the bytes of
1511 **savedCredentialId**. Otherwise, set the value of
1512 **credentialIdResult** to be the bytes of the
1513 credential ID returned from the successful
1514 authenticatorGetAssertion operation, as
1515 defined in 6.2.2 The
1516 authenticatorGetAssertion operation.
1517
1518 **clientDataJSONResult**
1519 whose value is the bytes of **clientDataJSON**.
1520
1521 **authenticatorDataResult**
1522 whose value is the bytes of the authenticator
1523 data returned by the authenticator.
1524
1525 **signatureResult**
1526 whose value is the bytes of the signature
1527 value returned by the authenticator.
1528
1529 **userHandleResult**
1530 whose value is the bytes of the user handle
1531 returned by the authenticator.
1532
1533 **clientExtensionResults**
1534 whose value is an **AuthenticationExtensions**
1535 object containing extension identifier ->
1536 client extension output entries. The entries
1537 are created by running each extension's client
1538 extension processing algorithm to create the
1539 client extension outputs, for each client
1540 extension in **clientDataJSON.clientExtensions**.
1541
1542 3. Let **constructAssertionAlg** be an algorithm that takes a
1543 global object **global**, and whose steps are:
1544 1. Let **pubKeyCred** be a new **PublicKeyCredential** object
1545 associated with global whose fields are:
1546
1547 **[[identifier]]**
1548 A new **ArrayBuffer**, created using
1549 global's **%ArrayBuffer%**, containing the
1550 bytes of
1551 **assertionCreationData.credentialIdResult**
1552
1553
1554

1187
1188
1189 authenticatorData
1190 A new ArrayBuffer, created using
1191 global's %ArrayBuffer%, containing the
bytes of the returned authenticatorData

1192
1193 signature
1194 A new ArrayBuffer, created using
1195 global's %ArrayBuffer%, containing the
1196 bytes of the returned signature

1197
1198 clientExtensionResults
1199 A new AuthenticationExtensions object
1200 containing the extension identifier -> client
1201 extension output entries created by running
1202 each extension's client extension processing
1203 algorithm to create the client extension
1204 outputs, for each client extension in
1205 clientDataJSON.clientExtensions.
1206
1207 3. For each remaining authenticator in issuedRequests invoke

1208
1209 the authenticatorCancel operation on authenticator and
1210 remove it from issuedRequests.
4. Return value and terminate this algorithm.

1211
1212 19. 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 with which to complete the operation.

1217
1218 4.1.5. Platform Authenticator Availability - PublicKeyCredential's
1219 isPlatformAuthenticatorAvailable() method

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
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

1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277

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

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

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

This method has no arguments and returns a boolean value.

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

[SecureContext]
partial interface PublicKeyCredential {
 [Unscopable] Promise < boolean > isPlatformAuthenticatorAvailable();
};

4.2. Authenticator Responses (interface AuthenticatorResponse)

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

[SecureContext]
interface AuthenticatorResponse {
 [SameObject] readonly attribute ArrayBuffer clientDataJSON;
};

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

4.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse)

The AuthenticatorAttestationResponse interface represents the

1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691

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

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

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

5.1.6. Availability of User-Verifying Platform Authenticator - PublicKeyCredential's isUserVerifyingPlatformAuthenticatorAvailable() method

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

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

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

This method has no arguments and returns a boolean value.

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

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

5.2. Authenticator Responses (interface AuthenticatorResponse)

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

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

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

5.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse)

The AuthenticatorAttestationResponse interface represents the

1278 authenticator's response to a client's request for the creation of a
1279 new public key credential. It contains information about the new
1280 credential that can be used to identify it for later use, and metadata
1281 that can be used by the Relying Party to assess the characteristics of
1282 the credential during registration.
1283 [SecureContext]
1284 interface AuthenticatorAttestationResponse : AuthenticatorResponse {
1285 [SameObject] readonly attribute ArrayBuffer attestationObject;
1286 };
1287
1288 clientDataJSON
1289 This attribute, inherited from AuthenticatorResponse, contains
1290 the JSON-serialized client data (see 5.3 Attestation) passed to
1291 the authenticator by the client in order to generate this
1292 credential. The exact JSON serialization must be preserved, as
1293 the hash of the serialized client data has been computed over
1294 it.
1295
1296 attestationObject, of type ArrayBuffer, readonly
1297 This attribute contains an attestation object, which is opaque
1298 to, and cryptographically protected against tampering by, the
1299 client. The attestation object contains both authenticator data
1300 and an attestation statement. The former contains the AAGUID, a
1301 unique credential ID, and the credential public key. The
1302 contents of the attestation statement are determined by the
1303 attestation statement format used by the authenticator. It also
1304 contains any additional information that the Relying Party's
1305 server requires to validate the attestation statement, as well
1306 as to decode and validate the authenticator data along with the
1307 JSON-serialized client data. For more details, see 5.3
1308 Attestation, 5.3.4 Generating an Attestation Object, and Figure
1309 3.
1310
1311 4.2.2. Web Authentication Assertion (interface
1312 AuthenticatorAssertionResponse)
1313
1314 The AuthenticatorAssertionResponse interface represents an
1315 authenticator's response to a client's request for generation of a new
1316 authentication assertion given the Relying Party's challenge and
1317 optional list of credentials it is aware of. This response contains a
1318 cryptographic signature proving possession of the credential private
1319 key, and optionally evidence of user consent to a specific transaction.
1320 [SecureContext]
1321 interface AuthenticatorAssertionResponse : AuthenticatorResponse {
1322 [SameObject] readonly attribute ArrayBuffer authenticatorData;
1323 [SameObject] readonly attribute ArrayBuffer signature;
1324 };
1325
1326 clientDataJSON
1327 This attribute, inherited from AuthenticatorResponse, contains
1328 the JSON-serialized client data (see 4.7.1 Client data used in
1329 WebAuthn signatures (dictionary CollectedClientData)) passed to
1330 the authenticator by the client in order to generate this
1331 assertion. The exact JSON serialization must be preserved, as
1332 the hash of the serialized client data has been computed over
1333 it.
1334
1335 authenticatorData, of type ArrayBuffer, readonly
1336 This attribute contains the authenticator data returned by the
1337 authenticator. See 5.1 Authenticator data.
1338
1339 signature, of type ArrayBuffer, readonly
1340 This attribute contains the raw signature returned from the
1341 authenticator. See 5.2.2 The authenticatorGetAssertion
1342 operation.
1343
1344 4.3. Parameters for Credential Generation (dictionary

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


```
1345 PublicKeyCredentialParameters)
1346
1347 dictionary PublicKeyCredentialParameters {
1348   required PublicKeyCredentialType type;
1349   required COSEAlgorithmIdentifier alg;
1350 };
1351
1352 This dictionary is used to supply additional parameters when creating a
1353 new credential.
1354
1355 The type member specifies the type of credential to be created.
1356
1357 The alg member specifies the cryptographic signature algorithm with
1358 which the newly generated credential will be used, and thus also the
1359 type of asymmetric key pair to be generated, e.g., RSA or Elliptic
1360 Curve.
1361
1362 Note: we use "alg" as the latter member name, rather than spelling-out
1363 "algorithm", because it will be serialized into a message to the
1364 authenticator, which may be sent over a low-bandwidth link.
1365
1366 4.4. Options for Credential Creation (dictionary
1367 MakePublicKeyCredentialOptions)
1368
1369 dictionary MakePublicKeyCredentialOptions {
1370   required PublicKeyCredentialEntity rp;
1371   required PublicKeyCredentialUserEntity user;
1372
1373   required BufferSource challenge;
1374   required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
1375
1376   unsigned long timeout;
1377   sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
1378   AuthenticatorSelectionCriteria authenticatorSelection;
1379
1380 AuthenticationExtensions extensions;
1381 };
1382
1383 rp, of type PublicKeyCredentialEntity
1384 This member contains data about the Relying Party responsible
1385 for the request.
1386
1387 Its value's name member is required, and contains the friendly
1388 name of the Relying Party (e.g. "Acme Corporation", "Widgets,
1389 Inc.", or "Awesome Site").
1390
1391 Its value's id member specifies the relying party identifier
1392 with which the credential should be associated. If omitted, its
1393 value will be the CredentialsContainer object's relevant
1394 settings object's origin's effective domain.
1395
1396 user, of type PublicKeyCredentialUserEntity
1397 This member contains data about the user account for which the
1398 Relying Party is requesting attestation.
1399
1400 Its value's name member is required, and contains a name for the
1401 user account (e.g., "john.p.smith@example.com" or
1402 "+14255551234").
1403
1404 Its value's displayName member is required, and contains a
1405 friendly name for the user account (e.g., "John P. Smith").
1406
1407 Its value's id member is required, and contains an identifier
1408 for the account, specified by the Relying Party. This is not
1409 meant to be displayed to the user, but is used by the Relying
1410 Party to control the number of credentials - an authenticator
1411 will never contain more than one credential for a given Relying
```

```
1762 operation.
1763
1764 5.3. Parameters for Credential Generation (dictionary
1765 PublicKeyCredentialParameters)
1766
1767 dictionary PublicKeyCredentialParameters {
1768   required PublicKeyCredentialType type;
1769   required COSEAlgorithmIdentifier alg;
1770 };
1771
1772 This dictionary is used to supply additional parameters when creating a
1773 new credential.
1774
1775 The type member specifies the type of credential to be created.
1776
1777 The alg member specifies the cryptographic signature algorithm with
1778 which the newly generated credential will be used, and thus also the
1779 type of asymmetric key pair to be generated, e.g., RSA or Elliptic
1780 Curve.
1781
1782 Note: we use "alg" as the latter member name, rather than spelling-out
1783 "algorithm", because it will be serialized into a message to the
1784 authenticator, which may be sent over a low-bandwidth link.
1785
1786 5.4. Options for Credential Creation (dictionary
1787 MakePublicKeyCredentialOptions)
1788
1789 dictionary MakePublicKeyCredentialOptions {
1790   required PublicKeyCredentialRpEntity rp;
1791   required PublicKeyCredentialUserEntity user;
1792
1793   required BufferSource challenge;
1794   required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
1795
1796   unsigned long timeout;
1797   sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
1798   AuthenticatorSelectionCriteria authenticatorSelection;
1799   AttestationConveyancePreference attestation = "none";
1800   AuthenticationExtensions extensions;
1801 };
1802
1803 rp, of type PublicKeyCredentialRpEntity
1804 This member contains data about the Relying Party responsible
1805 for the request.
1806
1807 Its value's name member contains the friendly name of the
1808 Relying Party (e.g. "Acme Corporation", "Widgets, Inc.", or
1809 "Awesome Site").
1810
1811 Its value's id member specifies the relying party identifier
1812 with which the credential should be associated. If omitted, its
1813 value will be the CredentialsContainer object's relevant
1814 settings object's origin's effective domain.
1815
1816 user, of type PublicKeyCredentialUserEntity
1817 This member contains data about the user account for which the
1818 Relying Party is requesting attestation.
1819
1820 Its value's name member contains a name for the user account
1821 (e.g., "john.p.smith@example.com" or "+14255551234").
1822
1823 Its value's displayName member contains a friendly name for the
1824 user account (e.g., "John P. Smith").
1825
1826 Its value's id member contains the user handle for the account,
1827 specified by the Relying Party.
```

1411 **Party under the same id.**
1412
1413 challenge, of type BufferSource
1414 This member contains a challenge intended to be used for
1415 generating the newly created credential's attestation object.
1416
1417 pubKeyCredParams, of type sequence<PublicKeyCredentialParameters>
1418 This member contains information about the desired properties of
1419 the credential to be created. The sequence is ordered from most
1420 preferred to least preferred. The platform makes a best-effort
1421 to create the most preferred credential that it can.
1422
1423 timeout, of type unsigned long
1424 This member specifies a time, in milliseconds, that the caller
1425 is willing to wait for the call to complete. This is treated as
1426 a hint, and may be overridden by the platform.
1427
1428 excludeCredentials, of type sequence<PublicKeyCredentialDescriptor>,
1429 defaulting to None
1430 This member is intended for use by Relying Parties that wish to
1431 limit the creation of multiple credentials for the same account
1432 on a single authenticator. The platform is requested to return
1433 an error if the new credential would be created on an
1434 authenticator that also contains one of the credentials
1435 enumerated in this parameter.
1436
1437 authenticatorSelection, of type AuthenticatorSelectionCriteria
1438 This member is intended for use by Relying Parties that wish to
1439 select the appropriate authenticators to participate in the
1440 create() or get() operation.

1441
1442 extensions, of type AuthenticationExtensions
1443 This member contains additional parameters requesting additional
1444 processing by the client and authenticator. For example, the
1445 caller may request that only authenticators with certain
1446 capabilities be used to create the credential, or that particular
1447 information be returned in the attestation object. Some
1448 extensions are defined in 8 WebAuthn Extensions; consult the
1449 IANA "WebAuthn Extension Identifier" registry established by
1450 [WebAuthn-Registries] for an up-to-date list of registered
1451 WebAuthn Extensions.
1452
1453 4.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)
1454
1455 The PublicKeyCredentialEntity dictionary describes a user account, or a
1456 Relying Party, with which a public key credential is associated.
1457 dictionary PublicKeyCredentialEntity {
1458 DOMString id;
1459 DOMString name;
1460 USVString icon;
1461 };
1462
1463 id, of type DOMString
1464 A unique identifier for the entity. For a relying party entity,
1465 sets the RP ID. For a user account entity, this will be an
1466 arbitrary string specified by the relying party.
1467
1468 name, of type DOMString
1469 A human-friendly identifier for the entity. For example, this
1470 could be a company name for a Relying Party, or a user's name.
1471 This identifier is intended for display.

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

1472 icon, of type USVString
1473 A serialized URL which resolves to an image associated with the
1474 entity. For example, this could be a user's avatar or a Relying
1475 Party's logo.
1476

1477
1478 4.4.2. User Account Parameters for Credential Generation (dictionary

1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517

PublicKeyCredentialUserEntity)

The PublicKeyCredentialUserEntity dictionary is used to supply additional user account attributes when creating a new credential.

dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
DOMString displayName;
};

displayName, of type DOMString
A friendly name for the user account (e.g., "John P. Smith").

4.4.3. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

Relying Parties may use the AuthenticatorSelectionCriteria dictionary to specify their requirements regarding authenticator attributes.

dictionary AuthenticatorSelectionCriteria {
AuthenticatorAttachment aa; // authenticatorAttachment
boolean rk = false; // requireResidentKey
boolean uv = false; // requireUserVerification
};

aa (authenticatorAttachment), of type AuthenticatorAttachment
If this member is present, eligible authenticators are filtered to only authenticators attached with the specified 4.4.4 Authenticator Attachment enumeration (enum AuthenticatorAttachment).

rk (requireResidentKey), of type boolean, defaulting to false
This member describes the Relying Parties' requirements regarding availability of the Client-side-resident Credential Private Key. If the parameter is set to true, the authenticator MUST create a Client-side-resident Credential Private Key when creating a public key credential.

uv (requireUserVerification), of type boolean, defaulting to false
This member describes the Relying Parties' requirements regarding the authenticator being capable of performing user verification. If the parameter is set to true, the authenticator

1891 icon, of type USVString
1892 A serialized URL which resolves to an image associated with the
1893 entity. For example, this could be a user's avatar or a Relying
1894 Party's logo. This URL MUST be an a priori authenticated URL.
1895 Authenticators MUST accept and store a 128 byte minimum length
1896 for a icon members's value. Authenticators MAY ignore a icon
1897 members's value if its length is greater than 128 bytes.
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960

5.4.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRpEntity)

The PublicKeyCredentialRpEntity dictionary is used to supply additional Relying Party attributes when creating a new credential.

dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
DOMString id;
};

id, of type DOMString
A unique identifier for the Relying Party entity, which sets the RP ID.

5.4.3. User Account Parameters for Credential Generation (dictionary PublicKeyCredentialUserEntity)

The PublicKeyCredentialUserEntity dictionary is used to supply additional user account attributes when creating a new credential.

dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
required BufferSource id;
required DOMString displayName;
};

id, of type BufferSource
The user handle of the user account entity.

displayName, of type DOMString
A friendly name for the user account (e.g., "John P. Smith").
Authenticators MUST accept and store a 64 byte minimum length for a displayName members's value. Authenticators MAY truncate a displayName member's value to a length equal to or greater than 64 bytes.

5.4.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

Relying Parties may use the AuthenticatorSelectionCriteria dictionary to specify their requirements regarding authenticator attributes.

dictionary AuthenticatorSelectionCriteria {
AuthenticatorAttachment authenticatorAttachment;
boolean requireResidentKey = false;
UserVerificationRequirement userVerification = "preferred";
};

authenticatorAttachment, of type AuthenticatorAttachment
If this member is present, eligible authenticators are filtered to only authenticators attached with the specified 5.4.5 Authenticator Attachment enumeration (enum AuthenticatorAttachment).

requireResidentKey, of type boolean, defaulting to false
This member describes the Relying Parties' requirements regarding availability of the Client-side-resident Credential Private Key. If the parameter is set to true, the authenticator MUST create a Client-side-resident Credential Private Key when creating a public key credential.

userVerification, of type UserVerificationRequirement, defaulting to "preferred"
This member describes the Relying Party's requirements regarding user verification for the create() operation. Eligible

MUST perform user verification when performing the create() operation and future 4.1.4 Use an existing credential to make an assertion - PublicKeyCredential's [[DiscoverFromExternalSource]](options) method operations when it is requested to verify the credential.

Note: These identifiers are intentionally short, rather than descriptive, because they will be serialized into a message to the authenticator, which may be sent over a low-bandwidth link.

4.4.4. Authenticator Attachment enumeration (enum AuthenticatorAttachment)

```
enum AuthenticatorAttachment {  
  "plat", // Platform attachment  
  "xplat" // Cross-platform attachment  
};
```

Clients may communicate with authenticators using a variety of mechanisms. For example, a client may use a platform-specific API to communicate with an authenticator which is physically bound to a platform. On the other hand, a client may use a variety of standardized cross-platform transport protocols such as Bluetooth (see 4.7.4 Authenticator Transport enumeration (enum AuthenticatorTransport)) to discover and communicate with cross-platform attached authenticators. Therefore, we use AuthenticatorAttachment to describe an authenticator's attachment modality. We define authenticators that are part of the client's platform as having a platform attachment, and refer to them as platform authenticators. While those that are reachable via cross-platform transport protocols are defined as having cross-platform attachment, and refer to them as roaming authenticators.

- * platform attachment - the respective authenticator is attached using platform-specific transports. Usually, authenticators of this class are non-removable from the platform.
- * cross-platform attachment - the respective authenticator is attached using cross-platform transports. Authenticators of this class are removable from, and can "roam" among, client platforms.

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

4.5. Options for Assertion Generation (dictionary)

authenticators are filtered to only those capable of satisfying this requirement.

5.4.5. Authenticator Attachment enumeration (enum AuthenticatorAttachment)

```
enum AuthenticatorAttachment {  
  "platform", // Platform attachment  
  "cross-platform" // Cross-platform attachment  
};
```

Clients may communicate with authenticators using a variety of mechanisms. For example, a client may use a platform-specific API to communicate with an authenticator which is physically bound to a platform. On the other hand, a client may use a variety of standardized cross-platform transport protocols such as Bluetooth (see 5.8.4 Authenticator Transport enumeration (enum AuthenticatorTransport)) to discover and communicate with cross-platform attached authenticators. Therefore, we use AuthenticatorAttachment to describe an authenticator's attachment modality. We define authenticators that are part of the client's platform as having a platform attachment, and refer to them as platform authenticators. While those that are reachable via cross-platform transport protocols are defined as having cross-platform attachment, and refer to them as roaming authenticators.

- * platform attachment - the respective authenticator is attached using platform-specific transports. Usually, authenticators of this class are non-removable from the platform.
- * cross-platform attachment - the respective authenticator is attached using cross-platform transports. Authenticators of this class are removable from, and can "roam" among, client platforms.

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

5.4.6. Attestation Conveyance Preference enumeration (enum AttestationConveyancePreference)

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

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

- * none - indicates that the Relying Party is not interested in authenticator attestation. The client may replace the AAGUID and attestation statement generated by the authenticator with meaningless client-generated values. For example, in order to avoid having to obtain user consent to relay uniquely identifying information to the Relying Party, or to save a roundtrip to a Privacy CA. This is the default value.
- * indirect - indicates that the Relying Party prefers an attestation

```
1568 PublicKeyCredentialRequestOptions)
1569
1570 The PublicKeyCredentialRequestOptions dictionary supplies get() with
1571 the data it needs to generate an assertion. Its challenge member must
1572 be present, while its other members are optional.
1573 dictionary PublicKeyCredentialRequestOptions {
1574   required BufferSource challenge;
1575   unsigned long timeout;
1576   USVString rpId;
1577   sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
1578
1579   AuthenticationExtensions extensions;
1580 };
1581
1582 challenge, of type BufferSource
1583 This member represents a challenge that the selected
1584 authenticator signs, along with other data, when producing an
1585 authentication assertion.
1586
1587 timeout, of type unsigned long
1588 This optional member specifies a time, in milliseconds, that the
1589 caller is willing to wait for the call to complete. The value is
1590 treated as a hint, and may be overridden by the platform.
1591
1592 rpId, of type USVString
1593 This optional member specifies the relying party identifier
1594 claimed by the caller. If omitted, its value will be the
1595 CredentialsContainer object's relevant settings object's
1596 origin's effective domain.
1597
1598 allowCredentials, of type sequence<PublicKeyCredentialDescriptor>,
1599 defaulting to None
1600 This optional member contains a list of
1601 PublicKeyCredentialDescriptor object representing public key
1602 credentials acceptable to the caller, in decending order of the
1603 caller's preference (the first item in the list is the most
1604 preferred credential, and so on down the list).
1605
1606 extensions, of type AuthenticationExtensions
1607 This optional member contains additional parameters requesting
1608 additional processing by the client and authenticator. For
1609 example, if transaction confirmation is sought from the user,
1610 then the prompt string might be included as an extension.
1611
1612 4.6. Authentication Extensions (typedef AuthenticationExtensions)
```

```
2024 conveyance yielding verifiable attestation statements, but allows
2025 the client to decide how to obtain such attestation statements. The
2026 client may replace the authenticator-generated attestation
2027 statements with attestation statements generated by a Privacy CA,
2028 in order to protect the user's privacy, or to assist Relying
2029 Parties with attestation verification in a heterogeneous ecosystem.
2030 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 rpId;
2046   sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
2047   UserVerificationRequirement userVerification = "preferred";
2048   AuthenticationExtensions extensions;
2049 };
2050
2051 challenge, of type BufferSource
2052 This member represents a challenge that the selected
2053 authenticator signs, along with other data, when producing an
2054 authentication assertion. See the 13.1 Cryptographic Challenges
2055 security consideration.
2056
2057 timeout, of type unsigned long
2058 This optional member specifies a time, in milliseconds, that the
2059 caller is willing to wait for the call to complete. The value is
2060 treated as a hint, and may be overridden by the platform.
2061
2062 rpId, of type USVString
2063 This optional member specifies the relying party identifier
2064 claimed by the caller. If omitted, its value will be the
2065 CredentialsContainer object's relevant settings object's
2066 origin's effective domain.
2067
2068 allowCredentials, of type sequence<PublicKeyCredentialDescriptor>,
2069 defaulting to None
2070 This optional member contains a list of
2071 PublicKeyCredentialDescriptor objects representing public key
2072 credentials acceptable to the caller, in decending order of the
2073 caller's preference (the first item in the list is the most
2074 preferred credential, and so on down the list).
2075
2076 userVerification, of type UserVerificationRequirement, defaulting to
2077 "preferred"
2078 This member describes the Relying Party's requirements regarding
2079 user verification for the get() operation. Eligible
2080 authenticators are filtered to only those capable of satisfying
2081 this requirement.
2082
2083 extensions, of type AuthenticationExtensions
2084 This optional member contains additional parameters requesting
2085 additional processing by the client and authenticator. For
2086 example, if transaction confirmation is sought from the user,
2087 then the prompt string might be included as an extension.
2088
2089 5.6. Abort operations with AbortSignal
2090
2091 Developers are encouraged to leverage the AbortController to manage the
2092 [[Create]](origin, options, sameOriginWithAncestors) and
2093 [[DiscoverFromExternalSource]](origin, options,
```

```
1612
1613
1614
1615
1616     This is a dictionary containing zero or more WebAuthn extensions, as
1617     defined in 8 WebAuthn Extensions. An AuthenticationExtensions instance
1618     can contain either client extensions or authenticator extensions,
1619     depending upon context.
1620
1621     4.7. Supporting Data Structures
1622
1623     The public key credential type uses certain data structures that are
1624     specified in supporting specifications. These are as follows.
1625
1626     4.7.1. Client data used in WebAuthn signatures (dictionary
1627     CollectedClientData)
1628
1629     The client data represents the contextual bindings of both the Relying
1630     Party and the client platform. It is a key-value mapping with
1631     string-valued keys. Values may be any type that has a valid encoding in
1632     JSON. Its structure is defined by the following Web IDL.
1633     dictionary CollectedClientData {
1634         required DOMString      challenge;
1635         required DOMString      origin;
1636         required DOMString      hashAlgorithm;
1637         DOMString                tokenBindingId;
1638         AuthenticationExtensions clientExtensions;
1639         AuthenticationExtensions authenticatorExtensions;
1640     };
```

```
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
```

sameOriginWithAncestors) operations. See DOM 3.3 Using AbortController and AbortSignal objects in APIs section for detailed instructions.

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

The visibility and focus state of the Window object determines whether the `[[Create]](origin, options, sameOriginWithAncestors)` and `[[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors)` operations should continue. When the Window object associated with the [Document loses focus, `[[Create]](origin, options, sameOriginWithAncestors)` and `[[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors)` operations SHOULD be aborted.

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.

5.7. Authentication Extensions (typedef AuthenticationExtensions)

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

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

5.8. Supporting Data Structures

The public key credential type uses certain data structures that are specified in supporting specifications. These are as follows.

5.8.1. Client data used in WebAuthn signatures (dictionary CollectedClientData)

The client data represents the contextual bindings of both the Relying Party and the client platform. It is a key-value mapping with string-valued keys. Values may be any type that has a valid encoding in JSON. Its structure is defined by the following Web IDL.

```
dictionary CollectedClientData {
    required DOMString type;
    required DOMString challenge;
    required DOMString origin;
    required DOMString hashAlgorithm;
    DOMString tokenBindingId;
    AuthenticationExtensions clientExtensions;
    AuthenticationExtensions authenticatorExtensions;
};
```

The type member contains the string "webauthn.create" when creating new credentials, and "webauthn.get" when getting an assertion from an existing credential. The purpose of this member is to prevent certain types of signature confusion attacks (where an attacker substitutes one

1641 The challenge member contains the base64url encoding of the challenge
1642 provided by the RP.

1643
1644 The origin member contains the fully qualified origin of the requester,
1645 as provided to the authenticator by the client, in the syntax defined
1646 by [RFC6454].

1647
1648 The hashAlgorithm member is a recognized algorithm name that supports
1649 the "digest" operation, which specifies the algorithm used to compute
1650 the hash of the serialized client data. This algorithm is chosen by the
1651 client at its sole discretion.

1652
1653 The tokenBindingId member contains the base64url encoding of the Token
1654 Binding ID that this client uses for the Token Binding protocol when
1655 communicating with the Relying Party. This can be omitted if no Token
1656 Binding has been negotiated between the client and the Relying Party.

1657
1658 The optional clientExtensions and authenticatorExtensions members
1659 contain additional parameters generated by processing the extensions
1660 passed in by the Relying Party. WebAuthn extensions are detailed in
1661 Section 8 WebAuthn Extensions.

1662
1663 This structure is used by the client to compute the following
1664 quantities:

1665
1666 JSON-serialized client data
1667 This is the UTF-8 encoding of the result of calling the initial
1668 value of JSON.stringify on a CollectedClientData dictionary.

1669
1670 Hash of the serialized client data
1671 This is the hash (computed using hashAlgorithm) of the
1672 JSON-serialized client data, as constructed by the client.

1673
1674 4.7.2. Credential Type enumeration (enum PublicKeyCredentialType)

1675
1676 enum PublicKeyCredentialType {
1677 "public-key"
1678 };

1679
1680 This enumeration defines the valid credential types. It is an extension
1681 point; values may be added to it in the future, as more credential
1682 types are defined. The values of this enumeration are used for
1683 versioning the Authentication Assertion and attestation structures
1684 according to the type of the authenticator.

1685
1686 Currently one credential type is defined, namely "public-key".

1687
1688 4.7.3. Credential Descriptor (dictionary PublicKeyCredentialDescriptor)

1689
1690 dictionary PublicKeyCredentialDescriptor {
1691 required PublicKeyCredentialType type;
1692 required BufferSource id;
1693 sequence<AuthenticatorTransport> transports;
1694 };

1695
1696 This dictionary contains the attributes that are specified by a caller
1697 when referring to a credential as an input parameter to the create() or
1698 get() methods. It mirrors the fields of the PublicKeyCredential object
1699 returned by the latter methods.

1700
1701 The type member contains the type of the credential the caller is
1702 referring to.

1703
1704 The id member contains the identifier of the credential that the caller
1705 is referring to.

1706
1707 4.7.4. Authenticator Transport enumeration (enum AuthenticatorTransport)

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
2192 JSON-serialized client data
2193 This is the UTF-8 encoding of the result of calling the initial
2194 value of JSON.stringify on a CollectedClientData dictionary.

2195
2196 Hash of the serialized client data
2197 This is the hash (computed using hashAlgorithm) of the
2198 JSON-serialized client data, as constructed by the client.

2199
2200 5.8.2. Credential Type enumeration (enum PublicKeyCredentialType)

2201
2202 enum PublicKeyCredentialType {
2203 "public-key"
2204 };

2205
2206 This enumeration defines the valid credential types. It is an extension
2207 point; values may be added to it in the future, as more credential
2208 types are defined. The values of this enumeration are used for
2209 versioning the Authentication Assertion and attestation structures
2210 according to the type of the authenticator.

2211
2212 Currently one credential type is defined, namely "public-key".

2213
2214 5.8.3. Credential Descriptor (dictionary PublicKeyCredentialDescriptor)

2215
2216 dictionary PublicKeyCredentialDescriptor {
2217 required PublicKeyCredentialType type;
2218 required BufferSource id;
2219 sequence<AuthenticatorTransport> transports;
2220 };

2221
2222 This dictionary contains the attributes that are specified by a caller
2223 when referring to a credential as an input parameter to the create() or
2224 get() methods. It mirrors the fields of the PublicKeyCredential object
2225 returned by the latter methods.

2226
2227 The type member contains the type of the credential the caller is
2228 referring to.

2229
2230 The id member contains the identifier of the credential that the caller
2231 is referring to.

2232
2233 5.8.4. Authenticator Transport enumeration (enum AuthenticatorTransport)


```
1708 enum AuthenticatorTransport {
1709     "usb",
1710     "nfc",
1711     "ble",
1712 };
1713
1714
1715 Authenticators may communicate with Clients using a variety of
1716 transports. This enumeration defines a hint as to how Clients might
1717 communicate with a particular Authenticator in order to obtain an
1718 assertion for a specific credential. Note that these hints represent
1719 the Relying Party's best belief as to how an Authenticator may be
1720 reached. A Relying Party may obtain a list of transports hints from
1721 some attestation statement formats or via some out-of-band mechanism;
1722 it is outside the scope of this specification to define that mechanism.
1723 * usb - the respective Authenticator may be contacted over USB.
1724 * nfc - the respective Authenticator may be contacted over Near Field
1725 Communication (NFC).
1726 * ble - the respective Authenticator may be contacted over Bluetooth
1727 Smart (Bluetooth Low Energy / BLE).
1728
1729 4.7.5. Cryptographic Algorithm Identifier (typedef COSEAlgorithmIdentifier)
1730
1731 typedef long COSEAlgorithmIdentifier;
1732
1733 A COSEAlgorithmIdentifier's value is a number identifying a
1734 cryptographic algorithm. The algorithm identifiers SHOULD be values
1735 registered in the IANA COSE Algorithms registry [IANA-COSE-ALGS-REG],
1736 for instance, -7 for "ES256" and -257 for "RS256".
1737
1738 5. WebAuthn Authenticator model
```

```
1739
1740 The API defined in this specification implies a specific abstract
1741 functional model for an authenticator. This section describes the
1742 authenticator model.
1743
1744 Client platforms may implement and expose this abstract model in any
1745 way desired. However, the behavior of the client's Web Authentication
1746 API implementation, when operating on the authenticators supported by
1747 that platform, MUST be indistinguishable from the behavior specified in
1748 4 Web Authentication API.
1749
1750 For authenticators, this model defines the logical operations that they
1751 must support, and the data formats that they expose to the client and
1752 the Relying Party. However, it does not define the details of how
```

```
2234 enum AuthenticatorTransport {
2235     "usb",
2236     "nfc",
2237     "ble",
2238 };
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
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
2291 The API defined in this specification implies a specific abstract
2292 functional model for an authenticator. This section describes the
2293 authenticator model.
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
```

1753 authenticators communicate with the client platform, unless they are
1754 required for interoperability with Relying Parties. For instance, this
1755 abstract model does not define protocols for connecting authenticators
1756 to clients over transports such as USB or NFC. Similarly, this abstract
1757 model does not define specific error codes or methods of returning
1758 them; however, it does define error behavior in terms of the needs of
1759 the client. Therefore, specific error codes are mentioned as a means of
1760 showing which error conditions must be distinguishable (or not) from
1761 each other in order to enable a compliant and secure client
1762 implementation.

1763
1764 In this abstract model, the authenticator provides key management and
1765 cryptographic signatures. It may be embedded in the WebAuthn client, or
1766 housed in a separate device entirely. The authenticator may itself
1767 contain a cryptographic module which operates at a higher security
1768 level than the rest of the authenticator. This is particularly
1769 important for authenticators that are embedded in the WebAuthn client,
1770 as in those cases this cryptographic module (which may, for example, be
1771 a TPM) could be considered more trustworthy than the rest of the
1772 authenticator.

1773
1774 Each authenticator stores some number of public key credentials. Each
1775 public key credential has an identifier which is unique (or extremely
1776 unlikely to be duplicated) among all public key credentials. Each
1777 credential is also associated with a Relying Party, whose identity is
1778 represented by a Relying Party Identifier (RP ID).

1779
1780 Each authenticator has an AAGUID, which is a 128-bit identifier that
1781 indicates the type (e.g. make and model) of the authenticator. The
1782 AAGUID MUST be chosen by the manufacturer to be identical across all
1783 substantially identical authenticators made by that manufacturer, and
1784 different (with probability $1-2^{-128}$ or greater) from the AAGUIDs of
1785 all other types of authenticators. The RP MAY use the AAGUID to infer
1786 certain properties of the authenticator, such as certification level
1787 and strength of key protection, using information from other sources.

1788
1789 The primary function of the authenticator is to provide WebAuthn
1790 signatures, which are bound to various contextual data. These data are
1791 observed, and added at different levels of the stack as a signature
1792 request passes from the server to the authenticator. In verifying a
1793 signature, the server checks these bindings against expected values.
1794 These contextual bindings are divided in two: Those added by the RP or
1795 the client, referred to as client data; and those added by the
1796 authenticator, referred to as the authenticator data. The authenticator
1797 signs over the client data, but is otherwise not interested in its
1798 contents. To save bandwidth and processing requirements on the
1799 authenticator, the client hashes the client data and sends only the
1800 result to the authenticator. The authenticator signs over the
1801 combination of the hash of the serialized client data, and its own
1802 authenticator data.

1803
1804 The goals of this design can be summarized as follows.
1805 * The scheme for generating signatures should accommodate cases where
1806 the link between the client platform and authenticator is very
1807 limited, in bandwidth and/or latency. Examples include Bluetooth
1808 Low Energy and Near-Field Communication.
1809 * The data processed by the authenticator should be small and easy to
1810 interpret in low-level code. In particular, authenticators should
1811 not have to parse high-level encodings such as JSON.
1812 * Both the client platform and the authenticator should have the
1813 flexibility to add contextual bindings as needed.
1814 * The design aims to reuse as much as possible of existing encoding
1815 formats in order to aid adoption and implementation.

1816
1817 Authenticators produce cryptographic signatures for two distinct
1818 purposes:
1819 1. An attestation signature is produced when a new public key
1820 credential is created via an authenticatorMakeCredential operation.
1821 An attestation signature provides cryptographic proof of certain
1822 properties of the the authenticator and the credential. For

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
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
2368 Authenticators produce cryptographic signatures for two distinct
2369 purposes:
2370 1. An attestation signature is produced when a new public key
2371 credential is created via an authenticatorMakeCredential operation.
2372 An attestation signature provides cryptographic proof of certain
2373 properties of the the authenticator and the credential. For

instance, an attestation signature asserts the authenticator type (as denoted by its AAGUID) and the credential public key. The attestation signature is signed by an attestation private key, which is chosen depending on the type of attestation desired. For more details on attestation, see 5.3 Attestation.

2. An assertion signature is produced when the authenticatorGetAssertion method is invoked. It represents an assertion by the authenticator that the user has consented to a specific transaction, such as logging in, or completing a purchase. Thus, an assertion signature asserts that the authenticator possessing a particular credential private key has established, to the best of its ability, that the user requesting this transaction is the same user who consented to creating that particular public key credential. It also asserts additional information, termed client data, that may be useful to the caller, such as the means by which user consent was provided, and the prompt shown to the user by the authenticator. The assertion signature format is illustrated in Figure 2, below.

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

5.1. Authenticator data

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

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

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

Length (in bytes)	Description
32	SHA-256 hash of the RP ID associated with the credential.
1	Flags (bit 0 is the least significant bit):
	* Bit 0: User Present (UP) result.
	+ 1 means the user is present.
	+ 0 means the user is not present.
	* Bit 1: Reserved for future use (RFU1).
	* Bit 2: User Verified (UV) result.
	+ 1 means the user is verified.
	+ 0 means the user is not verified.
	* Bits 3-5: Reserved for future use (RFU2).
	* Bit 6: Attestation data included (AT).
	+ Indicates whether the authenticator added attestation data.
	* Bit 7: Extension data included (ED).
	+ Indicates if the authenticator data has extensions.

4 Signature counter (signCount), 32-bit unsigned big-endian integer. variable (if present) attestation data (if present). See 5.3.1 Attestation data for details. Its length depends on the length of the credential public key and credential ID being attested.

variable (if present) Extension-defined authenticator data. This is a CBOR [RFC7049] map with extension identifiers as keys, and authenticator extension outputs as values. See 8 WebAuthn Extensions for details.

instance, an attestation signature asserts the authenticator type (as denoted by its AAGUID) and the credential public key. The attestation signature is signed by an attestation private key, which is chosen depending on the type of attestation desired. For more details on attestation, see 6.3 Attestation.

2. An assertion signature is produced when the authenticatorGetAssertion method is invoked. It represents an assertion by the authenticator that the user has consented to a specific transaction, such as logging in, or completing a purchase. Thus, an assertion signature asserts that the authenticator possessing a particular credential private key has established, to the best of its ability, that the user requesting this transaction is the same user who consented to creating that particular public key credential. It also asserts additional information, termed client data, that may be useful to the caller, such as the means by which user consent was provided, and the prompt shown to the user by the authenticator. The assertion signature format is illustrated in Figure 2, below.

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

6.1. Authenticator data

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

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

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

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

signCount 4 Signature counter, 32-bit unsigned big-endian integer. attestedCredentialData variable (if present) attested credential data (if present). See 6.3.1 Attested credential data for details. Its length depends on the length of the credential ID and credential public key being attested.

extensions variable (if present) Extension-defined authenticator data. This is a CBOR [RFC7049] map with extension identifiers as keys, and authenticator extension outputs as values. See 9 WebAuthn Extensions for details.

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

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

For attestation signatures, the authenticator MUST set the AT flag and include the attestation data. For authentication signatures, the AT flag MUST NOT be set and the attestation data MUST NOT be included.

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

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

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

5.2. Authenticator operations

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 authenticator data.

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

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

For attestation signatures, the authenticator MUST set the AT flag and include the attestedCredentialData. For authentication signatures, the AT flag MUST NOT be set and the attestedCredentialData MUST NOT be included.

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

The figure below shows a visual representation of the authenticator data structure.
Authenticator data layout Authenticator data layout.

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

6.1.1. Signature Counter Considerations

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

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

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

Authenticators:
* should implement per-RP ID signature counters. This prevents the signature counter value from being shared between Relying Parties and being possibly employed as a correlation handle for the user. Authenticators may implement a global signature counter, i.e., on a per-authenticator basis, but this is less privacy-friendly for users.

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

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

5.2.1. The authenticatorMakeCredential operation

This operation must be invoked in an authenticator session which has no other operations in progress. It takes the following input parameters:

- * The caller's RP ID, as determined by the user agent and the client.
- * The hash of the serialized client data, provided by the client.
- * The Relying Party's PublicKeyCredentialEntity.
- * The user account's PublicKeyCredentialUserEntity.
- * A sequence of pairs of PublicKeyCredentialType and COSEAlgorithmIdentifier requested by the Relying Party. This sequence is ordered from most preferred to least preferred. The platform makes a best-effort to create the most preferred credential that it can.
- * An optional list of PublicKeyCredentialDescriptor objects provided by the Relying Party with the intention that, if any of these are known to the authenticator, it should not create a new credential.
- * The rk member of the options.authenticatorSelection dictionary.
- * The uv member of the options.authenticatorSelection dictionary.
- * Extension data created by the client based on the extensions

requested by the Relying Party, if any.

When this operation is invoked, the authenticator must perform the following procedure:

- * Check if all the supplied parameters are syntactically well-formed and of the correct length. If not, return an error code equivalent to "UnknownError" and terminate the operation.

* should ensure that the signature counter value does not accidentally decrease (e.g., due to hardware failures).

6.2. Authenticator operations

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

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

6.2.1. The authenticatorMakeCredential operation

It takes the following input parameters:

hash

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

rpEntity

The Relying Party's PublicKeyCredentialRpEntity.

userEntity

The user account's PublicKeyCredentialUserEntity, containing the user handle given by the Relying Party.

requireResidentKey

The authenticatorSelection.requireResidentKey value given by the Relying Party.

requireUserPresence

A Boolean value provided by the client, which in invocations from a WebAuthn Client's [[Create]](origin, options, sameOriginWithAncestors) method is always set to the inverse of requireUserVerification.

requireUserVerification

The effective user verification requirement for credential creation, a Boolean value provided by the client.

credTypesAndPubKeyAlgs

A sequence of pairs of PublicKeyCredentialType and public key algorithms (COSEAlgorithmIdentifier) requested by the Relying Party. This sequence is ordered from most preferred to least preferred. The platform makes a best-effort to create the most preferred credential that it can.

excludeCredentialDescriptorList

An optional list of PublicKeyCredentialDescriptor objects provided by the Relying Party with the intention that, if any of these are known to the authenticator, it should not create a new credential. excludeCredentialDescriptorList contains a list of known credentials.

extensions

A map from extension identifiers to their authenticator extension inputs, created by the client based on the extensions requested by the Relying Party, if any.

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

When this operation is invoked, the authenticator must perform the following procedure:

1. Check if all the supplied parameters are syntactically well-formed and of the correct length. If not, return an error code equivalent to "UnknownError" and terminate the operation.

1962 * Check if at least one of the specified combinations of
1963 PublicKeyCredentialType and cryptographic parameters is supported.
1964 If not, return an error code equivalent to "NotSupportedError" and
1965 terminate the operation.
1966 * Check if a credential matching any of the supplied
1967 PublicKeyCredential identifiers is present on this authenticator.
1968 If so, return an error code equivalent to "NotAllowedError" and
1969 terminate the operation.
1970 * If rk is true and the authenticator cannot store a

1971 Client-side-resident Credential Private Key, return an error code
1972 equivalent to "ConstraintError" and terminate the operation.
1973 * If uv is true and the authenticator cannot perform user
1974 verification, return an error code equivalent to "ConstraintError"
1975 and terminate the operation.
1976 * Prompt the user for consent to create a new credential. The prompt
1977 for obtaining this consent is shown by the authenticator if it has
1978 its own output capability, or by the user agent otherwise. If the
1979 user denies consent, return an error code equivalent to
1980 "NotAllowedError" and terminate the operation.
1981 * Once user consent has been obtained, generate a new credential

1982 object:
1983 + Generate a set of cryptographic keys using the most preferred
1984 combination of PublicKeyCredentialType and cryptographic
1985 parameters supported by this authenticator.
1986 + Generate an identifier for this credential, such that this
1987 identifier is globally unique with high probability across all

1988 credentials with the same type across all authenticators.
1989 + Associate the credential with the specified RP ID and the
1990 user's account identifier user.id.
1991 + Delete any older credentials with the same RP ID and user.id
1992 that are stored locally by the authenticator.
1993 * If any error occurred while creating the new credential object,

1994 return an error code equivalent to "UnknownError" and terminate the
1995 operation.
1996 * Process all the supported extensions requested by the client, and
1997 generate the authenticator data with attestation data as specified
1998 in 5.1 Authenticator data. Use this authenticator data and the
1999 hash of the serialized client data to create an attestation object
2000 for the new credential using the procedure specified in 5.3.4
2001 Generating an Attestation Object. For more details on attestation,
2002 see 5.3 Attestation.

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.
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 a global signature counter
2638 Use the global signature counter's actual value when
2639 generating authenticator data.
2640 a per credential signature counter
2641 allocate the counter, associate it with the new
2642 credential, and initialize the counter value as zero.
2643 11. Let attestedCredentialData be the attested credential data byte
2644 array including the credentialId and publicKey.
2645 12. Let authenticatorData be the byte array specified in 6.1
2646 Authenticator data, including attestedCredentialData as the
2647 attestedCredentialData and processedExtensions, if any, as the
2648 extensions.
2649 13. Return the attestation object for the new credential created by the

2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015

On successful completion of this operation, the authenticator returns the attestation object to the client.

5.2.2. The authenticatorGetAssertion operation

This operation must be invoked in an authenticator session which has no other operations in progress. It takes the following input parameters:

- * The caller's RP ID, as determined by the user agent and the client.
- * The hash of the serialized client data, provided by the client.
- * A list of credentials acceptable to the Relying Party (possibly filtered by the client), if any.
- * Extension data created by the client based on the extensions requested by the Relying Party, if any.

2016
2017

When this method is invoked, the authenticator must perform the following procedure:

- * Check if all the supplied parameters are syntactically well-formed and of the correct length. If not, return an error code equivalent to "UnknownError" and terminate the operation.
- * If a list of credentials was supplied by the client, filter it by removing those credentials that are not present on this authenticator. If no list was supplied, create a list with all credentials stored for the caller's RP ID (as determined by an exact match of the RP ID).
- * If the previous step resulted in an empty list, return an error code equivalent to "NotAllowedError" and terminate the operation.
- * Prompt the user to select a credential from among the above list. Obtain user consent for using this credential. The prompt for

2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723

procedure specified in 6.3.4 Generating an Attestation Object using an authenticator-chosen attestation statement format, authenticatorData, and hash. For more details on attestation, see 6.3 Attestation.

On successful completion of this operation, the authenticator returns the attestation object to the client.

6.2.2. The authenticatorGetAssertion operation

It takes the following input parameters:

rpid

The caller's RP ID, as determined by the user agent and the client.

hash

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

allowCredentialDescriptorList

An optional list of PublicKeyCredentialDescriptors describing credentials acceptable to the Relying Party (possibly filtered by the client), if any.

requireUserPresence

A Boolean value provided by the client, which in invocations from a WebAuthn Client's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method is always set to the inverse of requireUserVerification.

requireUserVerification

The effective user verification requirement for assertion, a Boolean value provided by the client.

extensions

A map from extension identifiers to their authenticator extension inputs, created by the client based on the extensions requested by the Relying Party, if any.

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

When this method is invoked, the authenticator must perform the following procedure:

1. Check if all the supplied parameters are syntactically well-formed and of the correct length. If not, return an error code equivalent to "UnknownError" and terminate the operation.
2. If requireUserVerification is true and the authenticator cannot perform user verification, return an error code equivalent to "ConstraintError" and terminate the operation.
3. If allowCredentialDescriptorList was not supplied, set it to a list of all credentials stored for rpid (as determined by an exact match of rpid).
4. Remove any items from allowCredentialDescriptorList that do not match a credential bound to this authenticator. A match occurs if a credential matches rpid and an allowCredentialDescriptorList item's id and type members.
5. If allowCredentialDescriptorList is now empty, return an error code equivalent to "NotAllowedError" and terminate the operation.
6. Let selectedCredential be a credential as follows. If the size of allowCredentialDescriptorList

is exactly 1

Let selectedCredential be the credential matching allowCredentialDescriptorList[0].

is greater than 1

Prompt the user to select selectedCredential from the credentials matching the items in

obtaining this consent may be shown by the authenticator if it has its own output capability, or by the user agent otherwise.

* Process all the supported extensions requested by the client, and generate the authenticator data as specified in 5.1 Authenticator data, though without attestation data. Concatenate this authenticator data with the hash of the serialized client data to generate an assertion signature using the private key of the selected credential as shown in Figure 2, below. A simple, undelimited concatenation is safe to use here because the authenticator data describes its own length. The hash of the serialized client data (which potentially has a variable length) is always the last element.

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

[fido-signature-formats-figure2.svg] Generating an assertion signature.

On successful completion, the authenticator returns to the user agent:

- * The identifier of the credential (credential ID) used to generate the assertion signature.
- * The authenticator data used to generate the assertion signature.
- * The assertion signature.

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

If the user refuses consent, the authenticator returns an appropriate error status to the client.

5.2.3. The authenticatorCancel operation

This operation takes no input parameters and returns no result.

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

This operation is ignored if it is invoked in an authenticator session which does not have an authenticatorMakeCredential or

allowCredentialDescriptorList.

7. Obtain user consent for using selectedCredential. The prompt for obtaining this consent may be shown by the authenticator if it has its own output capability, or by the user agent otherwise. The prompt SHOULD display the rpId and any additional displayable data associated with selectedCredential, if possible. If requireUserVerification is true, the method of obtaining user consent MUST include user verification. If requireUserPresence is true, the method of obtaining user consent MUST include a test of user presence. If the user denies consent or if user verification fails, return an error code equivalent to "NotAllowedError" and terminate the operation.

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

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

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

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

Generating an assertion signature

Generating an assertion signature.

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

13. Return to the user agent:

- + selectedCredential's credential ID, if either a list of credentials of size 2 or greater was supplied by the client, or no such list was supplied. Otherwise, return only the below values.

Note: If the client supplies a list of exactly one credential and it was successfully employed, then its credential ID is not returned since the client already knows it. This saves transmitting these bytes over what may be a constrained connection in what is likely a common case.

- + authenticatorData
- + signature
- + The user handle associated with selectedCredential.

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

6.2.3. The authenticatorCancel operation

This operation takes no input parameters and returns no result.

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

This operation is ignored if it is invoked in an authenticator session which does not have an authenticatorMakeCredential or

2077 authenticatorGetAssertion operation currently in progress.
2078
2079 **5.3. Attestation**
2080
2081 Authenticators must also provide some form of attestation. The basic
2082 requirement is that the authenticator can produce, for each credential
2083 public key, an attestation statement verifiable by the Relying Party.
2084 Typically, this attestation statement contains a signature by an
2085 attestation private key over the attested credential public key and a
2086 challenge, as well as a certificate or similar data providing
2087 provenance information for the attestation public key, enabling the
2088 Relying Party to make a trust decision. However, if an attestation key
2089 pair is not available, then the authenticator MUST perform self
2090 attestation of the credential public key with the corresponding
2091 credential private key. All this information is returned by
2092 authenticators any time a new public key credential is generated, in
2093 the overall form of an attestation object. The relationship of the
2094 attestation object with authenticator data (containing attestation
2095 data) and the attestation statement is illustrated in figure 3, below.
2096 **Attestation Object Layout diagram Attestation object layout**
2097 **illustrating the included authenticator data (containing attestation**
2098 **data) and the attestation statement.**

2099
2100 This figure illustrates only the packed attestation statement format.
2101 Several additional attestation statement formats are defined in 7
2102 Defined Attestation Statement Formats.

2103
2104 An important component of the attestation object is the attestation
2105 statement. This is a specific type of signed data object, containing
2106 statements about a public key credential itself and the authenticator
2107 that created it. It contains an attestation signature created using the
2108 key of the attesting authority (except for the case of self
2109 attestation, when it is created using the credential private key). In
2110 order to correctly interpret an attestation statement, a Relying Party
2111 needs to understand these two aspects of attestation:
2112 1. The attestation statement format is the manner in which the
2113 signature is represented and the various contextual bindings are
2114 incorporated into the attestation statement by the authenticator.
2115 In other words, this defines the syntax of the statement. Various
2116 existing devices and platforms (such as TPMs and the Android OS)
2117 have previously defined attestation statement formats. This
2118 specification supports a variety of such formats in an extensible
2119 way, as defined in 5.3.2 Attestation Statement Formats.
2120 2. The attestation type defines the semantics of attestation
2121 statements and their underlying trust models. Specifically, it
2122 defines how a Relying Party establishes trust in a particular
2123 attestation statement, after verifying that it is cryptographically
2124 valid. This specification supports a number of attestation types,
2125 as described in 5.3.3 Attestation Types.
2126
2127 In general, there is no simple mapping between attestation statement
2128 formats and attestation types. For example, the "packed" attestation
2129 statement format defined in 7.2 Packed Attestation Statement Format
2130 can be used in conjunction with all attestation types, while other
2131 formats and types have more limited applicability.
2132
2133 The privacy, security and operational characteristics of attestation
2134 depend on:
2135 * The attestation type, which determines the trust model,
2136 * The attestation statement format, which may constrain the strength
2137 of the attestation by limiting what can be expressed in an
2138 attestation statement, and
2139 * The characteristics of the individual authenticator, such as its
2140 construction, whether part or all of it runs in a secure operating
2141 environment, and so on.
2142
2143 It is expected that most authenticators will support a small number of
2144 attestation types and attestation statement formats, while Relying

2791 authenticatorGetAssertion operation currently in progress.
2792
2793 **6.3. Attestation**
2794
2795 Authenticators must also provide some form of attestation. The basic
2796 requirement is that the authenticator can produce, for each credential
2797 public key, an attestation statement verifiable by the Relying Party.
2798 Typically, this attestation statement contains a signature by an
2799 attestation private key over the attested credential public key and a
2800 challenge, as well as a certificate or similar data providing
2801 provenance information for the attestation public key, enabling the
2802 Relying Party to make a trust decision. However, if an attestation key
2803 pair is not available, then the authenticator MUST perform self
2804 attestation of the credential public key with the corresponding
2805 credential private key. All this information is returned by
2806 authenticators any time a new public key credential is generated, in
2807 the overall form of an attestation object. The relationship of the
2808 attestation object with authenticator data (containing attested
2809 credential data) and the attestation statement is illustrated in figure
2810 3, below.
2811 **Attestation object layout illustrating the included authenticator data**
2812 **(containing attested credential data) and the attestation statement.**
2813 **Attestation object layout illustrating the included authenticator data**
2814 **(containing attested credential data) and the attestation statement.**

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

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
2843 In general, there is no simple mapping between attestation statement
2844 formats and attestation types. For example, the "packed" attestation
2845 statement format defined in 8.2 Packed Attestation Statement Format
2846 can be used in conjunction with all attestation types, while other
2847 formats and types have more limited applicability.
2848
2849 The privacy, security and operational characteristics of attestation
2850 depend on:
2851 * The attestation type, which determines the trust model,
2852 * The attestation statement format, which may constrain the strength
2853 of the attestation by limiting what can be expressed in an
2854 attestation statement, and
2855 * The characteristics of the individual authenticator, such as its
2856 construction, whether part or all of it runs in a secure operating
2857 environment, and so on.
2858
2859 It is expected that most authenticators will support a small number of
2860 attestation types and attestation statement formats, while Relying

Parties will decide what attestation types are acceptable to them by policy. Relying Parties will also need to understand the characteristics of the authenticators that they trust, based on information they have about these authenticators. For example, the FIDO Metadata Service [FIDOMetadataService] provides one way to access such information.

5.3.1. Attestation data

Attestation data is added to the authenticator data when generating an attestation object for a given credential. It has the following format:

Length (in bytes)	Description
16	The AAGUID of the authenticator.
2	Byte length L of Credential ID
L	Credential ID
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.

5.3.2. Attestation Statement Formats

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

- * Attestation statement format identifier:
- * Supported attestation types:
- * Syntax: The syntax of an attestation statement produced in this format, defined using [CDDL] for the extension point \$attStmtFormat defined in 5.3.4 Generating an Attestation Object.
- * Signing procedure: The signing procedure for computing an attestation statement in this format given the public key credential to be attested, the authenticator data structure containing the authenticator data for the attestation, and the hash of the serialized client data.
- * Verification procedures: The procedure for verifying an attestation statement, which takes as inputs the authenticator data structure containing the authenticator data claimed to have been used for the attestation and the hash of the serialized client data, and returns either:

- + An error indicating that the attestation is invalid, or
- + The attestation type, and the trust path of the attestation. This trust path is either empty (in case of self attestation), an identifier of a ECDAA-Issuer public key (in the case of ECDAA), or a set of X.509 certificates.

The initial list of specified attestation statement formats is in 7 Defined Attestation Statement Formats.

5.3.3. Attestation Types

WebAuthn supports multiple attestation types:

Basic Attestation

In the case of basic attestation [UAFProtocol], the authenticator's attestation key pair is specific to an authenticator model. Thus, authenticators of the same model often share the same attestation key pair. See 5.3.5.1 Privacy for futher information.

Parties will decide what attestation types are acceptable to them by policy. Relying Parties will also need to understand the characteristics of the authenticators that they trust, based on information they have about these authenticators. For example, the FIDO Metadata Service [FIDOMetadataService] provides one way to access such 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.
credentialLength	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.

6.3.2. Attestation Statement Formats

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

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

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

6.3.3. Attestation Types

WebAuthn supports multiple attestation types:

Basic Attestation

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

2208 Self Attestation
2209 In the case of self attestation, also known as surrogate basic
2210 attestation [UAFProtocol], the Authenticator does not have any
2211 specific attestation key. Instead it uses the authentication key
2212 itself to create the attestation signature. Authenticators
2213 without meaningful protection measures for an attestation
2214 private key typically use this attestation type.

2215 Privacy CA
2216 In this case, the Authenticator owns an authenticator-specific
2217 (endorsement) key. This key is used to securely communicate with
2218 a trusted third party, the Privacy CA. The Authenticator can
2219 generate multiple attestation key pairs and asks the Privacy CA
2220 to issue an attestation certificate for it. Using this approach,
2221 the Authenticator can limit the exposure of the endorsement key
2222 (which is a global correlation handle) to Privacy CA(s).
2223 Attestation keys can be requested for each public key credential
2224 individually.

2225 Note: This concept typically leads to multiple attestation
2226 certificates. The attestation certificate requested most
2227 recently is called "active".

2228 Elliptic Curve based Direct Anonymous Attestation (ECDAA)
2229 In this case, the Authenticator receives direct anonymous
2230 attestation (DAA) credentials from a single DAA-Issuer. These
2231 DAA credentials are used along with blinding to sign the
2232 attestation data. The concept of blinding avoids the DAA
2233 credentials being misused as global correlation handle. WebAuthn
2234 supports DAA using elliptic curve cryptography and bilinear
2235 pairings, called ECDAA (see [FIDOEcdaaAlgorithm]) in this
2236 specification. Consequently we denote the DAA-Issuer as
2237 ECDAA-Issuer (see [FIDOEcdaaAlgorithm]).

2238 5.3.4. Generating an Attestation Object
2239
2240 This section specifies the algorithm for generating an attestation
2241 object (see: Figure 3) for any attestation statement format.

2242 In order to construct an attestation object for a given public key
2243 credential using a particular attestation statement format, the
2244 authenticator MUST first generate the authenticator data.

2245 The authenticator MUST then run the signing procedure for the desired
2246 attestation statement format with this authenticator data and the hash
2247 of the serialized client data as input, and use this to construct an
2248 attestation statement in that attestation statement format.

2249 Finally, the authenticator MUST construct the attestation object as a
2250 CBOR map with the following syntax:



```
2258 attObj = {  
2259   authData: bytes,  
2260   $$attStmtType  
2261 }  
2262  
2263 attStmtTemplate = (  
2264   fmt: text,  
2265   attStmt: bytes  
2266 )  
2267  
2268 ; Every attestation statement format must have the above fields
```

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 Privacy CA
2939 In this case, the Authenticator owns an authenticator-specific
2940 (endorsement) key. This key is used to securely communicate with
2941 a trusted third party, the Privacy CA. The Authenticator can
2942 generate multiple attestation key pairs and asks the Privacy CA
2943 to issue an attestation certificate for it. Using this approach,
2944 the Authenticator can limit the exposure of the endorsement key
2945 (which is a global correlation handle) to Privacy CA(s).
2946 Attestation keys can be requested for each public key credential
2947 individually.

2948 Note: This concept typically leads to multiple attestation
2949 certificates. The attestation certificate requested most
2950 recently is called "active".

2951 Elliptic Curve based Direct Anonymous Attestation (ECDAA)
2952 In this case, the Authenticator receives direct anonymous
2953 attestation (DAA) credentials from a single DAA-Issuer. These
2954 DAA credentials are used along with blinding to sign the
2955 attested credential data. The concept of blinding avoids the DAA
2956 credentials being misused as global correlation handle. WebAuthn
2957 supports DAA using elliptic curve cryptography and bilinear
2958 pairings, called ECDAA (see [FIDOEcdaaAlgorithm]) in this
2959 specification. Consequently we denote the DAA-Issuer as
2960 ECDAA-Issuer (see [FIDOEcdaaAlgorithm]).

2961 6.3.4. Generating an Attestation Object
2962
2963 To generate an attestation object (see: Figure 3) given:

2964 attestationFormat
2965 An attestation statement format.

2966
2967 authData
2968 A byte array containing authenticator data.

2969
2970 hash
2971 The hash of the serialized client data.

2972 the authenticator MUST:
2973 1. Let attStmt be the result of running attestationFormat's signing
2974 procedure given authData and hash.
2975 2. Let fmt be attestationFormat's attestation statement format
2976 identifier
2977 3. Return the attestation object as a CBOR map with the following
2978 syntax, filled in with variables initialized by this algorithm:

```
2979 attObj = {  
2980   authData: bytes,  
2981   $$attStmtType  
2982 }  
2983  
2984 attStmtTemplate = (  
2985   fmt: text,  
2986   attStmt: { * tstr => any } ; Map is filled in by each  
2987 concrete attStmtType  
2988 )  
2989  
2990 ; Every attestation statement format must have the above fields
```


attStmtTemplate .within \$attStmtType

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

fmt
The attestation statement format identifier associated with the attestation statement. Each attestation statement format defines its identifier.

authData
The authenticator data used to generate the attestation statement.

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

5.3.5. Security Considerations

5.3.5.1. 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.

5.3.5.2. 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.

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

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.

If attestation certificate validation fails due to a revoked intermediate attestation CA certificate, and the Relying Party's policy requires rejecting the registration/authentication request in these situations, then it is recommended that the Relying Party also un-registers (or marks with a trust level equivalent to "self

attestation") public key credentials that were registered after the CA compromise date using an attestation certificate chaining up to the same intermediate CA. It is thus recommended that Relying Parties remember intermediate attestation CA certificates during Authenticator registration in order to un-register related public key credentials if the registration was performed after revocation of such certificates.

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

5.3.5.3. Attestation Certificate Hierarchy

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

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

6. Relying Party Operations

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

6.1. Registering a new credential

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

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

attestation") public key credentials that were registered after the CA compromise date using an attestation certificate chaining up to the same intermediate CA. It is thus recommended that Relying Parties remember intermediate attestation CA certificates during Authenticator registration in order to un-register related public key credentials if the registration was performed after revocation of such certificates.

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

6.3.5.3. Attestation Certificate Hierarchy

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

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

7. Relying Party Operations

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

7.1. Registering a new credential

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

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

2408 10. Verify that attStmt is a correct, **validly-signed attestation**
2409 **statement, using the attestation statement format fmt's**
2410 **verification procedure given authenticator data** authData and the
2411 **hash of the serialized client data computed in step 6.**
2412 11. If validation is successful, obtain a list of acceptable trust

2413 anchors (attestation root certificates or ECDAAs-Issuer public keys)
2414 for that attestation type and attestation statement format fmt,
2415 from a trusted source or from policy. For example, the FIDO
2416 Metadata Service [FIDOMetadataService] provides one way to obtain
2417 such information, using the **AAGUID** in the attestation data
2418 **contained in** authData.
2419 12. Assess the attestation trustworthiness using the outputs of the
2420 verification procedure in step 10, as follows:
2421 + If self attestation was used, check if self attestation is
2422 acceptable under Relying Party policy.
2423 + If ECDAAs were used, verify that the identifier of the
2424 ECDAAs-Issuer public key used is included in the set of
2425 acceptable trust anchors obtained in step 11.
2426 + Otherwise, use the X.509 certificates returned by the
2427 verification procedure to verify that the attestation public
2428 key correctly chains up to an acceptable root certificate.
2429 13. If the attestation statement attStmt verified successfully and is
2430 found to be trustworthy, then register the new credential with the
2431 account that was denoted in the options.user passed to create(), by
2432 associating it with the credential ID and credential public key
2433 **contained in authData's attestation data, as appropriate for the**
2434 **Relying Party's systems.**
2435 14. If the attestation statement attStmt successfully verified but is
2436 not trustworthy per step 12 above, the Relying Party SHOULD fail
2437 the registration ceremony.
2438 NOTE: However, if permitted by policy, the Relying Party MAY
2439 register the credential ID and credential public key but treat the
2440 credential as one with self attestation (see 5.3.3 Attestation
2441 Types). If doing so, the Relying Party is asserting there is no
2442 cryptographic proof that the public key credential has been
2443 generated by a particular authenticator model. See [FIDOSecRef] and
2444 [UAFProtocol] for a more detailed discussion.
2445 15. If verification of the attestation statement failed, the Relying
2446 **Party MUST fail the registration ceremony.**
2447
2448 Verification of attestation objects requires that the Relying Party has
2449 a trusted method of determining acceptable trust anchors in step 11
2450 above. Also, if certificates are being used, the Relying Party must
2451 have access to certificate status information for the intermediate CA
2452 certificates. The Relying Party must also be able to build the
2453 attestation certificate chain if the client did not provide this chain
2454 in the attestation information.
2455
2456 To avoid ambiguity during authentication, the Relying Party SHOULD
2457 check that each credential is registered to no more than one user. If
2458 registration is requested for a credential that is already registered
2459 to a different user, the Relying Party SHOULD fail this ceremony, or it
2460 MAY decide to accept the registration, e.g. while deleting the older
2461 registration.
2462
2463 6.2. Verifying an authentication assertion
2464
2465 When verifying a given PublicKeyCredential structure (credential) as
2466 part of an authentication ceremony, the Relying Party MUST proceed as
2467 follows:
2468 1. Using credential's id attribute (or the corresponding rawId, if
2469 base64url encoding is inappropriate for your use case), look up the
2470 corresponding credential public key.
2471 2. Let cData, aData and sig denote the value of credential's
2472 response's clientDataJSON, authenticatorData, and signature
2473 respectively.

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 ECDAAs-Issuer public keys)
3132 for that attestation type and attestation statement format fmt,
3133 from a trusted source or from policy. For example, the FIDO
3134 Metadata Service [FIDOMetadataService] provides one way to obtain
3135 such information, using the **aaguid** in the **attestedCredentialData** in
3136 **authData.**
3137 13. Assess the attestation trustworthiness using the outputs of the
3138 verification procedure in step 10, as follows:
3139 + If self attestation was used, check if self attestation is
3140 acceptable under Relying Party policy.
3141 + If ECDAAs were used, verify that the identifier of the
3142 ECDAAs-Issuer public key used is included in the set of
3143 acceptable trust anchors obtained in step 11.
3144 + Otherwise, use the X.509 certificates returned by the
3145 verification procedure to verify that the attestation public
3146 key correctly chains up to an acceptable root certificate.
3147 14. If the attestation statement attStmt verified successfully and is
3148 found to be trustworthy, then register the new credential with the
3149 account that was denoted in the options.user passed to create(), by
3150 associating it with the credentialId and credentialPublicKey in the
3151 **attestedCredentialData** in authData, **as appropriate for the Relying**
3152 **Party's system.**
3153 15. If the attestation statement attStmt successfully verified but is
3154 not trustworthy per step 12 above, the Relying Party SHOULD fail
3155 the registration ceremony.
3156 NOTE: However, if permitted by policy, the Relying Party MAY
3157 register the credential ID and credential public key but treat the
3158 credential as one with self attestation (see 6.3.3 Attestation
3159 Types). If doing so, the Relying Party is asserting there is no
3160 cryptographic proof that the public key credential has been
3161 generated by a particular authenticator model. See [FIDOSecRef] and
3162 [UAFProtocol] for a more detailed discussion.

3163 Verification of attestation objects requires that the Relying Party has
3164 a trusted method of determining acceptable trust anchors in step 11
3165 above. Also, if certificates are being used, the Relying Party must
3166 have access to certificate status information for the intermediate CA
3167 certificates. The Relying Party must also be able to build the
3168 attestation certificate chain if the client did not provide this chain
3169 in the attestation information.
3170
3171 To avoid ambiguity during authentication, the Relying Party SHOULD
3172 check that each credential is registered to no more than one user. If
3173 registration is requested for a credential that is already registered
3174 to a different user, the Relying Party SHOULD fail this ceremony, or it
3175 MAY decide to accept the registration, e.g. while deleting the older
3176 registration.
3177
3178 7.2. Verifying an authentication assertion
3179
3180 When verifying a given PublicKeyCredential structure (credential) as
3181 part of an authentication ceremony, the Relying Party MUST proceed as
3182 follows:
3183 1. Using credential's id attribute (or the corresponding rawId, if
3184 base64url encoding is inappropriate for your use case), look up the
3185 corresponding credential public key.
3186 2. Let cData, aData and sig denote the value of credential's
3187 response's clientDataJSON, authenticatorData, and signature
3188 respectively.

2474 3. Perform JSON deserialization on cData to extract the client data C
2475 used for the signature.
2476 4. Verify that the challenge member of C matches the challenge that
was sent to the authenticator in the
PublicKeyCredentialRequestOptions passed to the get() call.
2477 5. Verify that the origin member of C matches the Relying Party's
2478 origin.
2479 6. Verify that the tokenBindingId member of C (if present) matches the
2480 Token Binding ID for the TLS connection over which the signature
2481 was obtained.
2482 7. Verify that the clientExtensions member of C is a proper subset of
2483 the extensions requested by the Relying Party and that the
2484 authenticatorExtensions in C is also a proper subset of the
2485 extensions requested by the Relying Party.
2486 8. Verify that the RP ID hash in aData is the SHA-256 hash of the RP
2487 ID expected by the Relying Party.
2488 9. Let hash be the result of computing a hash over the cData using the
2489 algorithm represented by the hashAlgorithm member of C.
2490 10. Using the credential public key looked up in step 1, verify that
2491 sig is a valid signature over the binary concatenation of aData and
2492 hash.
2493 11. If all the above steps are successful, continue with the
2494 authentication ceremony as appropriate. Otherwise, fail the
2495 authentication ceremony.

2496 7. Defined Attestation Statement Formats
2497 WebAuthn supports pluggable attestation statement formats. This section
2498 defines an initial set of such formats.

2499 7.1. Attestation Statement Format Identifiers
2500 Attestation statement formats are identified by a string, called a
2501 attestation statement format identifier, chosen by the author of the
2502 attestation statement format.
2503 Attestation statement format identifiers SHOULD be registered per
2504 [WebAuthn-Registries] "Registries for Web Authentication (WebAuthn)".
2505 All registered attestation statement format identifiers are unique
2506 amongst themselves as a matter of course.
2507 Unregistered attestation statement format identifiers SHOULD use
2508 lowercase reverse domain-name naming, using a domain name registered by
2509 the developer, in order to assure uniqueness of the identifier. All
2510 attestation statement format identifiers MUST be a maximum of 32 octets
2511 in length and MUST consist only of printable USASCII characters,

3190 3. Perform JSON deserialization on cData to extract the client data C
3191 used for the signature.
3192 4. Verify that the type in C is the string webauthn.get.
3193 5. Verify that the challenge member of C matches the challenge that
3194 was sent to the authenticator in the
3195 PublicKeyCredentialRequestOptions passed to the get() call.
3196 6. Verify that the origin member of C matches the Relying Party's
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 greater than the signature counter value stored in
3217 conjunction with credential's id attribute.
3218 Update the stored signature counter value,
3219 associated with credential's id attribute, to be the
3220 value of adata.signCount.
3221 less than or equal to the signature counter value stored in
3222 conjunction with credential's id attribute.
3223 This is a signal that the authenticator may be
3224 cloned, i.e. at least two copies of the credential
3225 private key may exist and are being used in
3226 parallel. Relying Parties should incorporate this
3227 information into their risk scoring. Whether the
3228 Relying Party updates the stored signature counter
3229 value in this case, or not, or fails the
3230 authentication ceremony or not, is Relying
3231 Party-specific.
3232 13. If all the above steps are successful, continue with the
3233 authentication ceremony as appropriate. Otherwise, fail the
3234 authentication ceremony.

3235 8. Defined Attestation Statement Formats
3236 WebAuthn supports pluggable attestation statement formats. This section
3237 defines an initial set of such formats.

3238 8.1. Attestation Statement Format Identifiers
3239 Attestation statement formats are identified by a string, called a
3240 attestation statement format identifier, chosen by the author of the
3241 attestation statement format.
3242 Attestation statement format identifiers SHOULD be registered per
3243 [WebAuthn-Registries] "Registries for Web Authentication (WebAuthn)".
3244 All registered attestation statement format identifiers are unique
3245 amongst themselves as a matter of course.
3246 Unregistered attestation statement format identifiers SHOULD use
3247 lowercase reverse domain-name naming, using a domain name registered by
3248 the developer, in order to assure uniqueness of the identifier. All
3249 attestation statement format identifiers MUST be a maximum of 32 octets
3250 in length and MUST consist only of printable USASCII characters,

2520 excluding backslash and doublequote, i.e., VCHAR as defined in
2521 [RFC5234] but without %x22 and %x5c.
2522
2523 Note: This means attestation statement format identifiers based on
2524 domain names MUST incorporate only LDH Labels [RFC5890].
2525
2526 Implementations MUST match WebAuthn attestation statement format
2527 identifiers in a case-sensitive fashion.
2528
2529 Attestation statement formats that may exist in multiple versions
2530 SHOULD include a version in their identifier. In effect, different
2531 versions are thus treated as different formats, e.g., packed2 as a new
2532 version of the packed attestation statement format.
2533
2534 The following sections present a set of currently-defined and
2535 registered attestation statement formats and their identifiers. The
2536 up-to-date list of registered WebAuthn Extensions is maintained in the
2537 IANA "WebAuthn Attestation Statement Format Identifier" registry
2538 established by [WebAuthn-Registries].
2539
2540 **7.2. Packed Attestation Statement Format**
2541
2542 This is a WebAuthn optimized attestation statement format. It uses a
2543 very compact but still extensible encoding method. It is implementable
2544 by authenticators with limited resources (e.g., secure elements).
2545
2546 Attestation statement format identifier
2547 packed
2548
2549 Attestation types supported
2550 All
2551
2552 Syntax
2553 The syntax of a Packed Attestation statement is defined by the
2554 following CDDL:
2555
2556 \$\$attStmtType ::= (
2557 fmt: "packed",
2558 attStmt: packedStmtFormat
2559)
2560
2561 packedStmtFormat = {
2562 alg: rsaAlgName / eccAlgName,
2563 sig: bytes,
2564 x5c: [attestnCert: bytes, * (caCert: bytes)]
2565 } //
2566 alg: "ED256" / "ED512",
2567 sig: bytes,
2568 ecdaaKeyld: bytes
2569 }
2570
2571 The semantics of the fields are as follows:
2572
2573 alg
2574 A text string containing the name of the algorithm used to
2575 generate the attestation signature. The types rsaAlgName
2576 and eccAlgName are as defined in 5.3.1 Attestation data.
2577 "ED256" and "ED512" refer to algorithms defined in
2578 [FIDOecdaaAlgorithm].
2579
2580 sig
2581 A byte string containing the attestation signature.
2582
2583 x5c
2584 The elements of this array contain the attestation
2585 certificate and its certificate chain, each encoded in
2586 X.509 format. The attestation certificate must be the
2587 first element in the array.
2588

3260 excluding backslash and doublequote, i.e., VCHAR as defined in
3261 [RFC5234] but without %x22 and %x5c.
3262
3263 Note: This means attestation statement format identifiers based on
3264 domain names MUST incorporate only LDH Labels [RFC5890].
3265
3266 Implementations MUST match WebAuthn attestation statement format
3267 identifiers in a case-sensitive fashion.
3268
3269 Attestation statement formats that may exist in multiple versions
3270 SHOULD include a version in their identifier. In effect, different
3271 versions are thus treated as different formats, e.g., packed2 as a new
3272 version of the packed attestation statement format.
3273
3274 The following sections present a set of currently-defined and
3275 registered attestation statement formats and their identifiers. The
3276 up-to-date list of registered WebAuthn Extensions is maintained in the
3277 IANA "WebAuthn Attestation Statement Format Identifier" registry
3278 established by [WebAuthn-Registries].
3279
3280 **8.2. Packed Attestation Statement Format**
3281
3282 This is a WebAuthn optimized attestation statement format. It uses a
3283 very compact but still extensible encoding method. It is implementable
3284 by authenticators with limited resources (e.g., secure elements).
3285
3286 Attestation statement format identifier
3287 packed
3288
3289 Attestation types supported
3290 All
3291
3292 Syntax
3293 The syntax of a Packed Attestation statement is defined by the
3294 following CDDL:
3295
3296 \$\$attStmtType ::= (
3297 fmt: "packed",
3298 attStmt: packedStmtFormat
3299)
3300
3301 packedStmtFormat = {
3302 alg: COSEAlgorithmIdentifier,
3303 sig: bytes,
3304 x5c: [attestnCert: bytes, * (caCert: bytes)]
3305 } //
3306 alg: COSEAlgorithmIdentifier, (-260 for ED256 / -261
3307 for ED512)
3308 sig: bytes,
3309 ecdaaKeyld: bytes
3310 }
3311
3312 The semantics of the fields are as follows:
3313
3314 alg
3315 A COSEAlgorithmIdentifier containing the identifier of the
3316 algorithm used to generate the attestation signature.
3317
3318 sig
3319 A byte string containing the attestation signature.
3320
3321 x5c
3322 The elements of this array contain the attestation
3323 certificate and its certificate chain, each encoded in
3324 X.509 format. The attestation certificate must be the
3325 first element in the array.
3326

2589 ecdaaKeyld
 2590 The identifier of the ECDAAs-Issuer public key. This is the
 2591 BigIntegerToB encoding of the component "c" of the
 2592 ECDAAs-Issuer public key as defined section 3.3, step 3.5
 2593 in [FIDOEcdaaAlgorithm].
 2594
 2595
 2596 Signing procedure
 2597 The signing procedure for this attestation statement format is
 2598 similar to the procedure for generating assertion signatures.
 2599
 2600 Let authenticatorData denote the authenticator data for the
 2601 attestation, and let clientDataHash denote the hash of the
 2602 serialized client data.
 2603
 2604 If Basic or Privacy CA attestation is in use, the authenticator
 2605 produces the sig by concatenating authenticatorData and
 2606 clientDataHash, and signing the result using an attestation
 2607 private key selected through an authenticator-specific
 2608 mechanism. It sets x5c to the certificate chain of the
 2609 attestation public key and alg to the algorithm of the
 2610 attestation private key.
 2611
 2612 If ECDAAs is in use, the authenticator produces sig by
 2613 concatenating authenticatorData and clientDataHash, and signing
 2614 the result using ECDAAs-Sign (see section 3.5 of
 2615 [FIDOEcdaaAlgorithm]) with a ECDAAs-Issuer public key selected
 2616 through an authenticator-specific mechanism (see
 2617 [FIDOEcdaaAlgorithm]). It sets alg to the algorithm of the
 2618 ECDAAs-Issuer public key and ecdaaKeyld to the identifier of the
 2619 ECDAAs-Issuer public key (see above).
 2620
 2621 If self attestation is in use, the authenticator produces sig by
 2622 concatenating authenticatorData and clientDataHash, and signing
 2623 the result using the credential private key. It sets alg to the
 2624 algorithm of the credential private key, and omits the other
 2625 fields.
 2626
 2627 Verification procedure
 2628 Verify that the given attestation statement is valid CBOR
 2629 conforming to the syntax defined above.
 2630
 2631 Let authenticatorData denote the authenticator data claimed to
 2632 have been used for the attestation, and let clientDataHash
 2633 denote the hash of the serialized client data.
 2634
 2635 If x5c is present, this indicates that the attestation type is
 not ECDAAs. In this case:
 + Verify that sig is a valid signature over the concatenation of
 authenticatorData and clientDataHash using the attestation
 public key in x5c with the algorithm specified in alg.
 + Verify that x5c meets the requirements in 7.2.1 Packed
 attestation statement certificate requirements.
 + If x5c contains an extension with OID 1 3 6 1 4 1 45724 1 1 4
 (id-fido-gen-ce-aaguid) verify that the value of this
 extension matches the AAGUID in authenticatorData.
 + If successful, return attestation type Basic and trust path
 x5c.
 If ecdaaKeyld is present, then the attestation type is ECDAAs. In
 this case:
 + Verify that sig is a valid signature over the concatenation of
 authenticatorData and clientDataHash using ECDAAs-Verify with
 ECDAAs-Issuer public key identified by ecdaaKeyld (see
 [FIDOEcdaaAlgorithm]).

3327 ecdaaKeyld
 3328 The identifier of the ECDAAs-Issuer public key. This is the
 3329 BigIntegerToB encoding of the component "c" of the
 3330 ECDAAs-Issuer public key as defined section 3.3, step 3.5
 3331 in [FIDOEcdaaAlgorithm].
 3332
 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 ECDAAs is in use, the authenticator produces sig by
 3349 concatenating authenticatorData and clientDataHash, and
 3350 signing the result using ECDAAs-Sign (see section 3.5 of
 3351 [FIDOEcdaaAlgorithm]) after selecting an ECDAAs-Issuer public
 3352 key related to the ECDAAs signature private key through an
 3353 authenticator-specific mechanism (see [FIDOEcdaaAlgorithm]).
 3354 It sets alg to the algorithm of the selected ECDAAs-Issuer
 3355 public key and ecdaaKeyld to the identifier of the
 ECDAAs-Issuer public key (see above).
 4. If self attestation is in use, the authenticator produces sig
 by concatenating authenticatorData and clientDataHash, and
 signing the result using the credential private key. It sets
 alg to the algorithm of the credential private key, and omits
 the other fields.
 Verification procedure
 Given the verification procedure inputs attStmt,
 authenticatorData and clientDataHash, the verification procedure
 is as follows:
 1. Verify that attStmt is valid CBOR conforming to the syntax
 defined above, and perform CBOR decoding on it to extract the
 contained fields.
 2. If x5c is present, this indicates that the attestation type is
 not ECDAAs. In this case:
 o Verify that sig is a valid signature over the
 concatenation of authenticatorData and clientDataHash
 using the attestation public key in x5c with the
 algorithm specified in alg.
 o Verify that x5c meets the requirements in 8.2.1 Packed
 attestation statement certificate requirements.
 o If x5c contains an extension with OID 1 3 6 1 4 1 45724 1
 1 4 (id-fido-gen-ce-aaguid) verify that the value of this
 extension matches the aaguid in authenticatorData.
 o If successful, return attestation type Basic and
 attestation trust path x5c.
 3. If ecdaaKeyld is present, then the attestation type is ECDAAs.
 In this case:
 o Verify that sig is a valid signature over the
 concatenation of authenticatorData and clientDataHash
 using ECDAAs-Verify with ECDAAs-Issuer public key
 identified by ecdaaKeyld (see [FIDOEcdaaAlgorithm]).
 o If successful, return attestation type ECDAAs and
 attestation trust path ecdaaKeyld.

2656 + If successful, return attestation type ECDA and trust path
2657 ecdaaKeyld.
2658
2659 If neither x5c nor ecdaaKeyld is present, self attestation is in
2660 use.
2661
2662 + Validate that alg matches the algorithm of the credential
2663 private key in authenticatorData.
2664 + Verify that sig is a valid signature over the concatenation of
2665 authenticatorData and clientDataHash using the credential
2666 public key with alg.
2667 + If successful, return attestation type Self and empty trust
2668 path.
2669
2670 7.2.1. Packed attestation statement certificate requirements
2671
2672 The attestation certificate MUST have the following fields/extensions:
2673 * Version must be set to 3.
2674 * Subject field MUST be set to:
2675
2676 Subject-C
2677 Country where the Authenticator vendor is incorporated
2678
2679 Subject-O
2680 Legal name of the Authenticator vendor
2681
2682 Subject-OU
2683 Authenticator Attestation
2684
2685 Subject-CN
2686 No stipulation.
2687
2688 * If the related attestation root certificate is used for multiple
2689 authenticator models, the Extension OID 1 3 6 1 4 1 45724 1 1 4
2690 (id-fido-gen-ce-aaguid) MUST be present, containing the AAGUID as
2691 value.
2692 * The Basic Constraints extension MUST have the CA component set to
2693 false
2694 * An Authority Information Access (AIA) extension with entry
2695 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are
2696 both optional as the status of many attestation certificates is
2697 available through authenticator metadata services. See, for
2698 example, the FIDO Metadata Service [FIDOMetadataService].
2699
2700 7.3. TPM Attestation Statement Format
2701
2702 This attestation statement format is generally used by authenticators
2703 that use a Trusted Platform Module as their cryptographic engine.
2704
2705 Attestation statement format identifier
2706 tpm
2707
2708 Attestation types supported
2709 Privacy CA, ECDA
2710
2711 Syntax
2712 The syntax of a TPM Attestation statement is as follows:
2713
2714 \$\$attStmtType // = (
2715 fmt: "tpm",
2716 attStmt: tpmStmtFormat
2717)
2718
2719 tpmStmtFormat = {
2720 ver: "2.0",
2721 (
2722 alg: rsaAlgName / eccAlgName,
2723 x5c: [aikCert: bytes, * (caCert: bytes)]
2724) //
2725 }

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
3402 8.2.1. Packed attestation statement certificate requirements
3403
3404 The attestation certificate MUST have the following fields/extensions:
3405 * Version must be set to 3.
3406 * Subject field MUST be set to:
3407
3408 Subject-C
3409 Country where the Authenticator vendor is incorporated
3410
3411 Subject-O
3412 Legal name of the Authenticator vendor
3413
3414 Subject-OU
3415 Authenticator Attestation
3416
3417 Subject-CN
3418 No stipulation.
3419
3420 * If the related attestation root certificate is used for multiple
3421 authenticator models, the Extension OID 1 3 6 1 4 1 45724 1 1 4
3422 (id-fido-gen-ce-aaguid) MUST be present, containing the AAGUID as
3423 value.
3424 * The Basic Constraints extension MUST have the CA component set to
3425 false
3426 * An Authority Information Access (AIA) extension with entry
3427 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are
3428 both optional as the status of many attestation certificates is
3429 available through authenticator metadata services. See, for
3430 example, the FIDO Metadata Service [FIDOMetadataService].
3431
3432 8.3. TPM Attestation Statement Format
3433
3434 This attestation statement format is generally used by authenticators
3435 that use a Trusted Platform Module as their cryptographic engine.
3436
3437 Attestation statement format identifier
3438 tpm
3439
3440 Attestation types supported
3441 Privacy CA, ECDA
3442
3443 Syntax
3444 The syntax of a TPM Attestation statement is as follows:
3445
3446 \$\$attStmtType // = (
3447 fmt: "tpm",
3448 attStmt: tpmStmtFormat
3449)
3450
3451 tpmStmtFormat = {
3452 ver: "2.0",
3453 (
3454 alg: COSEAlgorithmIdentifier,
3455 x5c: [aikCert: bytes, * (caCert: bytes)]
3456) //
3457 }

alg: "ED256" / "ED512",
 ecdaaKeyId: bytes
),
 sig: bytes,
 certInfo: bytes,
 pubArea: bytes
 }
 The semantics of the above fields are as follows:
 ver
 The version of the TPM specification to which the signature conforms.
 alg
 The name of the algorithm used to generate the attestation signature. The types rsaAlgName and eccAlgName are as defined in 5.3.1 Attestation data. The types "ED256" and "ED512" refer to the algorithms specified in [FIDOEcdaaAlgorithm].
 x5c
 The AIK certificate used for the attestation and its certificate chain, in X.509 encoding.
 ecdaaKeyId
 The identifier of the ECDAA-Issuer public key. This is the BigNumberToB encoding of the component "c" as defined section 3.3, step 3.5 in [FIDOEcdaaAlgorithm].
 sig
 The attestation signature, in the form of a TPMT_SIGNATURE structure as specified in [TPMv2-Part2] section 11.3.4.
 certInfo
 The TPMS_ATTEST structure over which the above signature was computed, as specified in [TPMv2-Part2] section 10.12.8.
 pubArea
 The TPMT_PUBLIC structure (see [TPMv2-Part2] section 12.2.4) used by the TPM to represent the credential public key.
 Signing procedure
 Let authenticatorData denote the authenticator data for the attestation, and let clientDataHash denote the hash of the serialized client data.
 Concatenate authenticatorData and clientDataHash to form attToBeSigned.
 Generate a signature using the procedure specified in [TPMv2-Part3] Section 18.2, using the attestation private key and setting the qualifyingData parameter to attToBeSigned.
 Set the pubArea field to the public area of the credential public key, the certInfo field to the output parameter of the same name, and the sig field to the signature obtained from the above procedure.
 Verification procedure
 Verify that the given attestation statement is valid CBOR conforming to the syntax defined above.

alg: COSEAlgorithmIdentifier, (-260 for ED256 / -261 for ED512)
 ecdaaKeyId: bytes
),
 sig: bytes,
 certInfo: bytes,
 pubArea: bytes
 }
 The semantics of the above fields are as follows:
 ver
 The version of the TPM specification to which the signature conforms.
 alg
 A COSEAlgorithmIdentifier containing the identifier of the algorithm used to generate the attestation signature.
 x5c
 The AIK certificate used for the attestation and its certificate chain, in X.509 encoding.
 ecdaaKeyId
 The identifier of the ECDAA-Issuer public key. This is the BigNumberToB encoding of the component "c" as defined section 3.3, step 3.5 in [FIDOEcdaaAlgorithm].
 sig
 The attestation signature, in the form of a TPMT_SIGNATURE structure as specified in [TPMv2-Part2] section 11.3.4.
 certInfo
 The TPMS_ATTEST structure over which the above signature was computed, as specified in [TPMv2-Part2] section 10.12.8.
 pubArea
 The TPMT_PUBLIC structure (see [TPMv2-Part2] section 12.2.4) used by the TPM to represent the credential public key.
 Signing procedure
 Let authenticatorData denote the authenticator data for the attestation, and let clientDataHash denote the hash of the serialized client data.
 Concatenate authenticatorData and clientDataHash to form attToBeSigned.
 Generate a signature using the procedure specified in [TPMv2-Part3] Section 18.2, using the attestation private key and setting the extraData parameter to the digest of attToBeSigned using the hash algorithm corresponding to the "alg" signature algorithm. (For the "RS256" algorithm, this would be a SHA-256 digest.)
 Set the pubArea field to the public area of the credential public key, the certInfo field to the output parameter of the same name, and the sig field to the signature obtained from the above procedure.
 Verification procedure
 Given the verification procedure inputs attStmt, authenticatorData and clientDataHash, the verification procedure is as follows:

2791 Let authenticatorData denote the authenticator data claimed to
2792 have been used for the attestation, and let clientDataHash
2793 denote the hash of the serialized client data.
2794
2795 Verify that the public key specified by the parameters and
2796 unique fields of pubArea is identical to the public key
2797 contained in the attestation data inside authenticatorData.
2798
2799 Concatenate authenticatorData and clientDataHash to form
2800 attToBeSigned.
2801
2802 Validate that certInfo is valid:
2803
2804 + Verify that magic is set to TPM_GENERATED_VALUE.
2805 + Verify that type is set to TPM_ST_ATTEST_CERTIFY.
2806 + Verify that extraData is set to attToBeSigned.
2807
2808 + Verify that attested contains a TPMS_CERTIFY_INFO structure,
2809 whose name field contains a valid Name for pubArea, as
2810 computed using the algorithm in the nameAlg field of pubArea
2811 using the procedure specified in [TPMv2-Part1] section 16.
2812
2813 If x5c is present, this indicates that the attestation type is
2814 not ECDA. In this case:
2815
2816 + Verify the sig is a valid signature over certInfo using the
2817 attestation public key in x5c with the algorithm specified in
2818 alg.
2819 + Verify that x5c meets the requirements in 7.3.1 TPM
2820 attestation statement certificate requirements.
2821 + If x5c contains an extension with OID 1 3 6 1 4 1 45724 1 1 4
2822 (id-fido-gen-ce-aaguid) verify that the value of this
2823 extension matches the AAGUID in authenticatorData.
2824 + If successful, return attestation type Privacy CA and trust
2825 path x5c.
2826
2827 If ecdaaKeyId is present, then the attestation type is ECDA.
2828
2829 + Perform ECDA-Verify on sig to verify that it is a valid
2830 signature over certInfo (see [FIDOEcdaaAlgorithm]).
2831 + If successful, return attestation type ECDA and the
2832 identifier of the ECDA-Issuer public key ecdaaKeyId.
2833
2834 7.3.1. TPM attestation statement certificate requirements
2835
2836 TPM attestation certificate MUST have the following fields/extensions:
2837 * Version must be set to 3.
2838 * Subject field MUST be set to empty.
2839 * The Subject Alternative Name extension must be set as defined in
2840 [TPMv2-EK-Profile] section 3.2.9.
2841 * The Extended Key Usage extension MUST contain the
2842 "joint-iso-itu-t(2) internationalorganizations(23) 133 tcg-kp(8)
2843 tcg-kp-AIKCertificate(3)" OID.
2844 * The Basic Constraints extension MUST have the CA component set to
2845 false.
2846 * An Authority Information Access (AIA) extension with entry
2847 id-ad-ocsp and a CRL Distribution Point extension [RFC5280] are
2848 both optional as the status of many attestation certificates is
2849 available through metadata services. See, for example, the FIDO
2850 Metadata Service [FIDOMetadataService].
2851
2852 7.4. Android Key Attestation Statement Format
2853
2854 When the authenticator in question is a platform-provided Authenticator
2855 on the Android "N" or later platform, the attestation statement is
2856 based on the Android key attestation. In these cases, the attestation
2857 statement is produced by a component running in a secure operating
2858 environment, but the authenticator data for the attestation is produced
2859 outside this environment. The Relying Party is expected to check that
2860 the authenticator data claimed to have been used for the attestation is

3525 Verify that attStmt is valid CBOR conforming to the syntax
3526 defined above, and perform CBOR decoding on it to extract the
3527 contained fields.
3528
3529 Verify that the public key specified by the parameters and
3530 unique fields of pubArea is identical to the credentialPublicKey
3531 in the attestedCredentialData in authenticatorData.
3532
3533 Concatenate authenticatorData and clientDataHash to form
3534 attToBeSigned.
3535
3536 Validate that certInfo is valid:
3537
3538 + Verify that magic is set to TPM_GENERATED_VALUE.
3539 + Verify that type is set to TPM_ST_ATTEST_CERTIFY.
3540 + Verify that extraData is set to the hash of attToBeSigned
3541 using the hash algorithm employed in "alg".
3542 + Verify that attested contains a TPMS_CERTIFY_INFO structure,
3543 whose name field contains a valid Name for pubArea, as
3544 computed using the algorithm in the nameAlg field of pubArea
3545 using the procedure specified in [TPMv2-Part1] section 16.
3546
3547 If x5c is present, this indicates that the attestation type is
3548 not ECDA. In this case:
3549
3550 + Verify the sig is a valid signature over certInfo using the
3551 attestation public key in x5c with the algorithm specified in
3552 alg.
3553 + Verify that x5c meets the requirements in 8.3.1 TPM
3554 attestation statement certificate requirements.
3555 + If x5c contains an extension with OID 1 3 6 1 4 1 45724 1 1 4
3556 (id-fido-gen-ce-aaguid) verify that the value of this
3557 extension matches the aaguid in authenticatorData.
3558 + If successful, return attestation type Privacy CA and
3559 attestation trust path x5c.
3560
3561 If ecdaaKeyId is present, then the attestation type is ECDA.
3562
3563 + Perform ECDA-Verify on sig to verify that it is a valid
3564 signature over certInfo (see [FIDOEcdaaAlgorithm]).
3565 + If successful, return attestation type ECDA and the
3566 identifier of the ECDA-Issuer public key ecdaaKeyId.
3567
3568 8.3.1. TPM attestation statement certificate requirements
3569
3570 TPM attestation certificate MUST have the following fields/extensions:
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
3586 8.4. Android Key Attestation Statement Format
3587
3588 When the authenticator in question is a platform-provided Authenticator
3589 on the Android "N" or later platform, the attestation statement is
3590 based on the Android key attestation. In these cases, the attestation
3591 statement is produced by a component running in a secure operating
3592 environment, but the authenticator data for the attestation is produced
3593 outside this environment. The Relying Party is expected to check that
3594 the authenticator data claimed to have been used for the attestation is

2860 consistent with the fields of the attestation certificate's extension
2861 data.
2862
2863 Attestation statement format identifier
2864 android-key
2865
2866 Attestation types supported
2867 Basic
2868
2869 Syntax
2870 An Android key attestation statement consists simply of the
2871 Android attestation statement, which is a series of DER encoded
2872 X.509 certificates. See the Android developer documentation. Its
2873 syntax is defined as follows:
2874
2875 `$$attStmtType //=(`
2876 `fmt: "android-key",`
2877 `attStmt: androidStmtFormat`
2878 `)`
2879
2880 `androidStmtFormat = bytes`

2881
2882 Signing procedure
2883 Let authenticatorData denote the authenticator data for the
2884 attestation, and let clientDataHash denote the hash of the
2885 serialized client data.
2886
2887 Concatenate authenticatorData and clientDataHash to form
2888 attToBeSigned.
2889
2890 Request an Android Key Attestation by calling
2891 "keyStore.getCertificateChain(myKeyUUID)") providing
2892 attToBeSigned as the challenge value (e.g., by using
2893 setAttestationChallenge), and set the attestation statement to
2894 the returned value.

2895
2896 Verification procedure
2897 Verification is performed as follows:
2898
2899 + Let authenticatorData denote the authenticator data claimed to
2900 have been used for the attestation, and let clientDataHash
2901 denote the hash of the serialized client data.
2902 + Verify that the public key in the first certificate in the
2903 series of certificates represented by the signature matches
2904 the credential public key in the attestation data field of
2905 authenticatorData.
2906 + Verify that in the attestation certificate extension data:
2907 o The value of the attestationChallenge field is identical
2908 to the concatenation of authenticatorData and
2909 clientDataHash.
2910 o The AuthorizationList.allApplications field is not
2911 present, since PublicKeyCredentials must be bound to the
2912 RP ID.
2913 o The value in the AuthorizationList.origin field is equal
2914 to KM_TAG_GENERATED.
2915 o The value in the AuthorizationList.purpose field is equal
2916 to KM_PURPOSE_SIGN.
2917 + If successful, return attestation type Basic with the trust
2918 path set to the entire attestation statement.

3595 consistent with the fields of the attestation certificate's extension
3596 data.
3597
3598 Attestation statement format identifier
3599 android-key
3600
3601 Attestation types supported
3602 Basic Attestation
3603
3604 Syntax
3605 An Android key attestation statement consists simply of the
3606 Android attestation statement, which is a series of DER encoded
3607 X.509 certificates. See the Android developer documentation. Its
3608 syntax is defined as follows:
3609
3610 `$$attStmtType //=(`
3611 `fmt: "android-key",`
3612 `attStmt: androidStmtFormat`
3613 `)`
3614
3615 `androidStmtFormat = {`
3616 `alg: COSEAlgorithmIdentifier,`
3617 `sig: bytes,`
3618 `x5c: [credCert: bytes, * (caCert: bytes)]`
3619 `}`
3620
3621
3622 Signing procedure
3623 Let authenticatorData denote the authenticator data for the
3624 attestation, and let clientDataHash denote the hash of the
3625 serialized client data.
3626

3627 Request an Android Key Attestation by calling
3628 "keyStore.getCertificateChain(myKeyUUID)") providing
3629 clientDataHash as the challenge value (e.g., by using
3630 setAttestationChallenge). Set x5c to the returned value.
3631
3632 The authenticator produces sig by concatenating
3633 authenticatorData and clientDataHash, and signing the result
3634 using the credential private key. It sets alg to the algorithm
3635 of the signature format.
3636
3637 Verification procedure
3638 Given the verification procedure inputs attStmt,
3639 authenticatorData and clientDataHash, the verification procedure
3640 is as follows:
3641
3642 + Verify that attStmt is valid CBOR conforming to the syntax
3643 defined above, and perform CBOR decoding on it to extract the
3644 contained fields.
3645 + Verify that the public key in the first certificate in the
3646 series of certificates represented by the signature matches
3647 the credentialPublicKey in the attestedCredentialData in
3648 authenticatorData.
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

2919
2920
2921
2922 When the authenticator in question is a platform-provided Authenticator
2923 on certain Android platforms, the attestation statement is based on the
2924 SafetyNet API. In this case the authenticator data is completely
2925 controlled by the caller of the SafetyNet API (typically an application
2926 running on the Android platform) and the attestation statement only
2927 provides some statements about the health of the platform and the
2928 identity of the calling application.

2929
2930 Attestation statement format identifier
2931 android-safetynet

2932
2933 Attestation types supported
2934 Basic

2935
2936 Syntax
2937 The syntax of an Android Attestation statement is defined as
2938 follows:

2939
2940 `$$attStmtType //= (`
2941 `fmt: "android-safetynet",`
2942 `attStmt: safetynetStmtFormat`
2943 `)`

2944
2945 `safetynetStmtFormat = {`
2946 `ver: text,`
2947 `response: bytes`
2948 `}`

2949
2950 The semantics of the above fields are as follows:

2951
2952 ver
2953 The version number of Google Play Services responsible for
2954 providing the SafetyNet API.

2955
2956 response
2957 The value returned by the above SafetyNet API. This value
2958 is a JWS [RFC7515] object (see SafetyNet online
2959 documentation) in Compact Serialization.

2960
2961 Signing procedure
2962 Let authenticatorData denote the authenticator data for the
2963 attestation, and let clientDataHash denote the hash of the
2964 serialized client data.

2965
2966 Concatenate authenticatorData and clientDataHash to form
2967 attToBeSigned.

2968
2969 Request a SafetyNet attestation, providing attToBeSigned as the
2970 nonce value. Set response to the result, and ver to the version
2971 of Google Play Services running in the authenticator.

2972
2973 Verification procedure
2974 Verification is performed as follows:

2975
2976 + Verify that the given attestation statement is valid CBOR
2977 conforming to the syntax defined above.

2978
2979 + Verify that response is a valid SafetyNet response of version
2980 ver.
2981 + Verify that the nonce in the response is identical to the

3662
3663 statement.

3664 8.5. Android SafetyNet Attestation Statement Format

3665
3666 When the authenticator in question is a platform-provided Authenticator
3667 on certain Android platforms, the attestation statement is based on the
3668 SafetyNet API. In this case the authenticator data is completely
3669 controlled by the caller of the SafetyNet API (typically an application
3670 running on the Android platform) and the attestation statement only
3671 provides some statements about the health of the platform and the
3672 identity of the calling application. This attestation does not provide
3673 information regarding provenance of the authenticator and its
3674 associated data. Therefore platform-provided authenticators should make
3675 use of the Android Key Attestation when available, even if the
3676 SafetyNet API is also present.

3677
3678 Attestation statement format identifier
3679 android-safetynet

3680
3681 Attestation types supported
3682 Basic Attestation

3683
3684 Syntax
3685 The syntax of an Android Attestation statement is defined as
3686 follows:

3687
3688 `$$attStmtType //= (`
3689 `fmt: "android-safetynet",`
3690 `attStmt: safetynetStmtFormat`
3691 `)`

3692
3693 `safetynetStmtFormat = {`
3694 `ver: text,`
3695 `response: bytes`
3696 `}`

3697
3698 The semantics of the above fields are as follows:

3699
3700 ver
3701 The version number of Google Play Services responsible for
3702 providing the SafetyNet API.

3703
3704 response
3705 The UTF-8 encoded result of the getJwsResult() call of the
3706 SafetyNet API. This value is a JWS [RFC7515] object (see
3707 SafetyNet online documentation) in Compact Serialization.

3708
3709 Signing procedure
3710 Let authenticatorData denote the authenticator data for the
3711 attestation, and let clientDataHash denote the hash of the
3712 serialized client data.

3713
3714 Concatenate authenticatorData and clientDataHash to form
3715 attToBeSigned.

3716
3717 Request a SafetyNet attestation, providing attToBeSigned as the
3718 nonce value. Set response to the result, and ver to the version
3719 of Google Play Services running in the authenticator.

3720
3721 Verification procedure
3722 Given the verification procedure inputs attStmt,
3723 authenticatorData and clientDataHash, the verification procedure
3724 is as follows:

3725
3726 + Verify that attStmt is valid CBOR conforming to the syntax
3727 defined above, and perform CBOR decoding on it to extract the
3728 contained fields.

3729
3730 + Verify that response is a valid SafetyNet response of version
3731 ver.
3732 + Verify that the nonce in the response is identical to the

concatenation of **the** authenticatorData and clientDataHash.
+ Verify that the attestation certificate is issued to the
hostname "attest.android.com" (see SafetyNet online
documentation).
+ Verify that the ctsProfileMatch attribute in the payload of
response is true.
+ If successful, return attestation type Basic with the **trust**
path set to the above attestation certificate.

7.6. FIDO U2F Attestation Statement Format

This attestation statement format is used with FIDO U2F authenticators
using the formats defined in [FIDO-U2F-Message-Formats].

Attestation statement format identifier
fido-u2f

Attestation types supported
Basic, **self** attestation

Syntax
The syntax of a FIDO U2F attestation statement is defined as
follows:

```

$$attStmtType //= (
    fmt: "fido-u2f",
    attStmt: u2fStmtFormat
)

u2fStmtFormat = {
    x5c: [ attestnCert: bytes, * (caCert: bytes) ],
    sig: bytes
}

```

The semantics of the above fields are as follows:

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

sig
The attestation signature.

Signing procedure
If the credential public key of the given credential is not of
algorithm -7 ("ES256"), stop and return an error.

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

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

Generate a **signature as specified in [FIDO-U2F-Message-Formats]
section 4.3, with the application parameter set to the SHA-256
hash of the RP ID associated with the given credential, the
challenge parameter set to tbsHash, and the key handle parameter
set to the credential ID of the given credential. Set this as
sig and set the attestation certificate of the attestation
public key as x5c.**

concatenation of authenticatorData and clientDataHash.
+ Verify that the attestation certificate is issued to the
hostname "attest.android.com" (see SafetyNet online
documentation).
+ Verify that the ctsProfileMatch attribute in the payload of
response is true.
+ If successful, return attestation type Basic with the
**attestation trust path set to the above attestation
certificate.**

8.6. FIDO U2F Attestation Statement Format

This attestation statement format is used with FIDO U2F authenticators
using the formats defined in [FIDO-U2F-Message-Formats].

Attestation statement format identifier
fido-u2f

Attestation types supported
Basic **Attestation, Self Attestation, Privacy CA**

Syntax
The syntax of a FIDO U2F attestation statement is defined as
follows:

```

$$attStmtType //= (
    fmt: "fido-u2f",
    attStmt: u2fStmtFormat
)

u2fStmtFormat = {
    x5c: [ attestnCert: bytes, * (caCert: bytes) ],
    sig: bytes
}

```

The semantics of the above fields are as follows:

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

sig
The attestation signature. **The signature was calculated
over the (raw) U2F registration response message
[FIDO-U2F-Message-Formats] received by the platform from
the authenticator.**

Signing procedure
If the credential public key of the given credential is not of
algorithm -7 ("ES256"), stop and return an error. **Otherwise, let
authenticatorData denote the authenticator data for the
attestation, and let clientDataHash denote the hash of the
serialized client data.**

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

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

3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068

Verification procedure
Verification is performed as follows:

- + Verify that the given attestation statement is valid CBOR conforming to the syntax defined above.
- + If x5c is not a certificate for an ECDSA public key over the P-256 curve, stop verification and return an error.
- + Let authenticatorData denote the authenticator data claimed to have been used for the attestation, and let clientDataHash denote the hash of the serialized client data.
- + If clientDataHash is 256 bits long, set tbsHash to this value.

Otherwise set tbsHash to the SHA-256 hash of clientDataHash.

- + From authenticatorData, extract the claimed RP ID hash, the claimed credential ID and the claimed credential public key.
- + Generate the claimed to-be-signed data as specified in [FIDO-U2F-Message-Formats] section 4.3, with the application parameter set to the claimed RP ID hash, the challenge parameter set to tbsHash, the key handle parameter set to the claimed credential ID of the given credential, and the user public key parameter set to the claimed credential public key.
- + Verify that the sig is a valid ECDSA P-256 signature over the to-be-signed data constructed above.
- + If successful, return attestation type Basic with the trust path set to x5c.

3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094

8. WebAuthn Extensions

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

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

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

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

3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3870

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

1. Verify that attStmt is valid CBOR conforming to the syntax defined above, and perform CBOR decoding on it to extract the contained fields.
2. Let attCert be value of the first element of x5c. Let certificate public key be the public key conveyed by attCert. If certificate public key is not an Elliptic Curve (EC) public key over the P-256 curve, terminate this algorithm and return an appropriate error.
3. Extract the claimed rpIdHash from authenticatorData, and the claimed credentialId and credentialPublicKey from authenticatorData.attestedCredentialData.
4. If clientDataHash is 256 bits long, set tbsHash to this value. Otherwise set tbsHash to the SHA-256 hash of clientDataHash.
5. Convert the COSE_KEY formatted credentialPublicKey (see Section 7 of [RFC8152]) to CTAP1/U2F public Key format [FIDO-CTAP].
 - o Let publicKeyU2F represent the result of the conversion operation and set its first byte to 0x04. Note: This signifies uncompressed ECC key format.
 - o Extract the value corresponding to the "-2" key (representing x coordinate) from credentialPublicKey, confirm its size to be of 32 bytes and concatenate it with publicKeyU2F. If size differs or "-2" key is not found, terminate this algorithm and return an appropriate error.
 - o Extract the value corresponding to the "-3" key (representing y coordinate) from credentialPublicKey, confirm its size to be of 32 bytes and concatenate it with publicKeyU2F. If size differs or "-3" key is not found, terminate this algorithm and return an appropriate error.
6. Let verificationData be the concatenation of (0x00 || rpIdHash || tbsHash || credentialId || publicKeyU2F) (see Section 4.3 of [FIDO-U2F-Message-Formats]).
7. Verify the sig using verificationData and certificate public key per [SEC1].
8. If successful, return attestation type Basic with the attestation trust path set to x5c.

9. WebAuthn Extensions

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

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

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

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

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

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

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

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

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

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

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

8.1. 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,

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

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

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

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

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

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

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

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

9 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.

8.2. 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.

8.3. Extending request parameters

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

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

```
var assertionPromise = navigator.credentials.get({
  publicKey: {
    challenge: "...",
    extensions: {
      "webauthnExample_foobar": 42
    }
  }
});
```

Extension definitions MUST specify the valid values for their client extension input. Clients SHOULD ignore extensions with an invalid

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

Extension definitions MUST specify the valid values for their client extension input. Clients SHOULD ignore extensions with an invalid

client extension input. If an extension does not require any parameters from the Relying Party, it SHOULD be defined as taking a Boolean client argument, set to true to signify that the extension is requested by the Relying Party.

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

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

8.4. Client extension processing

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

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

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

8.5. Authenticator extension processing

As specified in 5.1 Authenticator data, the CBOR authenticator extension input value of each processed authenticator extension is included in the extensions data part of the authenticator data. This part is a CBOR map, with CBOR extension identifier values as keys, and the CBOR authenticator extension input value of each extension as the value.

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

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

8.6. Example Extension

This section is not normative.

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

The extension identifier is chosen as webauthnExample_geo. The client extension input is the constant value true, since the extension does not require the Relying Party to pass any particular information to the

client 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 an input to this client processing. Supported client extensions are recorded as a dictionary in the client data with the key clientExtensions. For each such extension, the client adds an entry to this dictionary with the extension identifier as the key, and the extension's client extension input as the value.

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

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

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.

9.6. Example Extension

This section is not normative.

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

The extension identifier is chosen as webauthnExample_geo. The client extension input is the constant value true, since the extension does not require the Relying Party to pass any particular information to the


```
3302 client, other than that it requests the use of the extension. The
3303 Relying Party sets this value in its request for an assertion:
3304 var assertionPromise =
3305   navigator.credentials.get({
3306     publicKey: {
3307       challenge: "SGFuFNvbG8gc2hvdCBmaXJzdC4",

3308       allowCredentials: [], /* Empty filter */
3309       extensions: { 'webauthnExample_geo': true }
3310     }
3311   });
3312
3313 The extension also requires the client to set the authenticator
3314 parameter to the fixed value true.
3315
3316 The extension requires the authenticator to specify its geolocation in
3317 the authenticator extension output, if known. The extension e.g.
3318 specifies that the location shall be encoded as a two-element array of
3319 floating point numbers, encoded with CBOR. An authenticator does this
3320 by including it in the authenticator data. As an example, authenticator
3321 data may be as follows (notation taken from [RFC7049]):
3322 81 (hex)          -- Flags, ED and UP both set.
3323 20 05 58 1F       -- Signature counter
3324 A1               -- CBOR map of one element
3325   73             -- Key 1: CBOR text string of 19 byt
3326 es
3327   77 65 62 61 75 74 68 6E 45 78 61
3328   6D 70 6C 65 5F 67 65 6F       -- "webauthnExample_geo" [=UTF-8 enc
3329 oded=] string
3330   82               -- Value 1: CBOR array of two elemen
3331 ts
3332   FA 42 82 1E B3       -- Element 1: Latitude as CBOR encod
3333 ed float
3334   FA C1 5F E3 7F       -- Element 2: Longitude as CBOR enco
3335 ded float
3336
3337 The extension defines the client extension output to be the geolocation
3338 information, if known, as a GeoJSON [GeoJSON] point. The client
3339 constructs the following client data:
3340 {
3341   'extensions': {
3342     'webauthnExample_geo': {
3343       'type': 'Point',
3344       'coordinates': [65.059962, -13.993041]
3345     }
3346   }
3347 }
3348
3349
3350 9. Defined Extensions
3351
3352 This section defines the initial set of extensions to be registered in
3353 the IANA "WebAuthn Extension Identifier" registry established by
3354 [WebAuthn-Registries]. These are recommended for implementation by user
3355 agents targeting broad interoperability.
3356
3357 9.1. FIDO AppId Extension (appid)
3358
3359 This authentication extension allows Relying Parties that have
3360 previously registered a credential using the legacy FIDO JavaScript
3361 APIs to request an assertion. Specifically, this extension allows
3362 Relying Parties to specify an appid [FIDO-APPID] to overwrite the
3363 otherwise computed rpId. This extension is only valid if used during
3364 the get() call; other usage will result in client error.
3365
3366 Extension identifier
3367 appid
3368
```

```
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   'extensions': {
4123     'webauthnExample_geo': {
4124       'type': 'Point',
4125       'coordinates': [65.059962, -13.993041]
4126     }
4127   }
4128 }
4129
4130
4131 10. Defined Extensions
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
4138 10.1. FIDO AppId Extension (appid)
4139
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
```


3365 Client extension input
3370 A single JSON string specifying a FIDO appld.
3371
3372 Client extension processing
3373 If rpld is present, reject promise with a DOMException whose
3374 name is "NotAllowedError", and terminate this algorithm. Replace
3375 the calculation of rpld in Step 3 of 4.1.4 Use an existing
3376 credential to make an assertion - PublicKeyCredential's
3377 [[DiscoverFromExternalSource]](options) method with the
3378 following procedure: The client uses the value of appld to
3379 perform the Appld validation procedure (as defined by
3380 [FIDO-APPID]). If valid, the value of rpld for all client
3381 processing should be replaced by the value of appld.

3382
3383 Client extension output
3384 Returns the JSON value true to indicate to the RP that the
3385 extension was acted upon
3386

3387 Authenticator extension input
3388 None.
3389

3390 Authenticator extension processing
3391 None.
3392

3393 Authenticator extension output
3394 None.
3395

3396 9.2. Simple Transaction Authorization Extension (txAuthSimple)

3397 This registration extension and authentication extension allows for a
3398 simple form of transaction authorization. A Relying Party can specify a
3399 prompt string, intended for display on a trusted device on the
3400 authenticator.
3401

3402 Extension identifier
3403 txAuthSimple
3404

3405 Client extension input
3406 A single JSON string prompt.
3407

3408 Client extension processing
3409 None, except creating the authenticator extension input from the
3410 client extension input.
3411

3412 Client extension output
3413 Returns the authenticator extension output string UTF-8 decoded
3414 into a JSON string
3415

3416 Authenticator extension input
3417 The client extension input encoded as a CBOR text string (major
3418 type 3).
3419

3420 Authenticator extension processing
3421 The authenticator MUST display the prompt to the user before
3422 performing either user verification or test of user presence.
3423 The authenticator may insert line breaks if needed.
3424

3425 Authenticator extension output
3426 A single CBOR string, representing the prompt as displayed
3427 (including any eventual line breaks).
3428

3429 9.3. Generic Transaction Authorization Extension (txAuthGeneric)

3430 This registration extension and authentication extension allows images
3431 to be used as transaction authorization prompts as well. This allows
3432 authenticators without a font rendering engine to be used and also
3433 supports a richer visual appearance.
3434

4150 Client extension input
4151 A single JSON string specifying a FIDO appld.
4152

4153 Client extension processing
4154 If rpld 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 rpld 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 appld to perform the
4163 Appld validation procedure (as defined by [FIDO-APPID]). If
4164 valid, the value of rpld for all client processing should be
4165 replaced by the value of appld.
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 This registration extension and authentication extension allows for a
4182 simple form of transaction authorization. A Relying Party can specify a
4183 prompt string, intended for display on a trusted device on the
4184 authenticator.
4185

4186 Extension identifier
4187 txAuthSimple
4188

4189 Client extension input
4190 A single JSON string prompt.
4191

4192 Client extension processing
4193 None, except creating the authenticator extension input from the
4194 client extension input.
4195

4196 Client extension output
4197 Returns the authenticator extension output string UTF-8 decoded
4198 into a JSON string
4199

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

4204 Authenticator extension processing
4205 The authenticator MUST display the prompt to the user before
4206 performing either user verification or test of user presence.
4207 The authenticator may insert line breaks if needed.
4208

4209 Authenticator extension output
4210 A single CBOR string, representing the prompt as displayed
4211 (including any eventual line breaks).
4212

4213 10.3. Generic Transaction Authorization Extension (txAuthGeneric)

4214 This registration extension and authentication extension allows images
4215 to be used as transaction authorization prompts as well. This allows
4216 authenticators without a font rendering engine to be used and also
4217 supports a richer visual appearance.
4218

3436
3437 Extension identifier
3438 txAuthGeneric
3439
3440 Client extension input
3441 A CBOR map defined as follows:
3442
3443 txAuthGenericArg = {
3444 contentType: text, ; MIME-Type of the content, e.g.
3445 "image/png"
3446 content: bytes
3447 }
3448
3449 Client extension processing
3450 None, except creating the authenticator extension input from the
3451 client extension input.
3452
3453 Client extension output
3454 Returns the base64url encoding of the authenticator extension
3455 output value as a JSON string
3456
3457 Authenticator extension input
3458 The client extension input encoded as a CBOR map.
3459
3460 Authenticator extension processing
3461 The authenticator MUST display the content to the user before
3462 performing either user verification or test of user presence.
3463 The authenticator may add other information below the content.
3464 No changes are allowed to the content itself, i.e., inside
3465 content boundary box.
3466
3467 Authenticator extension output
3468 The hash value of the content which was displayed. The
3469 authenticator MUST use the same hash algorithm as it uses for
3470 the signature itself.
3471
3472 9.4. Authenticator Selection Extension (authnSel)
3473
3474 This registration extension allows a Relying Party to guide the
3475 selection of the authenticator that will be leveraged when creating the
3476 credential. It is intended primarily for Relying Parties that wish to
3477 tightly control the experience around credential creation.
3478
3479 Extension identifier
3480 authnSel
3481
3482 Client extension input
3483 A sequence of AAGUIDs:
3484
3485 typedef sequence<AAGUID> AuthenticatorSelectionList;
3486
3487 Each AAGUID corresponds to an authenticator model that is
3488 acceptable to the Relying Party for this credential creation.
3489 The list is ordered by decreasing preference.
3490
3491 An AAGUID is defined as an array containing the globally unique
3492 identifier of the authenticator model being sought.
3493
3494 typedef BufferSource AAGUID;
3495
3496 Client extension processing
3497 This extension can only be used during create(). If the client
3498 supports the Authenticator Selection Extension, it MUST use the
3499 first available authenticator whose AAGUID is present in the
3500 AuthenticatorSelectionList. If none of the available
3501 authenticators match a provided AAGUID, the client MUST select
3502 an authenticator from among the available authenticators to
3503 generate the credential.
3504
3505 Client extension output

4220
4221 Extension identifier
4222 txAuthGeneric
4223
4224 Client extension input
4225 A CBOR map defined as follows:
4226
4227 txAuthGenericArg = {
4228 contentType: text, ; MIME-Type of the content, e.g.
4229 "image/png"
4230 content: bytes
4231 }
4232
4233 Client extension processing
4234 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

3506 Returns the JSON value true to indicate to the RP that the
3507 extension was acted upon
3508
3509 Authenticator extension input
3510 None.
3511
3512 Authenticator extension processing
3513 None.
3514
3515 Authenticator extension output
3516 None.
3517
3518 9.5. Supported Extensions Extension (exts)
3519
3520 This registration extension enables the Relying Party to determine
3521 which extensions the authenticator supports.
3522
3523 Extension identifier
3524 exts
3525
3526 Client extension input
3527 The Boolean value true to indicate that this extension is
3528 requested by the Relying Party.
3529
3530 Client extension processing
3531 None, except creating the authenticator extension input from the
3532 client extension input.
3533
3534 Client extension output
3535 Returns the list of supported extensions as a JSON array of
3536 extension identifier strings
3537
3538 Authenticator extension input
3539 The Boolean value true, encoded in CBOR (major type 7, value
3540 21).
3541
3542 Authenticator extension processing
3543 The authenticator sets the authenticator extension output to be
3544 a list of extensions that the authenticator supports, as defined
3545 below. This extension can be added to attestation objects.
3546
3547 Authenticator extension output
3548 The SupportedExtensions extension is a list (CBOR array) of
3549 extension identifier (UTF-8 encoded strings).
3550
3551 9.6. User Verification Index Extension (uvi)
3552
3553 This registration extension and authentication extension enables use of
3554 a user verification index.
3555
3556 Extension identifier
3557 uvi
3558
3559 Client extension input
3560 The Boolean value true to indicate that this extension is
3561 requested by the Relying Party.
3562
3563 Client extension processing
3564 None, except creating the authenticator extension input from the
3565 client extension input.
3566
3567 Client extension output
3568 Returns a JSON string containing the base64url encoding of the
3569 authenticator extension output
3570
3571 Authenticator extension input
3572 The Boolean value true, encoded in CBOR (major type 7, value
3573 21).
3574
3575 Authenticator extension processing

4290 Returns the JSON value true to indicate to the RP that the
4291 extension was acted upon
4292
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 Client extension processing
4315 None, except creating the authenticator extension input from the
4316 client extension input.
4317
4318 Client extension output
4319 Returns the list of supported extensions as a JSON array of
4320 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 Client extension processing
4348 None, except creating the authenticator extension input from the
4349 client extension input.
4350
4351 Client extension output
4352 Returns a JSON string containing the base64url encoding of the
4353 authenticator extension output
4354
4355 Authenticator extension input
4356 The Boolean value true, encoded in CBOR (major type 7, value
4357 21).
4358
4359 Authenticator extension processing

3576 The authenticator sets the authenticator extension output to be
3577 a user verification index indicating the method used by the user
3578 to authorize the operation, as defined below. This extension can
3579 be added to attestation objects and assertions.
3580
3581 Authenticator extension output
3582 The user verification index (UVI) is a value uniquely
3583 identifying a user verification data record. The UVI is encoded
3584 as CBOR byte string (type 0x58). Each UVI value MUST be specific
3585 to the related key (in order to provide unlinkability). It also
3586 must contain sufficient entropy that makes guessing impractical.
3587 UVI values MUST NOT be reused by the Authenticator (for other
3588 biometric data or users).
3589
3590 The UVI data can be used by servers to understand whether an
3591 authentication was authorized by the exact same biometric data
3592 as the initial key generation. This allows the detection and
3593 prevention of "friendly fraud".
3594
3595 As an example, the UVI could be computed as SHA256(KeyID ||
3596 SHA256(rawUVI)), where the rawUVI reflects (a) the biometric
3597 reference data, (b) the related OS level user ID and (c) an
3598 identifier which changes whenever a factory reset is performed
3599 for the device, e.g. rawUVI = biometricReferenceData ||
3600 OSLevelUserID || FactoryResetCounter.
3601
3602 Servers supporting UVI extensions MUST support a length of up to
3603 32 bytes for the UVI value.
3604
3605 Example for authenticator data containing one UVI extension
3606
3607 ... -- [=RP ID=] hash (32 bytes)
3608 81 -- UP and ED set
3609 00 00 00 01 -- (initial) signature counter
3610 ... -- all public key alg etc.
3611 A1 -- extension: CBOR map of one elemen
3612 t
3613 63 -- Key 1: CBOR text string of 3 byte
3614 s
3615 75 76 69 -- "uvi" [=UTF-8 encoded=] string
3616 58 20 -- Value 1: CBOR byte string with 0x
3617 20 bytes
3618 00 43 B8 E3 BE 27 95 8C -- the UVI value itself
3619 28 D5 74 BF 46 8A 85 CF
3620 46 9A 14 F0 E5 16 69 31
3621 DA 4B CF FF C1 BB 11 32
3622 82
3623
3624 9.7. Location Extension (loc)
3625
3626 The location registration extension and authentication extension
3627 provides the client device's current location to the WebAuthn Relying
3628 Party.
3629
3630 Extension identifier
3631 loc
3632
3633 Client extension input
3634 The Boolean value true to indicate that this extension is
3635 requested by the Relying Party.
3636
3637 Client extension processing
3638 None, except creating the authenticator extension input from the
3639 client extension input.
3640
3641 Client extension output
3642 Returns a JSON object that encodes the location information in
3643 the authenticator extension output as a Coordinates value, as
3644 defined by The W3C Geolocation API Specification.

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
4416 loc
4417
4418 Client extension input
4419 The Boolean value true to indicate that this extension is
4420 requested by the Relying Party.
4421
4422 Client extension processing
4423 None, except creating the authenticator extension input from the
4424 client extension input.
4425
4426 Client extension output
4427 Returns a JSON object that encodes the location information in
4428 the authenticator extension output as a Coordinates value, as
4429 defined by The W3C Geolocation API Specification.

3645 Authenticator extension input
3646 The Boolean value true, encoded in CBOR (major type 7, value
3647 21).
3648
3649 Authenticator extension processing
3650 If the authenticator does not support the extension, then the
3651 authenticator MUST ignore the extension request. If the
3652 authenticator accepts the extension, then the authenticator
3653 SHOULD only add this extension data to a packed attestation or
3654 assertion.
3655
3656 Authenticator extension output
3657 If the authenticator accepts the extension request, then
3658 authenticator extension output SHOULD provide location data in
3659 the form of a CBOR-encoded map, with the first value being the
3660 extension identifier and the second being an array of returned
3661 values. The array elements SHOULD be derived from (key,value)
3662 pairings for each location attribute that the authenticator
3663 supports. The following is an example of authenticator data
3664 where the returned array is comprised of a {longitude, latitude,
3665 altitude} triplet, following the coordinate representation
3666 defined in The W3C Geolocation API Specification.
3667
3668 ... -- [=RP ID=] hash (32 bytes)
3669 81 -- UP and ED set
3670 00 00 00 01 -- (initial) signature counter
3671 ... -- all public key alg etc.
3672 A1 -- extension: CBOR map of one elemen
3673 t
3674 63 -- Value 1: CBOR text string of 3 by
3675 tes
3676 6C 6F 63 -- "loc" [=UTF-8 encoded=] string
3677 86 -- Value 2: array of 6 elements
3678 68 -- Element 1: CBOR text string of 8 bytes
3679 6C 61 74 69 74 75 64 65 -- "latitude" [=UTF-8 encoded=] stri
3680 ng
3681 FB ... -- Element 2: Latitude as CBOR encoded double-p
3682 recision float
3683 69 -- Element 3: CBOR text string of 9 bytes
3684 6C 6F 6E 67 69 74 75 64 65 -- "longitude" [=UTF-8 encoded=] str
3685 ing
3686 FB ... -- Element 4: Longitude as CBOR encoded double-
3687 precision float
3688 68 -- Element 5: CBOR text string of 8 bytes
3689 61 6C 74 69 74 75 64 65 -- "altitude" [=UTF-8 encoded=] stri
3690 ng
3691 FB ... -- Element 6: Altitude as CBOR encoded double-p
3692 recision float
3693
3694 9.8. User Verification Method Extension (uvm)
3695
3696 This registration extension and authentication extension enables use of
3697 a user verification method.
3698
3699 Extension identifier
3700 uvm
3701
3702 Client extension input
3703 The Boolean value true to indicate that this extension is
3704 requested by the WebAuthn Relying Party.
3705
3706 Client extension processing
3707 None, except creating the authenticator extension input from the
3708 client extension input.
3709
3710 Client extension output
3711 Returns a JSON array of 3-element arrays of numbers that encodes
3712 the factors in the authenticator extension output
3713
3714

4430 Authenticator extension input
4431 The Boolean value true, encoded in CBOR (major type 7, value
4432 21).
4433
4434 Authenticator extension processing
4435 If the authenticator does not support the extension, then the
4436 authenticator MUST ignore the extension request. If the
4437 authenticator accepts the extension, then the authenticator
4438 SHOULD only add this extension data to a packed attestation or
4439 assertion.
4440
4441 Authenticator extension output
4442 If the authenticator accepts the extension request, then
4443 authenticator extension output SHOULD provide location data in
4444 the form of a CBOR-encoded map, with the first value being the
4445 extension identifier and the second being an array of returned
4446 values. The array elements SHOULD be derived from (key,value)
4447 pairings for each location attribute that the authenticator
4448 supports. The following is an example of authenticator data
4449 where the returned array is comprised of a {longitude, latitude,
4450 altitude} triplet, following the coordinate representation
4451 defined in The W3C Geolocation API Specification.
4452
4453 ... -- [=RP ID=] hash (32 bytes)
4454 81 -- UP and ED set
4455 00 00 00 01 -- (initial) signature counter
4456 ... -- all public key alg etc.
4457 A1 -- extension: CBOR map of one elemen
4458 t
4459 63 -- Value 1: CBOR text string of 3 by
4460 tes
4461 6C 6F 63 -- "loc" [=UTF-8 encoded=] string
4462 86 -- Value 2: array of 6 elements
4463 68 -- Element 1: CBOR text string of 8 bytes
4464 6C 61 74 69 74 75 64 65 -- "latitude" [=UTF-8 encoded=] stri
4465 ng
4466 FB ... -- Element 2: Latitude as CBOR encoded double-p
4467 recision float
4468 69 -- Element 3: CBOR text string of 9 bytes
4469 6C 6F 6E 67 69 74 75 64 65 -- "longitude" [=UTF-8 encoded=] str
4470 ing
4471 FB ... -- Element 4: Longitude as CBOR encoded double-
4472 precision float
4473 68 -- Element 5: CBOR text string of 8 bytes
4474 61 6C 74 69 74 75 64 65 -- "altitude" [=UTF-8 encoded=] stri
4475 ng
4476 FB ... -- Element 6: Altitude as CBOR encoded double-p
4477 recision float
4478
4479 10.8. User Verification Method Extension (uvm)
4480
4481 This registration extension and authentication extension enables use of
4482 a user verification method.
4483
4484 Extension identifier
4485 uvm
4486
4487 Client extension input
4488 The Boolean value true to indicate that this extension is
4489 requested by the WebAuthn Relying Party.
4490
4491 Client extension processing
4492 None, except creating the authenticator extension input from the
4493 client extension input.
4494
4495 Client extension output
4496 Returns a JSON array of 3-element arrays of numbers that encodes
4497 the factors in the authenticator extension output
4498
4499

3715 Authenticator extension input
3716 The Boolean value true, encoded in CBOR (major type 7, value
3717 21).
3718
3719 Authenticator extension processing
3720 The authenticator sets the authenticator extension output to be
3721 a user verification index indicating the method used by the user
3722 to authorize the operation, as defined below. This extension can
3723 be added to attestation objects and assertions.
3724
3725 Authenticator extension output
3726 Authenticators can report up to 3 different user verification
3727 methods (factors) used in a single authentication instance,
3728 using the CBOR syntax defined below:
3729
3730 uvmFormat = [1*3 uvmEntry]
3731 uvmEntry = [
3732 userVerificationMethod: uint .size 4,
3733 keyProtectionType: uint .size 2,
3734 matcherProtectionType: uint .size 2
3735]
3736
3737 The semantics of the fields in each uvmEntry are as follows:
3738
3739 userVerificationMethod
3740 The authentication method/factor used by the authenticator
3741 to verify the user. Available values are defined in
3742 [FIDOReg], "User Verification Methods" section.
3743
3744 keyProtectionType
3745 The method used by the authenticator to protect the FIDO
3746 registration private key material. Available values are
3747 defined in [FIDOReg], "Key Protection Types" section.
3748
3749 matcherProtectionType
3750 The method used by the authenticator to protect the
3751 matcher that performs user verification. Available values
3752 are defined in [FIDOReg], "Matcher Protection Types"
3753 section.
3754
3755 If >3 factors can be used in an authentication instance the
3756 authenticator vendor must select the 3 factors it believes will
3757 be most relevant to the Server to include in the UVM.
3758
3759 Example for authenticator data containing one UVM extension for
3760 a multi-factor authentication instance where 2 factors were
3761 used:
3762
3763 ... -- [=RP ID=] hash (32 bytes)
3764 81 -- UP and ED set
3765 00 00 00 01 -- (initial) signature counter
3766 ... -- all public key alg etc.
3767 A1 -- extension: CBOR map of one element
3768 63 -- Key 1: CBOR text string of 3 bytes
3769 75 76 6d -- "uvm" [=UTF-8 encoded=] string
3770 82 -- Value 1: CBOR array of length 2 indicating two factor
3771 usage
3772 83 -- Item 1: CBOR array of length 3
3773 02 -- Subitem 1: CBOR integer for User Verification Method
3774 Fingerprint
3775 04 -- Subitem 2: CBOR short for Key Protection Type TEE
3776 02 -- Subitem 3: CBOR short for Matcher Protection Type TE
3777 E
3778 83 -- Item 2: CBOR array of length 3
3779 04 -- Subitem 1: CBOR integer for User Verification Method
3780 Passcode
3781 01 -- Subitem 2: CBOR short for Key Protection Type Softwa
3782 re
3783 01 -- Subitem 3: CBOR short for Matcher Protection Type So

4500 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
4523 The semantics of the fields in each uvmEntry are as follows:
4524
4525 userVerificationMethod
4526 The authentication method/factor used by the authenticator
4527 to verify the user. Available values are defined in
4528 [FIDOReg], "User Verification Methods" section.
4529
4530 keyProtectionType
4531 The method used by the authenticator to protect the FIDO
4532 registration private key material. Available values are
4533 defined in [FIDOReg], "Key Protection Types" section.
4534
4535 matcherProtectionType
4536 The method used by the authenticator to protect the
4537 matcher that performs user verification. Available values
4538 are defined in [FIDOReg], "Matcher Protection Types"
4539 section.
4540
4541 If >3 factors can be used in an authentication instance the
4542 authenticator vendor must select the 3 factors it believes will
4543 be most relevant to the Server to include in the UVM.
4544
4545 Example for authenticator data containing one UVM extension for
4546 a multi-factor authentication instance where 2 factors were
4547 used:
4548
4549 ... -- [=RP ID=] hash (32 bytes)
4550 81 -- UP and ED set
4551 00 00 00 01 -- (initial) signature counter
4552 ... -- all public key alg etc.
4553 A1 -- extension: CBOR map of one element
4554 63 -- Key 1: CBOR text string of 3 bytes
4555 75 76 6d -- "uvm" [=UTF-8 encoded=] string
4556 82 -- Value 1: CBOR array of length 2 indicating two factor
4557 usage
4558 83 -- Item 1: CBOR array of length 3
4559 02 -- Subitem 1: CBOR integer for User Verification Method
4560 Fingerprint
4561 04 -- Subitem 2: CBOR short for Key Protection Type TEE
4562 02 -- Subitem 3: CBOR short for Matcher Protection Type TE
4563 E
4564 83 -- Item 2: CBOR array of length 3
4565 04 -- Subitem 1: CBOR integer for User Verification Method
4566 Passcode
4567 01 -- Subitem 2: CBOR short for Key Protection Type Softwa
4568 re
4569 01 -- Subitem 3: CBOR short for Matcher Protection Type So

ftware

10. IANA Considerations

10.1. WebAuthn Attestation Statement Format Identifier Registrations

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

- * WebAuthn Attestation Statement Format Identifier: packed
- * Description: The "packed" attestation statement format is a WebAuthn-optimized format for attestation data. It uses a very compact but still extensible encoding method. This format is implementable by authenticators with limited resources (e.g., secure elements).
- * Specification Document: Section 7.2 Packed Attestation Statement Format of this specification
- * WebAuthn Attestation Statement Format Identifier: tpm
- * Description: The TPM attestation statement format returns an attestation statement in the same format as the packed attestation statement format, although the the rawData and signature fields are computed differently.
- * Specification Document: Section 7.3 TPM Attestation Statement Format of this specification
- * WebAuthn Attestation Statement Format Identifier: android-key
- * Description: Platform-provided authenticators based on Android versions "N", and later, may provide this proprietary "hardware attestation" statement.
- * Specification Document: Section 7.4 Android Key Attestation Statement Format of this specification
- * WebAuthn Attestation Statement Format Identifier: android-safetynet
- * Description: Android-based, platform-provided authenticators may produce an attestation statement based on the Android SafetyNet API.
- * Specification Document: Section 7.5 Android SafetyNet Attestation Statement Format of this specification
- * WebAuthn Attestation Statement Format Identifier: fido-u2f
- * Description: Used with FIDO U2F authenticators
- * Specification Document: Section 7.6 FIDO U2F Attestation Statement Format of this specification

10.2. WebAuthn Extension Identifier Registrations

This section registers the extension identifier values defined in Section 8 WebAuthn Extensions in the IANA "WebAuthn Extension Identifier" registry established by [WebAuthn-Registries].

- * WebAuthn Extension Identifier: appid
- * Description: This authentication extension allows Relying Parties that have previously registered a credential using the legacy FIDO JavaScript APIs to request an assertion.
- * Specification Document: Section 9.1 FIDO AppId Extension (appid) of this specification
- * WebAuthn Extension Identifier: txAuthSimple
- * Description: This registration extension and authentication extension allows for a simple form of transaction authorization. A WebAuthn Relying Party can specify a prompt string, intended for display on a trusted device on the authenticator
- * Specification Document: Section 9.2 Simple Transaction Authorization Extension (txAuthSimple) of this specification
- * WebAuthn Extension Identifier: txAuthGeneric
- * Description: 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 than accomplished with the webauthn.txauth.simple extension.
- * Specification Document: Section 9.3 Generic Transaction Authorization Extension (txAuthGeneric) of this specification
- * WebAuthn Extension Identifier: authnSel

ftware

11. IANA Considerations

11.1. WebAuthn Attestation Statement Format Identifier Registrations

This section registers the attestation statement formats defined in Section 8 Defined Attestation Statement Formats in the IANA "WebAuthn Attestation Statement Format Identifier" registry established by [WebAuthn-Registries].

- * WebAuthn Attestation Statement Format Identifier: packed
- * Description: The "packed" attestation statement format is a WebAuthn-optimized format for attestation. It uses a very compact but still extensible encoding method. This format is implementable by authenticators with limited resources (e.g., secure elements).
- * Specification Document: Section 8.2 Packed Attestation Statement Format of this specification
- * WebAuthn Attestation Statement Format Identifier: tpm
- * Description: The TPM attestation statement format returns an attestation statement in the same format as the packed attestation statement format, although the the rawData and signature fields are computed differently.
- * Specification Document: Section 8.3 TPM Attestation Statement Format of this specification
- * WebAuthn Attestation Statement Format Identifier: android-key
- * Description: Platform-provided authenticators based on versions "N", and later, may provide this proprietary "hardware attestation" statement.
- * Specification Document: Section 8.4 Android Key Attestation Statement Format of this specification
- * WebAuthn Attestation Statement Format Identifier: android-safetynet
- * Description: Android-based, platform-provided authenticators may produce an attestation statement based on the Android SafetyNet API.
- * Specification Document: Section 8.5 Android SafetyNet Attestation Statement Format of this specification
- * WebAuthn Attestation Statement Format Identifier: fido-u2f
- * Description: Used with FIDO U2F authenticators
- * Specification Document: Section 8.6 FIDO U2F Attestation Statement Format of this specification

11.2. WebAuthn Extension Identifier Registrations

This section registers the extension identifier values defined in Section 9 WebAuthn Extensions in the IANA "WebAuthn Extension Identifier" registry established by [WebAuthn-Registries].

- * WebAuthn Extension Identifier: appid
- * Description: This authentication extension allows Relying Parties that have previously registered a credential using the legacy FIDO JavaScript APIs to request an assertion.
- * Specification Document: Section 10.1 FIDO AppId Extension (appid) of this specification
- * WebAuthn Extension Identifier: txAuthSimple
- * Description: This registration extension and authentication extension allows for a simple form of transaction authorization. A WebAuthn Relying Party can specify a prompt string, intended for display on a trusted device on the authenticator
- * Specification Document: Section 10.2 Simple Transaction Authorization Extension (txAuthSimple) of this specification
- * WebAuthn Extension Identifier: txAuthGeneric
- * Description: 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 than accomplished with the webauthn.txauth.simple extension.
- * Specification Document: Section 10.3 Generic Transaction Authorization Extension (txAuthGeneric) of this specification
- * WebAuthn Extension Identifier: authnSel

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

* Specification Document: Section 9.4 Authenticator Selection Extension (authnSel) of this specification

* WebAuthn Extension Identifier: exts

* Description: This registration extension enables the Relying Party to determine which extensions the authenticator supports. The extension data is a list (CBOR array) of extension identifiers encoded as UTF-8 Strings. This extension is added automatically by the authenticator. This extension can be added to attestation statements.

* Specification Document: Section 9.5 Supported Extensions Extension (exts) of this specification

* WebAuthn Extension Identifier: uvi

* Description: This registration extension and authentication extension enables use of a user verification index. The user verification index is a value uniquely identifying a user verification data record. The UVI data can be used by servers to understand whether an authentication was authorized by the exact same biometric data as the initial key generation. This allows the detection and prevention of "friendly fraud".

* Specification Document: Section 9.6 User Verification Index Extension (uvi) of this specification

* WebAuthn Extension Identifier: loc

* Description: The location registration extension and authentication extension provides the client device's current location to the WebAuthn relying party, if supported by the client device and subject to user consent.

* Specification Document: Section 9.7 Location Extension (loc) of this specification

* WebAuthn Extension Identifier: uvm

* Description: This registration extension and authentication extension enables use of a user verification method. The user verification method extension returns to the Webauthn relying party which user verification methods (factors) were used for the WebAuthn operation.

* Specification Document: Section 9.8 User Verification Method Extension (uvm) of this specification

10.3. COSE Algorithm Registrations

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

- * Name: RS256
- * Value: -257
- * Description: RSASSA-PKCS1-v1_5 w/ SHA-256
- * Reference: Section 8.2 of [RFC8017]
- * Recommended: No
- * Name: RS384
- * Value: -258
- * Description: RSASSA-PKCS1-v1_5 w/ SHA-384
- * Reference: Section 8.2 of [RFC8017]
- * Recommended: No
- * Name: RS512
- * Value: -259
- * Description: RSASSA-PKCS1-v1_5 w/ SHA-512
- * Reference: Section 8.2 of [RFC8017]
- * Recommended: No

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

* Specification Document: Section 10.4 Authenticator Selection Extension (authnSel) of this specification

* WebAuthn Extension Identifier: exts

* Description: This registration extension enables the Relying Party to determine which extensions the authenticator supports. The extension data is a list (CBOR array) of extension identifiers encoded as UTF-8 Strings. This extension is added automatically by the authenticator. This extension can be added to attestation statements.

* Specification Document: Section 10.5 Supported Extensions Extension (exts) of this specification

* WebAuthn Extension Identifier: uvi

* Description: This registration extension and authentication extension enables use of a user verification index. The user verification index is a value uniquely identifying a user verification data record. The UVI data can be used by servers to understand whether an authentication was authorized by the exact same biometric data as the initial key generation. This allows the detection and prevention of "friendly fraud".

* Specification Document: Section 10.6 User Verification Index Extension (uvi) of this specification

* WebAuthn Extension Identifier: loc

* Description: The location registration extension and authentication extension provides the client device's current location to the WebAuthn relying party, if supported by the client device and subject to user consent.

* Specification Document: Section 10.7 Location Extension (loc) of this specification

* WebAuthn Extension Identifier: uvm

* Description: This registration extension and authentication extension enables use of a user verification method. The user verification method extension returns to the Webauthn relying party which user verification methods (factors) were used for the WebAuthn operation.

* Specification Document: Section 10.8 User Verification Method Extension (uvm) of this specification

11.3. COSE Algorithm Registrations

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

- * Name: RS256
- * Value: -257
- * Description: RSASSA-PKCS1-v1_5 w/ SHA-256
- * Reference: Section 8.2 of [RFC8017]
- * Recommended: No
- * Name: RS384
- * Value: -258
- * Description: RSASSA-PKCS1-v1_5 w/ SHA-384
- * Reference: Section 8.2 of [RFC8017]
- * Recommended: No
- * Name: RS512
- * Value: -259
- * Description: RSASSA-PKCS1-v1_5 w/ SHA-512
- * Reference: Section 8.2 of [RFC8017]
- * Recommended: No
- * Name: ED256
- * Value: -260
- * Description: TPM_ECC_BN_P256 curve w/ SHA-256
- * Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
- * Recommended: Yes
- * Name: ED512
- * Value: -261

3916
3917
3918
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
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

11. 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.

11.1. 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.
 - + The script may store data such as the credential ID in local storage, to improve future UX by narrowing the choice of credential for the user.

The sample code for generating and registering a new key follows:

```
if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }  
  
var publicKey = {
```

4709
4710
4711
4712
4713
4714
4715
4716
4717
4718
4719
4720
4721
4722
4723
4724
4725
4726
4727
4728
4729
4730
4731
4732
4733
4734
4735
4736
4737
4738
4739
4740
4741
4742
4743
4744
4745
4746
4747
4748
4749
4750
4751
4752
4753
4754
4755
4756
4757
4758
4759
4760
4761
4762
4763
4764
4765
4766
4767
4768
4769
4770
4771
4772
4773
4774
4775
4776
4777
4778

* Description: ECC_BN_ISOP512 curve w/ SHA-512
* Reference: Section 4.2 of [FIDOEcdaaAlgorithm]
* Recommended: Yes
* Name: RS1
* Value: -262
* Description: RSASSA-PKCS1-v1_5 w/ SHA-1
* Reference: Section 8.2 of [RFC8017]
* 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.
 - + The script may store data such as the credential ID in local storage, to improve future UX by narrowing the choice of credential for the user.

The sample code for generating and registering a new key follows:

```
if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }  
  
var publicKey = {
```

```

3978 challenge: Uint8Array.from(window.atob("PGifxAoBwCkWkm4b1Cill5otCphilh6MijdbW
3979 FjomA="), c=>c.charCodeAt(0)),

3980
3981 // Relying Party:
3982 rp: {
3983   name: "Acme"
3984 },
3985
3986 // User:
3987 user: {
3988   id: "1098237235409872"

3989   name: "john.p.smith@example.com",
3990   displayName: "John P. Smith",
3991   icon: "https://pics.acme.com/00/p/aBjjjpqPb.png"
3992 },
3993
3994 // This Relying Party will accept either an ES256 or RS256 credential, but
3995 // prefers an ES256 credential.
3996 pubKeyCredParams: [
3997   {
3998     type: "public-key",
3999     alg: -7 // "ES256" as registered in the IANA COSE Algorithms registry
4000   },
4001   {
4002     type: "public-key",
4003     alg: -257 // Value registered by this specification for "RS256"
4004   }
4005 ],
4006
4007 timeout: 60000, // 1 minute
4008 excludeCredentials: [], // No exclude list of PKCredDescriptors
4009 extensions: {"webauthn.location": true} // Include location information
4010 // in attestation
4011 };
4012
4013 // Note: The following call will cause the authenticator to display UI.
4014 navigator.credentials.create({ publicKey })
4015 .then(function (newCredentialInfo) {
4016   // Send new credential info to server for verification and registration.
4017 })
4018 .catch(function (err) {
4019   // No acceptable authenticator or user refused consent. Handle appropriately
4020 });
4021
4022 11.2. Registration Specifically with Platform Authenticator
4023
4024 This is flow for when the Relying Party is specifically interested in
4025 creating a public key credential with a platform authenticator.

4026 1. The user visits example.com and clicks on the login button, which
4027   redirects the user to login.example.com.
4028 2. The user enters a username and password to log in. After successful
4029   login, the user is redirected back to example.com.
4030 3. The Relying Party script runs the code snippet below.
4031 4. The user agent asks the user whether they are willing to register
4032   with the Relying Party using an available platform authenticator.
4033 5. If the user is not willing, terminate this flow.
4034 6. The user is shown appropriate UI and guided in creating a
4035   credential using one of the available platform authenticators. Upon
4036   successful credential creation, the RP script conveys the new
4037   credential to the server.
4038 if (!PublicKeyCredential) { /* Platform not capable of the API. Handle error. */
4039 }
4040
4041 PublicKeyCredential.isPlatformAuthenticatorAvailable()
4042 .then(function (userIntent) {
4043

```

```

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("MIIBkzCCATigAwIBAJCCAQMwggE4oAMCAQIwggGTMII
4792 ="), c=>c.charCodeAt(0)),
4793   name: "john.p.smith@example.com",
4794   displayName: "John P. Smith",
4795   icon: "https://pics.acme.com/00/p/aBjjjpqPb.png"
4796 },
4797
4798 // This Relying Party will accept either an ES256 or RS256 credential, but
4799 // prefers an ES256 credential.
4800 pubKeyCredParams: [
4801   {
4802     type: "public-key",
4803     alg: -7 // "ES256" as registered in the IANA COSE Algorithms registry
4804   },
4805   {
4806     type: "public-key",
4807     alg: -257 // Value registered by this specification for "RS256"
4808   }
4809 ],
4810
4811 timeout: 60000, // 1 minute
4812 excludeCredentials: [], // No exclude list of PKCredDescriptors
4813 extensions: {"loc": true} // Include location information
4814 // in attestation
4815 };
4816
4817 // Note: The following call will cause the authenticator to display UI.
4818 navigator.credentials.create({ publicKey })
4819 .then(function (newCredentialInfo) {
4820   // Send new credential info to server for verification and registration.
4821 })
4822 .catch(function (err) {
4823   // No acceptable authenticator or user refused consent. Handle appropriately
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
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

```

```

4044 // If the user has affirmed willingness to register with RP using an ava
4045 ilable platform authenticator
4046 if (userIntent) {
4047     var publicKeyOptions = { /* Public key credential creation options.
4048 */};
4049
4050 // Create and register credentials.
4051 return navigator.credentials.create({ "publicKey": publicKeyOptions
4052 });
4053 } else {
4054
4055 // Record that the user does not intend to use a platform authentica
4056 tor
4057 // and default the user to a password-based flow in the future.
4058 }
4059
4060 }).then(function (newCredentialInfo) {
4061 // Send new credential info to server for verification and registration.
4062 }).catch( function(err) {
4063 // Something went wrong. Handle appropriately.
4064 });
4065

```

11.3. 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.

If the Relying Party script does not have any hints available (e.g., from locally stored data) to help it narrow the list of credentials, then the sample code for performing such an authentication might look like this:

```

4110 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4111
4112 var options = {
4113     challenge: new TextEncoder().encode("climb a mountain"),

```

```

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 });
4870

```

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.

If the Relying Party script does not have any hints available (e.g., from locally stored data) to help it narrow the list of credentials, then the sample code for performing such an authentication might look like this:

```

4910 if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4911
4912 var options = {
4913     // The challenge must be produced by the server, see the Securit

```

```
4114         timeout: 60000, // 1 minute
4115         allowCredentials: [{ type: "public-key" }]
4116     };
4117
4118     navigator.credentials.get({ "publicKey": options })
4119     .then(function (assertion) {
4120         // Send assertion to server for verification
4121     }).catch(function (err) {
4122         // No acceptable credential or user refused consent. Handle appropriately.
4123     });
4124
4125     On the other hand, if the Relying Party script has some hints to help
4126     it narrow the list of credentials, then the sample code for performing
4127     such an authentication might look like the following. Note that this
4128     sample also demonstrates how to use the extension for transaction
4129     authorization.
4130     if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4131
4132     var encoder = new TextEncoder();
4133     var acceptableCredential1 = {
4134         type: "public-key",
4135         id: encoder.encode("!!!!!!hi there!!!!!!\n")
4136     };
4137     var acceptableCredential2 = {
4138         type: "public-key",
4139         id: encoder.encode("roses are red, violets are blue\n")
4140     };
4141
4142     var options = {
4143         challenge: encoder.encode("climb a mountain"),
4144
4145         timeout: 60000, // 1 minute
4146         allowCredentials: [acceptableCredential1, acceptableCredential2]
4147     };
4148     extensions: { 'webauthn.txauth.simple':
4149         "Wave your hands in the air like you just don't care" };
4150
4151     navigator.credentials.get({ "publicKey": options })
4152     .then(function (assertion) {
4153         // Send assertion to server for verification
4154     }).catch(function (err) {
4155         // No acceptable credential or user refused consent. Handle appropriately.
4156     });
4157
4158     11.4. Decommissioning
```

```
4919 y Considerations
4920     challenge: new Uint8Array([4,101,15 /* 29 more random bytes gene
4921 rated by the server */]),
4922     timeout: 60000, // 1 minute
4923     allowCredentials: [{ type: "public-key" }]
4924 };
4925
4926 navigator.credentials.get({ "publicKey": options })
4927 .then(function (assertion) {
4928     // Send assertion to server for verification
4929 }).catch(function (err) {
4930     // No acceptable credential or user refused consent. Handle appropriately.
4931 });
4932
4933     On the other hand, if the Relying Party script has some hints to help
4934     it narrow the list of credentials, then the sample code for performing
4935     such an authentication might look like the following. Note that this
4936     sample also demonstrates how to use the extension for transaction
4937     authorization.
4938     if (!PublicKeyCredential) { /* Platform not capable. Handle error. */ }
4939
4940     var encoder = new TextEncoder();
4941     var acceptableCredential1 = {
4942         type: "public-key",
4943         id: encoder.encode("!!!!!!hi there!!!!!!\n")
4944     };
4945     var acceptableCredential2 = {
4946         type: "public-key",
4947         id: encoder.encode("roses are red, violets are blue\n")
4948     };
4949
4950     var options = {
4951         // The challenge must be produced by the server, see the Securit
4952 y Considerations
4953     challenge: new Uint8Array([8,18,33 /* 29 more random bytes gener
4954 ated by the server */]),
4955     timeout: 60000, // 1 minute
4956     allowCredentials: [acceptableCredential1, acceptableCredential2]
4957 };
4958     extensions: { 'txAuthSimple':
4959         "Wave your hands in the air like you just don't care" }
4960 };
4961
4962     navigator.credentials.get({ "publicKey": options })
4963     .then(function (assertion) {
4964         // Send assertion to server for verification
4965     }).catch(function (err) {
4966         // No acceptable credential or user refused consent. Handle appropriately.
4967     });
4968
4969     12.4. Aborting Authentication Operations
4970
4971     The below example shows how a developer may use the AbortSignal
4972     parameter to abort a credential registration operation. A similiar
4973     procedure applies to an authentication operation.
4974     const authAbortController = new AbortController();
4975     const authAbortSignal = authAbortController.signal;
4976
4977     authAbortSignal.onabort = function () {
4978         // Once the page knows the abort started, inform user it is attempting to ab
4979 ort.
4980     }
4981
4982     var options = {
4983         // A list of options.
4984     }
4985
4986     navigator.credentials.create({
4987         publicKey: options,
4988         signal: authAbortSignal})
```


4159 The following are possible situations in which decommissioning a
4160 credential might be desired. Note that all of these are handled on the
4161 server side and do not need support from the API specified here.
4162 * Possibility #1 -- user reports the credential as lost.
4163 + User goes to server.example.net, authenticates and follows a
4164 link to report a lost/stolen device.
4165 + Server returns a page showing the list of registered
4166 credentials with friendly names as configured during
4167 registration.
4168 + User selects a credential and the server deletes it from its
4169 database.
4170 + In future, the Relying Party script does not specify this
4171 credential in any list of acceptable credentials, and
4172 assertions signed by this credential are rejected.
4173 * Possibility #2 -- server deregisters the credential due to
4174 inactivity.
4175 + Server deletes credential from its database during maintenance
4176 activity.
4177 + In the future, the Relying Party script does not specify this
4178 credential in any list of acceptable credentials, and
4179 assertions signed by this credential are rejected.
4180 * Possibility #3 -- user deletes the credential from the device.
4181 + User employs a device-specific method (e.g., device settings
4182 UI) to delete a credential from their device.
4183 + From this point on, this credential will not appear in any
4184 selection prompts, and no assertions can be generated with it.
4185 + Sometime later, the server deregisters this credential due to
4186 inactivity.
4187
4188
4189

12. Acknowledgements

4190 We thank the following for their contributions to, and thorough review
4191 of, this specification: Richard Barnes, Dominic Battr, Domenic
4192 Denicola, Rahul Ghosh, Brad Hill, Jing Jin, Angelo Liao, Anne van
4193 Kesteren, Ian Kilpatrick, Giridhar Mandyam, Axel Nennker, Kimberly
4194 Paulhamus, Adam Powers, Yaron Sheffer, Mike West, Jeffrey Yasskin,
4195 Boris Zbarsky.
4196
4197

```
4989 .then(function (attestation) {  
4990   // Register the user.  
4991 }).catch(function (error) {  
4992   if (error == "AbortError") {  
4993     // Inform user the credential hasn't been created.  
4994     // Let the server know a key hasn't been created.  
4995   }  
4996 });  
4997  
4998 // Assume widget shows up whenever auth occurs.  
4999 if (widget == "disappear") {  
5000   authAbortSignal.abort();  
5001 }  
5002 }  
5003  
5004
```

12.5. Decommissioning

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

13. Security Considerations

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,
Boris Zbarsky.

4198	Index
4199	
4200	Terms defined by this specification
4201	
4202	* aa, in 4.4.3
4203	* AAGUID, in 9.4
4204	* alg, in 4.3
4205	* allowCredentials, in 4.5
4206	* Assertion, in 3
4207	* assertion signature, in 5
4208	* attachment modality, in 4.4.4
4209	* Attestation, in 3
4210	* Attestation Certificate, in 3
4211	* Attestation data, in 5.3.1
4212	* attestation key pair, in 3
4213	* attestationObject, in 4.2.1
4214	* attestation object, in 5.3
4215	* attestation private key, in 3
4216	* attestation public key, in 3
4217	* attestation signature, in 5
4218	* attestation statement, in 5.3
4219	* attestation statement format, in 5.3
4220	* attestation statement format identifier, in 7.1
4221	* attestation type, in 5.3
4222	* Authentication, in 3
4223	* Authentication Assertion, in 3
4224	* authentication extension, in 8
4225	* AuthenticationExtensions
4226	+ definition of, in 4.6
4227	+ (typedef), in 4.6
4228	* Authenticator, in 3
4229	* AuthenticatorAssertionResponse, in 4.2.2
4230	* AuthenticatorAttachment, in 4.4.4
4231	* AuthenticatorAttestationResponse, in 4.2.1
4232	* authenticatorCancel, in 5.2.3
4233	* authenticator data, in 5.1
4234	* authenticatorData, in 4.2.2
4235	* authenticator data claimed to have been used for the attestation,
4236	in 5.3.2
4237	* authenticator data for the attestation, in 5.3.2
4238	* authenticator extension, in 8
4239	* authenticator extension input, in 8.3
4240	* authenticator extension output, in 8.5
4241	* Authenticator extension processing, in 8.5
4242	* authenticatorExtensions, in 4.7.1
4243	* authenticatorGetAssertion, in 5.2.2
4244	* authenticatorMakeCredential, in 5.2.1
4245	* AuthenticatorResponse, in 4.2
4246	* authenticatorSelection, in 4.4
4247	* AuthenticatorSelectionCriteria, in 4.4.3
4248	* AuthenticatorSelectionList, in 9.4
4249	* AuthenticatorTransport, in 4.7.4
4250	* Authorization Gesture, in 3
4251	* Base64url Encoding, in 2.1
4252	* Basic Attestation, in 5.3.3
4253	* Biometric Recognition, in 3
4254	* ble, in 4.7.4
4255	* CBOR, in 2.1
4256	* Ceremony, in 3

5059	Index
5060	
5061	Terms defined by this specification
5062	
5063	* aaguid, in 6.3.1
5064	* AAGUID, in 10.4
5065	* 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
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

4257 * challenge
4258 + dict-member for MakePublicKeyCredentialOptions, in 4.4
4259 + dict-member for PublicKeyCredentialRequestOptions, in 4.5
4260 + dict-member for CollectedClientData, in 4.7.1
4261 * Client, in 3
4262 * client data, in 4.7.1
4263 * clientDataJSON, in 4.2
4264 * client extension, in 8
4265 * client extension input, in 8.3
4266 * client extension output, in 8.4
4267 * Client extension processing, in 8.4
4268 * clientExtensionResults, in 4.1
4269 * clientExtensions, in 4.7.1
4270 * Client-Side, in 3
4271 * client-side credential private key storage, in 3
4272 * Client-side-resident Credential Private Key, in 3
4273 * CollectedClientData, in 4.7.1
4274 * Conforming User Agent, in 3

4275 * COSEAlgorithmIdentifier
4276 + definition of, in 4.7.5
4277 + (typedef), in 4.7.5
4278 * [[Create]](options), in 4.1.3
4279 * credential key pair, in 3
4280 * credential private key, in 3
4281 * Credential Public Key, in 3
4282 * cross-platform attached, in 4.4.4
4283 * cross-platform attachment, in 4.4.4
4284 * DAA, in 5.3.3
4285 * [[DiscoverFromExternalSource]](options), in 4.1.4
4286 * [[discovery]], in 4.1
4287 * displayName, in 4.4.2
4288 * ECDA, in 5.3.3
4289 * ECDA-Issuer public key, in 7.2
4290 * Elliptic Curve based Direct Anonymous Attestation, in 5.3.3
4291 * excludeCredentials, in 4.4
4292 * extension identifier, in 8.1

4293 * extensions
4294 + dict-member for MakePublicKeyCredentialOptions, in 4.4
4295 + dict-member for PublicKeyCredentialRequestOptions, in 4.5
4296 * hashAlgorithm, in 4.7.1
4297 * Hash of the serialized client data, in 4.7.1
4298 * icon, in 4.4.1

4299 * id
4300 + dict-member for PublicKeyCredentialEntity, in 4.4.1
4301 + dict-member for PublicKeyCredentialDescriptor, in 4.7.3
4302 * [[identifier]], in 4.1

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 * ECDA, in 6.3.3
5180 * ECDA-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
5196 + dict-member for PublicKeyCredentialRpEntity, in 5.4.2
5197 + dict-member for PublicKeyCredentialUserEntity, in 5.4.3
5198 + dict-member for PublicKeyCredentialDescriptor, in 5.8.3

4303 * identifier of the ECDAAs-Issuer public key, in 7.2
4304 * isPlatformAuthenticatorAvailable(), in 4.1.5
4305 * JSON-serialized client data, in 4.7.1
4306 * MakePublicKeyCredentialOptions, in 4.4
4307 * name, in 4.4.1
4308 * nfc, in 4.7.4
4309 * origin, in 4.7.1
4310 * "plat", in 4.4.4
4311 * plat, in 4.4.4
4312 * platform attachment, in 4.4.4
4313 * platform authenticators, in 4.4.4
4314 * Privacy CA, in 5.3.3
4315 * pubKeyCredParams, in 4.4

4316 * publicKey
4317 + dict-member for CredentialCreationOptions, in 4.1.1
4318 + dict-member for CredentialRequestOptions, in 4.1.2
4319 * public-key, in 4.7.2
4320 * Public Key Credential, in 3
4321 * PublicKeyCredential, in 4.1
4322 * PublicKeyCredentialDescriptor, in 4.7.3
4323 * PublicKeyCredentialEntity, in 4.4.1
4324 * PublicKeyCredentialParameters, in 4.3
4325 * PublicKeyCredentialRequestOptions, in 4.5
4326 * PublicKeyCredentialType, in 4.7.2
4327 * PublicKeyCredentialUserEntity, in 4.4.2
4328 * Rate Limiting, in 3
4329 * rawId, in 4.1
4330 * Registration, in 3
4331 * registration extension, in 8
4332 * Relying Party, in 3
4333 * Relying Party Identifier, in 3
4334 * response, in 4.1
4335 * rk, in 4.4.3
4336 * roaming authenticators, in 4.4.4
4337 * rp, in 4.4
4338 * rpId, in 4.5
4339 * RP ID, in 3
4340 * Self Attestation, in 5.3.3
4341 * signature, in 4.2.2
4342 * Signing procedure, in 5.3.2
4343 * Test of User Presence, in 3

4344 * timeout
4345 + dict-member for MakePublicKeyCredentialOptions, in 4.4
4346 + dict-member for PublicKeyCredentialRequestOptions, in 4.5
4347 * tokenBindingId, in 4.7.1
4348 * transports, in 4.7.3
4349 * [[type]], in 4.1
4350 * type
4351 + dict-member for PublicKeyCredentialParameters, in 4.3
4352 + dict-member for PublicKeyCredentialDescriptor, in 4.7.3
4353 * UP, in 3
4354 * usb, in 4.7.4
4355 * user, in 4.4
4356 * User Consent, in 3
4357 * User Present, in 3

5199 * [[identifier]], in 5.1
5200 * identifier of the ECDAAs-Issuer public key, in 8.2
5201 * indirect, in 5.4.6
5202 * isUserVerifyingPlatformAuthenticatorAvailable(), in 5.1.6
5203 * JSON-serialized client data, in 5.8.1
5204 * MakePublicKeyCredentialOptions, in 5.4
5205 * managing authenticator, in 4
5206 * name, in 5.4.1
5207 * nfc, in 5.8.4
5208 * none, in 5.4.6
5209 * origin, in 5.8.1
5210 * platform, in 5.4.5
5211 * "platform", in 5.4.5
5212 * platform attachment, in 5.4.5
5213 * platform authenticators, in 5.4.5
5214 * "preferred", in 5.8.6
5215 * preferred, in 5.8.6
5216 * Privacy CA, in 6.3.3
5217 * pubKeyCredParams, in 5.4
5218 * publicKey
5219 + dict-member for CredentialCreationOptions, in 5.1.1
5220 + dict-member for CredentialRequestOptions, in 5.1.2
5221 * public-key, in 5.8.2
5222 * Public Key Credential, in 4
5223 * PublicKeyCredential, in 5.1
5224 * PublicKeyCredentialDescriptor, in 5.8.3
5225 * PublicKeyCredentialEntity, in 5.4.1
5226 * PublicKeyCredentialParameters, in 5.3
5227 * PublicKeyCredentialRequestOptions, in 5.5
5228 * PublicKeyCredentialRpEntity, in 5.4.2
5229 * Public Key Credential Source, in 4
5230 * PublicKeyCredentialType, in 5.8.2
5231 * PublicKeyCredentialUserEntity, in 5.4.3
5232 * Rate Limiting, in 4
5233 * rawId, in 5.1
5234 * Registration, in 4
5235 * registration extension, in 9
5236 * Relying Party, in 4
5237 * Relying Party Identifier, in 4
5238 * "required", in 5.8.6
5239 * required, in 5.8.6
5240 * requireResidentKey, in 5.4.4
5241 * response, in 5.1
5242 * roaming authenticators, in 5.4.5
5243 * rp, in 5.4
5244 * rpId, in 5.5
5245 * RP ID, in 4
5246 * rpIdHash, in 6.1
5247 * Self Attestation, in 6.3.3
5248 * signature, in 5.2.2
5249 * Signature Counter, in 6.1.1
5250 * signatureResult, in 5.1.4.1
5251 * signCount, in 6.1
5252 * Signing procedure, in 6.3.2
5253 * [[Store]](credential, sameOriginWithAncestors), in 5.1.5
5254 * Test of User Presence, in 4
5255 * timeout
5256 + dict-member for MakePublicKeyCredentialOptions, in 5.4
5257 + dict-member for PublicKeyCredentialRequestOptions, in 5.5
5258 * tokenBindingId, in 5.8.1
5259 * transports, in 5.8.3
5260 * [[type]], in 5.1
5261 * type
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

- * **User Verification**, in 3
- * **User Verified**, in 3
- * **UV**, in 3
- * **uv**, in 4.4.3
- * **Verification procedures**, in 5.3.2
- * **Web Authentication API**, in 4
- * **WebAuthn Client**, in 3
- * **"xplat"**, in 4.4.4
- * **xplat**, in 4.4.4

Terms defined by reference

- * [CREDENTIAL-MANAGEMENT-1] defines the following terms:
 - + **Credential**
 - + **CredentialCreationOptions**
 - + **CredentialRequestOptions**
 - + **CredentialsContainer**
 - + **[[CollectFromCredentialStore]](options)**
 - + **[[Store]](credential)**
- + **[[discovery]]**
- + **[[type]]**
- + **create()**
- + **get()**
- + **id**
- + **remote**
- + **type**
- * [ECMAScript] defines the following terms:
 - + **%arraybuffer%**
 - + **internal slot**
 - + **stringify**
- * [ENCODING] defines the following terms:
 - + **utf-8 encode**
- * [HTML] defines the following terms:
 - + **ascii serialization of an origin**
 - + **dom manipulation task source**
 - + **effective domain**
 - + **global object**
 - + **in parallel**
 - + **is a registrable domain suffix of or is equal to**
 - + **is not a registrable domain suffix of and is not equal to**
 - + **origin**
 - + **promise**
 - + **relevant settings object**
 - + **task**
 - + **task source**
- * [HTML52] defines the following terms:

- * **userHandle**, in 5.2.2
- * **User Handle**, in 4
- * **userHandleResult**, in 5.1.4.1
- * **User Present**, in 4
- * **userVerification**
 - + **dict-member for AuthenticatorSelectionCriteria**, in 5.4.4
 - + **dict-member for PublicKeyCredentialRequestOptions**, in 5.5
- * **User Verification**, in 4
- * **UserVerificationRequirement**, in 5.8.6
- * **User Verified**, in 4
- * **UV**, in 4
- * **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 encode**
- * [FETCH] defines the following terms:
 - + **window**
- * [HTML] defines the following terms:
 - + **ascii serialization of an origin**
 - + **effective domain**
 - + **environment settings object**
 - + **global object**
 - + **is a registrable domain suffix of or is equal to**
 - + **is not a registrable domain suffix of and is not equal to**
 - + **origin**
 - + **relevant settings object**
- * [HTML52] defines the following terms:

4404	+ document.domain
4405	+ opaque origin
4406	+ origin
4407	* [INFRA] defines the following terms:
4408	+ append (for list)
4409	+ append (for set)
4410	+ continue
4411	+ for each (for list)
4412	+ for each (for map)
4413	+ is empty
4414	+ is not empty
4415	+ item
4416	+ list
4417	+ map
4418	+ ordered set
4419	+ remove
4420	+ set
4421	* [secure-contexts] defines the following terms:
4422	+ secure context
4423	* [TokenBinding] defines the following terms:
4424	+ token binding
4425	+ token binding id
4426	* [URL] defines the following terms:
4427	+ domain
4428	+ empty host
4429	+ host
4430	+ ipv4 address
4431	+ ipv6 address
4432	+ opaque host
4433	+ url serializer
4434	+ valid domain
4435	+ valid domain string
4436	* [WebCryptoAPI] defines the following terms:
4437	+ recognized algorithm name
4438	* [WebIDL] defines the following terms:
4439	+ ArrayBuffer
4440	+ BufferSource
4441	+ ConstraintError
4442	+ DOMException
4443	+ DOMString
4444	+ NotAllowedError
4445	+ NotFoundError
4446	+ NotSupportedError
4447	+ Promise
4448	+ SameObject
4449	+ SecureContext
4450	+ SecurityError
4451	+ TypeError
4452	+ USVString
4453	+ UnknownError
4454	+ Unscopable
4455	+ boolean
4456	+ interface object
4457	+ long
4458	+ present
4459	+ simple exception
4460	+ unsigned long

5334	+ document.domain
5335	+ opaque origin
5336	+ origin
5337	* [INFRA] defines the following terms:
5338	+ append (for list)
5339	+ append (for set)
5340	+ byte sequence
5341	+ continue
5342	+ empty
5343	+ for each (for list)
5344	+ for each (for map)
5345	+ is empty
5346	+ is not empty
5347	+ item (for list)
5348	+ item (for struct)
5349	+ list
5350	+ map
5351	+ ordered set
5352	+ remove
5353	+ set
5354	+ size
5355	+ struct
5356	+ while
5357	+ willful violation
5358	* [mixed-content] defines the following terms:
5359	+ a priori authenticated url
5360	* [page-visibility] defines the following terms:
5361	+ visibility states
5362	* [secure-contexts] defines the following terms:
5363	+ secure contexts
5364	* [TokenBinding] defines the following terms:
5365	+ token binding
5366	+ token binding id
5367	* [URL] defines the following terms:
5368	+ domain
5369	+ empty host
5370	+ host
5371	+ ipv4 address
5372	+ ipv6 address
5373	+ opaque host
5374	+ url serializer
5375	+ valid domain
5376	+ valid domain string
5377	* [WebCryptoAPI] defines the following terms:
5378	+ recognized algorithm name
5379	* [WebIDL] defines the following terms:
5380	+ AbortError
5381	+ ArrayBuffer
5382	+ BufferSource
5383	+ ConstraintError
5384	+ DOMException
5385	+ DOMString
5386	+ Exposed
5387	+ NotAllowedError
5388	+ NotSupportedError
5389	+ Promise
5390	+ SameObject
5391	+ SecureContext
5392	+ SecurityError
5393	+ USVString
5394	+ UnknownError
5395	+ boolean
5396	+ interface object
5397	+ long
5398	+ present
5399	+ unsigned long

4461
4462
4463
4464
4465
4466
4467
4468
4469
4470
4471
4472
4473
4474
4475
4476
4477
4478
4479
4480
4481
4482
4483
4484
4485
4486
4487

References

Normative References

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

[CREDENTIAL-MANAGEMENT-1]
Mike West. Credential Management Level 1. URL: <https://www.w3.org/TR/credential-management-1/>

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

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

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

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

[FIDOReg]
R. Lindemann; D. Baghdasaryan; B. Hill. FIDO UAF Registry of Predefined Values. FIDO Alliance Proposed Standard. URL: <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-reg-v1.0-ps-20141208.html>

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

[HTML52]
Steve Faulkner; et al. HTML 5.2. URL: <https://www.w3.org/TR/html52/>

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

5400
5401
5402
5403
5404
5405
5406
5407
5408
5409
5410
5411
5412
5413
5414
5415
5416
5417
5418
5419
5420
5421
5422
5423
5424
5425
5426
5427
5428
5429
5430
5431
5432
5433
5434
5435
5436
5437
5438
5439
5440
5441
5442
5443
5444
5445
5446
5447
5448
5449
5450
5451
5452
5453
5454
5455
5456
5457
5458
5459
5460
5461
5462
5463
5464
5465
5466
5467
5468
5469

* [whatwg html] defines the following terms:
+ focus

References

Normative References

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

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

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

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

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

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

[FIDO-CTAP]
R. Lindemann; et al. FIDO 2.0: Client to Authenticator Protocol. FIDO Alliance Review Draft. URL: <https://fidoalliance.org/specs/fido-v2.0-rd-20170927/fido-client-to-authenticator-protocol-v2.0-rd-20170927.html>

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

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

[FIDOReg]
R. Lindemann; D. Baghdasaryan; B. Hill. FIDO UAF Registry of Predefined Values. FIDO Alliance Proposed Standard. URL: <https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-reg-v1.0-ps-20141208.html>

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

[HTML52]
Steve Faulkner; et al. HTML 5.2. 2 November 2017. PR. URL: <https://www.w3.org/TR/html52/>

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

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

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

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

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

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

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

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

[SECURE-CONTEXTS]
Mike West. Secure Contexts. 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. URL: <https://www.w3.org/TR/WebCryptoAPI/>

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

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

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

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

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

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

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

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

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

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

[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. URL:
<https://heycam.github.io/webidl/>

[WebIDL-1]
Cameron McCormack. WebIDL Level 1. 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>

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

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

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

[GeoJSON]
The GeoJSON Format Specification. URL:
<http://geojson.org/geojson-spec.html>

[ISOBiometricVocabulary]
ISO/IEC JTC1/SC37. Information technology -- Vocabulary -- Biometrics. 15 December 2012. International Standard: ISO/IEC 2382-37:2012(E) First Edition. URL:
http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194_ISOIEC_2382-37_2012.zip

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

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

[RFC6265]
A. Barth. HTTP State Management Mechanism. April 2011. Proposed Standard. URL: <https://tools.ietf.org/html/rfc6265>

[RFC6454]
A. Barth. The Web Origin Concept. December 2011. Proposed Standard. URL: <https://tools.ietf.org/html/rfc6454>

[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>

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

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

[GeoJSON]
The GeoJSON Format Specification. URL:
<http://geojson.org/geojson-spec.html>

[ISOBiometricVocabulary]
ISO/IEC JTC1/SC37. Information technology -- Vocabulary -- Biometrics. 15 December 2012. International Standard: ISO/IEC 2382-37:2012(E) First Edition. URL:
http://standards.iso.org/ittf/PubliclyAvailableStandards/c055194_ISOIEC_2382-37_2012.zip

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

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

[RFC6265]
A. Barth. HTTP State Management Mechanism. April 2011. Proposed Standard. URL: <https://tools.ietf.org/html/rfc6265>

[RFC6454]
A. Barth. The Web Origin Concept. December 2011. Proposed Standard. URL: <https://tools.ietf.org/html/rfc6454>

4637 [RFC7515]
4638 M. Jones; J. Bradley; N. Sakimura. JSON Web Signature (JWS). May
4639 2015. Proposed Standard. URL:
4640 https://tools.ietf.org/html/rfc7515
4641
4642 [RFC8017]
4643 K. Moriarty, Ed.; et al. PKCS #1: RSA Cryptography
4644 Specifications Version 2.2. November 2016. Informational. URL:
4645 https://tools.ietf.org/html/rfc8017
4646
4647 [TPMv2-EK-Profile]
4648 TCG EK Credential Profile for TPM Family 2.0. URL:
4649 http://www.trustedcomputinggroup.org/wp-content/uploads/Credenti
4650 al_Profile_EK_V2.0_R14_published.pdf
4651
4652 [TPMv2-Part1]
4653 Trusted Platform Module Library, Part 1: Architecture. URL:
4654 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
4655 2.0-Part-1-Architecture-01.38.pdf
4656
4657 [TPMv2-Part2]
4658 Trusted Platform Module Library, Part 2: Structures. URL:
4659 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
4660 2.0-Part-2-Structures-01.38.pdf
4661
4662 [TPMv2-Part3]
4663 Trusted Platform Module Library, Part 3: Commands. URL:
4664 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
4665 2.0-Part-3-Commands-01.38.pdf
4666
4667 [UAFProtocol]
4668 R. Lindemann; et al. FIDO UAF Protocol Specification v1.0. FIDO
4669 Alliance Proposed Standard. URL:
4670 https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua
4671 f-protocol-v1.0-ps-20141208.html
4672
4673 IDL Index
4674
4675 [SecureContext]
4676 interface PublicKeyCredential : Credential {
4677 [SameObject] readonly attribute ArrayBuffer rawId;
4678 [SameObject] readonly attribute AuthenticatorResponse response;
4679 [SameObject] readonly attribute AuthenticationExtensions clientExtensionResu
4680 lts;
4681 };
4682
4683 partial dictionary CredentialCreationOptions {
4684 MakePublicKeyCredentialOptions publicKey;
4685 };
4686
4687 partial dictionary CredentialRequestOptions {
4688 PublicKeyCredentialRequestOptions publicKey;
4689 };
4690
4691 [SecureContext]
4692 partial interface PublicKeyCredential {
4693 [Unscopable] Promise < boolean > isPlatformAuthenticatorAvailable();
4694 };
4695
4696 [SecureContext]
4697 interface AuthenticatorResponse {
4698 [SameObject] readonly attribute ArrayBuffer clientDataJSON;
4699 };
4700
4701 [SecureContext]
4702 interface AuthenticatorAttestationResponse : AuthenticatorResponse {
4703 [SameObject] readonly attribute ArrayBuffer attestationObject;
4704 };
4705
4706

5610 [RFC7515]
5611 M. Jones; J. Bradley; N. Sakimura. JSON Web Signature (JWS). May
5612 2015. Proposed Standard. URL:
5613 https://tools.ietf.org/html/rfc7515
5614
5615 [RFC8017]
5616 K. Moriarty, Ed.; et al. PKCS #1: RSA Cryptography
5617 Specifications Version 2.2. November 2016. Informational. URL:
5618 https://tools.ietf.org/html/rfc8017
5619
5620 [TPMv2-EK-Profile]
5621 TCG EK Credential Profile for TPM Family 2.0. URL:
5622 http://www.trustedcomputinggroup.org/wp-content/uploads/Credenti
5623 al_Profile_EK_V2.0_R14_published.pdf
5624
5625 [TPMv2-Part1]
5626 Trusted Platform Module Library, Part 1: Architecture. URL:
5627 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
5628 2.0-Part-1-Architecture-01.38.pdf
5629
5630 [TPMv2-Part2]
5631 Trusted Platform Module Library, Part 2: Structures. URL:
5632 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
5633 2.0-Part-2-Structures-01.38.pdf
5634
5635 [TPMv2-Part3]
5636 Trusted Platform Module Library, Part 3: Commands. URL:
5637 http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-
5638 2.0-Part-3-Commands-01.38.pdf
5639
5640 [UAFProtocol]
5641 R. Lindemann; et al. FIDO UAF Protocol Specification v1.0. FIDO
5642 Alliance Proposed Standard. URL:
5643 https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-ua
5644 f-protocol-v1.0-ps-20141208.html
5645
5646 IDL Index
5647
5648 [SecureContext, Exposed=Window]
5649 interface PublicKeyCredential : Credential {
5650 [SameObject] readonly attribute ArrayBuffer rawId;
5651 [SameObject] readonly attribute AuthenticatorResponse response;
5652 AuthenticationExtensions getClientExtensionResults();
5653 };
5654
5655 partial dictionary CredentialCreationOptions {
5656 MakePublicKeyCredentialOptions publicKey;
5657 };
5658
5659 partial dictionary CredentialRequestOptions {
5660 PublicKeyCredentialRequestOptions publicKey;
5661 };
5662
5663 [SecureContext, Exposed=Window]
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

```
4707 [SecureContext]
4708 interface AuthenticatorAssertionResponse : AuthenticatorResponse {
4709     [SameObject] readonly attribute ArrayBuffer authenticatorData;
4710     [SameObject] readonly attribute ArrayBuffer signature;
4711 };
4712
4713 dictionary PublicKeyCredentialParameters {
4714     required PublicKeyCredentialType type;
4715     required COSEAlgorithmIdentifier alg;
4716 };
4717
4718 dictionary MakePublicKeyCredentialOptions {
4719     required PublicKeyCredentialEntity rp;
4720     required PublicKeyCredentialUserEntity user;
4721
4722     required BufferSource challenge;
4723     required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
4724
4725     unsigned long timeout;
4726     sequence<PublicKeyCredentialDescriptor> excludeCredentials = [];
4727     AuthenticatorSelectionCriteria authenticatorSelection;
4728
4729     AuthenticationExtensions extensions;
4730 };
4731
4732 dictionary PublicKeyCredentialEntity {
4733     DOMString id;
4734     DOMString name;
4735     USVString icon;
4736 };
4737
4738 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
4739     DOMString displayName;
4740 };
4741
4742 dictionary AuthenticatorSelectionCriteria {
4743     AuthenticatorAttachment aa; // authenticatorAttachment
4744     boolean rk = false; // requireResidentKey
4745     boolean uv = false; // requireUserVerification
4746 };
4747
4748 enum AuthenticatorAttachment {
4749     "plat" // Platform attachment
4750     "xplat" // Cross-platform attachment
4751 };
4752
4753 dictionary PublicKeyCredentialRequestOptions {
4754     required BufferSource challenge;
4755     unsigned long timeout;
4756     USVString rpId;
4757     sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
4758
4759     AuthenticationExtensions extensions;
4760 };
4761
4762 typedef record<DOMString, any> AuthenticationExtensions;
4763
4764 dictionary CollectedClientData {
```

```
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
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
5707     USVString icon;
5708 };
5709
5710 dictionary PublicKeyCredentialRpEntity : PublicKeyCredentialEntity {
5711     DOMString id;
5712 };
5713
5714 dictionary PublicKeyCredentialUserEntity : PublicKeyCredentialEntity {
5715     required BufferSource id;
5716     required DOMString displayName;
5717 };
5718
5719 dictionary AuthenticatorSelectionCriteria {
5720     AuthenticatorAttachment authenticatorAttachment;
5721     boolean requireResidentKey = false;
5722     UserVerificationRequirement userVerification = "preferred";
5723 };
5724
5725 enum AuthenticatorAttachment {
5726     "platform" // Platform attachment
5727     "cross-platform" // Cross-platform attachment
5728 };
5729
5730 enum AttestationConveyancePreference {
5731     "none",
5732     "indirect",
5733     "direct"
5734 };
5735
5736 dictionary PublicKeyCredentialRequestOptions {
5737     required BufferSource challenge;
5738     unsigned long timeout;
5739     USVString rpId;
5740     sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
5741     UserVerificationRequirement userVerification = "preferred";
5742     AuthenticationExtensions extensions;
5743 };
5744
5745 typedef record<DOMString, any> AuthenticationExtensions;
5746
5747 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
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;

```

```

5774 enum UserVerificationRequirement {
5775     "required",
5776     "preferred",
5777     "discouraged"
5778 };
5779
5780 typedef sequence<AAGUID> AuthenticatorSelectionList;
5781
5782 typedef BufferSource AAGUID;

```

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 focus. 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

```
5812 #base64url-encodingReferenced in:
5813 * 5.1. PublicKeyCredential Interface
5814 * 5.1.3. Create a new credential - PublicKeyCredential's
5815 [[Create]](origin, options, sameOriginWithAncestors) method (2)
```


4798 * 4.1.4. Use an existing credential to make an assertion -
4799 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
4800 method (2)
4801 * 6.2. Verifying an authentication assertion
4802
4803 #cborReferenced in:
4804 * 4.1.3. Create a new credential - PublicKeyCredential's
4805 [[Create]](options) method
4806 * 4.1.4. Use an existing credential to make an assertion -
4807 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
4808 method
4809 * 5.1. Authenticator data (2)
4810 * 8. WebAuthn Extensions (2) (3)
4811 * 8.2. Defining extensions (2)
4812 * 8.3. Extending request parameters
4813 * 8.4. Client extension processing (2)
4814 * 8.5. Authenticator extension processing (2) (3) (4) (5)
4815
4816 #attestationReferenced in:
4817 * 3. Terminology
4818 * 5. WebAuthn Authenticator model (2)
4819 * 5.3. Attestation (2) (3) (4)

4820
4821 #attestation-certificateReferenced in:
4822 * 3. Terminology (2)
4823 * 7.3.1. TPM attestation statement certificate requirements

4824
4825 #attestation-key-pairReferenced in:
4826 * 3. Terminology (2)
4827 * 5.3. Attestation

4828
4829 #attestation-private-keyReferenced in:
4830 * 5. WebAuthn Authenticator model
4831 * 5.3. Attestation

4832
4833 #attestation-public-keyReferenced in:
4834 * 5.3. Attestation

4835
4836 #authenticationReferenced in:
4837 * 1. Introduction (2)
4838 * 3. Terminology (2) (3) (4) (5) (6) (7)
4839 * 6.2. Verifying an authentication assertion
4840
4841 #authentication-assertionReferenced in:
4842 * 1. Introduction
4843 * 3. Terminology (2) (3)
4844 * 4.1. PublicKeyCredential Interface
4845 * 4.2.2. Web Authentication Assertion (interface
4846 AuthenticatorAssertionResponse)
4847 * 4.5. Options for Assertion Generation (dictionary
4848 PublicKeyCredentialRequestOptions)
4849 * 8. WebAuthn Extensions
4850
4851 #authenticatorReferenced in:
4852 * 1. Introduction (2) (3) (4)
4853 * 1.1. Use Cases
4854 * 2. Conformance
4855 * 3. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
4856 (14) (15)
4857 * 4. Web Authentication API (2) (3)
4858 * 4.1. PublicKeyCredential Interface
4859 * 4.1.3. Create a new credential - PublicKeyCredential's
4860 [[Create]](options) method (2)
4861 * 4.1.4. Use an existing credential to make an assertion -

5817 * 5.1.4.1. PublicKeyCredential's
5818 [[DiscoverFromExternalSource]](origin, options,
5819 sameOriginWithAncestors) method (2)
5820 * 7.2. Verifying an authentication assertion
5821
5822 #cborReferenced in:
5823 * 5.1.3. Create a new credential - PublicKeyCredential's
5824 [[Create]](origin, options, sameOriginWithAncestors) method
5825 * 5.1.4.1. PublicKeyCredential's
5826 [[DiscoverFromExternalSource]](origin, options,
5827 sameOriginWithAncestors) method
5828 * 6.1. Authenticator data (2)
5829 * 9. WebAuthn Extensions (2) (3)
5830 * 9.2. Defining extensions (2)
5831 * 9.3. Extending request parameters
5832 * 9.4. Client extension processing (2)
5833 * 9.5. Authenticator extension processing (2) (3) (4) (5)
5834
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

5849
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:
5867 * 1. Introduction
5868 * 4. Terminology (2) (3) (4) (5) (6) (7) (8)
5869 * 5.1. PublicKeyCredential Interface
5870 * 5.2.2. Web Authentication Assertion (interface
5871 AuthenticatorAssertionResponse)
5872 * 5.5. Options for Assertion Generation (dictionary
5873 PublicKeyCredentialRequestOptions)
5874 * 9. WebAuthn Extensions
5875
5876 #authenticatorReferenced in:
5877 * 1. Introduction (2) (3) (4)
5878 * 1.1. Use Cases
5879 * 2.2. Authenticators
5880 * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
5881 (14) (15) (16) (17)
5882 * 5. Web Authentication API (2) (3)
5883 * 5.1. PublicKeyCredential Interface
5884 * 5.1.3. Create a new credential - PublicKeyCredential's
5885 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
5886 * 5.1.4.1. PublicKeyCredential's

```
4862 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
4863 method (2) (3)
4864 * 4.2. Authenticator Responses (interface AuthenticatorResponse)
4865 * 4.2.1. Information about Public Key Credential (interface
4866 AuthenticatorAttestationResponse) (2)
4867 * 4.2.2. Web Authentication Assertion (interface
4868 AuthenticatorAssertionResponse)
4869 * 4.4.4. Authenticator Attachment enumeration (enum)

4870 AuthenticatorAttachment)
4871 * 4.5. Options for Assertion Generation (dictionary)

4872 PublicKeyCredentialRequestOptions)
4873 * 5. WebAuthn Authenticator model (2) (3) (4) (5) (6)
4874 * 5.1. Authenticator data
4875 * 5.2.1. The authenticatorMakeCredential operation
4876 * 5.2.2. The authenticatorGetAssertion operation (2) (3)
4877 * 5.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)
4878 * 5.3.2. Attestation Statement Formats
4879 * 5.3.4. Generating an Attestation Object (2)
4880 * 5.3.5.1. Privacy
4881 * 5.3.5.2. Attestation Certificate and Attestation Certificate CA
4882 Compromise
4883 * 6.1. Registering a new credential
4884 * 7.2. Packed Attestation Statement Format
4885 * 7.4. Android Key Attestation Statement Format
4886 * 7.5. Android SafetyNet Attestation Statement Format
4887 * 9.5. Supported Extensions Extension (exts)
4888 * 9.6. User Verification Index Extension (uvi)
4889 * 9.7. Location Extension (loc) (2) (3) (4)
4890 * 9.8. User Verification Method Extension (uvm)
4891 * 11. Sample scenarios

4892 #authorization-gestureReferenced in:
4893 * 1.1.1. Registration
4894 * 1.1.2. Authentication
4895 * 1.1.3. Other use cases and configurations
4896 * 3. Terminology (2) (3) (4) (5) (6)

4897

4898 #biometric-recognitionReferenced in:
4899 * 3. Terminology (2)

4900

4901 #ceremonyReferenced in:
4902 * 1. Introduction
4903 * 3. Terminology (2) (3) (4) (5) (6) (7)
4904 * 6.1. Registering a new credential
4905 * 6.2. Verifying an authentication assertion

4906

4907 #clientReferenced in:
4908 * 3. Terminology
4909 * 4.1.5. Platform Authenticator Availability - PublicKeyCredential's
4910 isPlatformAuthenticatorAvailable() method (2) (3) (4)
4911

4912

4913 #client-side-resident-credential-private-keyReferenced in:
4914 * 3. Terminology (2)
4915 * 4.1.3. Create a new credential - PublicKeyCredential's
4916 [[Create]](options) method
4917 * 4.4.3. Authenticator Selection Criteria (dictionary
4918 AuthenticatorSelectionCriteria) (2)
4919 * 5.2.1. The authenticatorMakeCredential operation

4920

4921 #conforming-user-agentReferenced in:
4922 * 1. Introduction
```

```
5887 [[DiscoverFromExternalSource]](origin, options,
5888 sameOriginWithAncestors) method (2) (3) (4) (5)
5889 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
5890 * 5.2.1. Information about Public Key Credential (interface
5891 AuthenticatorAttestationResponse) (2)
5892 * 5.2.2. Web Authentication Assertion (interface
5893 AuthenticatorAssertionResponse)
5894 * 5.4.1. Public Key Entity Description (dictionary
5895 PublicKeyCredentialEntity) (2)
5896 * 5.4.3. User Account Parameters for Credential Generation
5897 (dictionary PublicKeyCredentialUserEntity)
5898 * 5.4.5. Authenticator Attachment enumeration (enum
5899 AuthenticatorAttachment)
5900 * 5.4.6. Attestation Conveyance Preference enumeration (enum
5901 AttestationConveyancePreference) (2)
5902 * 5.5. Options for Assertion Generation (dictionary
5903 PublicKeyCredentialRequestOptions)
5904 * 6. WebAuthn Authenticator model (2) (3) (4) (5) (6)
5905 * 6.1. Authenticator data
5906 * 6.2.1. The authenticatorMakeCredential operation (2) (3)
5907 * 6.2.2. The authenticatorGetAssertion operation (2) (3)
5908 * 6.3. Attestation (2) (3) (4) (5) (6) (7) (8) (9)
5909 * 6.3.2. Attestation Statement Formats
5910 * 6.3.4. Generating an Attestation Object
5911 * 6.3.5.1. Privacy
5912 * 6.3.5.2. Attestation Certificate and Attestation Certificate CA
5913 Compromise
5914 * 7.1. Registering a new credential
5915 * 8.2. Packed Attestation Statement Format
5916 * 8.4. Android Key Attestation Statement Format
5917 * 8.5. Android SafetyNet Attestation Statement Format
5918 * 10.5. Supported Extensions Extension (exts)
5919 * 10.6. User Verification Index Extension (uvi)
5920 * 10.7. Location Extension (loc) (2) (3) (4)
5921 * 10.8. User Verification Method Extension (uvm)
5922 * 12. Sample scenarios

5923 #authorization-gestureReferenced in:
5924 * 1.1.1. Registration
5925 * 1.1.2. Authentication
5926 * 1.1.3. Other use cases and configurations
5927 * 4. Terminology (2) (3) (4) (5) (6)
5928 * 5.1.4. Use an existing credential to make an assertion -
5929 PublicKeyCredential's [[Get]](options) method (2)

5930

5931 #biometric-recognitionReferenced in:
5932 * 4. Terminology (2)

5933

5934 #ceremonyReferenced in:
5935 * 1. Introduction
5936 * 4. Terminology (2) (3) (4) (5) (6) (7)
5937 * 7.1. Registering a new credential
5938 * 7.2. Verifying an authentication assertion

5939

5940 #clientReferenced in:
5941 * 4. Terminology
5942 * 5.1.6. Availability of User-Verifying Platform Authenticator -
5943 PublicKeyCredential's
5944 isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4)

5945

5946 #client-side-resident-credential-private-keyReferenced in:
5947 * 4. Terminology (2)
5948 * 5.1.3. Create a new credential - PublicKeyCredential's
5949 [[Create]](origin, options, sameOriginWithAncestors) method
5950 * 5.4.4. Authenticator Selection Criteria (dictionary
5951 AuthenticatorSelectionCriteria) (2)
5952 * 6.2.1. The authenticatorMakeCredential operation

5953

5954 #conforming-user-agentReferenced in:
5955 * 1. Introduction
```

4923 * 2. Conformance (2) (3)
4924 * 3. Terminology (2)

#credential-public-keyReferenced in:
* 3. Terminology (2) (3)
* 4.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse)
* 5. WebAuthn Authenticator model
* 5.1. Authenticator data
* 5.3. Attestation (2) (3)
* 5.3.1. Attestation data (2)
* 7.4. Android Key Attestation Statement Format
* 11.1. Registration (2)

#credential-key-pairReferenced in:
* 3. Terminology (2) (3)
* 4.1.3. Create a new credential - PublicKeyCredential's [[Create]](options) method

#credential-private-keyReferenced in:
* 3. Terminology (2) (3) (4)
* 4.1. PublicKeyCredential Interface
* 4.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse)
* 5. WebAuthn Authenticator model
* 5.2.2. The authenticatorGetAssertion operation
* 5.3. Attestation (2)

5957 * 2.1. User Agents
5958 * 2.2. Authenticators
5959 * 4. Terminology (2)

#credential-idReferenced in:
* 4. Terminology (2) (3) (4)
* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method
* 5.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse)
* 6.2.2. The authenticatorGetAssertion operation (2)
* 6.3.1. Attested credential data
* 7.1. Registering a new credential
* 8.6. FIDO U2F Attestation Statement Format
* 12.1. Registration
* 12.3. Authentication (2) (3)

#credential-public-keyReferenced in:
* 4. Terminology (2) (3) (4) (5) (6) (7)
* 5.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse)
* 6. WebAuthn Authenticator model
* 6.3. Attestation (2) (3)
* 6.3.1. Attested credential data (2)
* 12.1. Registration (2)

#credential-key-pairReferenced in:
* 4. Terminology (2) (3)

#credential-private-keyReferenced in:
* 4. Terminology (2) (3) (4) (5) (6)
* 5.1. PublicKeyCredential Interface
* 5.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse)
* 6. WebAuthn Authenticator model
* 6.2.2. The authenticatorGetAssertion operation
* 6.3. Attestation (2)
* 7.2. Verifying an authentication assertion

#public-key-credential-sourceReferenced in:
* 4. Terminology (2) (3) (4) (5) (6) (7) (8)
* 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method

#public-key-credential-source-managing-authenticatorReferenced in:
* 4. Terminology

#public-key-credentialReferenced in:
* 1. Introduction (2) (3) (4) (5)
* 4. Terminology (2) (3) (4) (5) (6) (7) (8)
* 5. Web Authentication API (2) (3) (4)
* 5.1. PublicKeyCredential Interface
* 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method
* 5.1.4. Use an existing credential to make an assertion - PublicKeyCredential's [[Get]](options) method
* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method (2)
* 5.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse)
* 5.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)
* 5.4.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)

- * 5.5. Options for Assertion Generation (dictionary PublicKeyCredentialRequestOptions)
- * 5.8. Supporting Data Structures
- * 6. WebAuthn Authenticator model (2) (3) (4) (5)
- * 6.2.2. The authenticatorGetAssertion operation (2) (3)
- * 6.3. Attestation (2)
- * 6.3.2. Attestation Statement Formats
- * 6.3.3. Attestation Types
- * 6.3.5.2. Attestation Certificate and Attestation Certificate CA Compromise (2)
- * 7.1. Registering a new credential
- * 9. WebAuthn Extensions (2)
- * 12. Sample scenarios

#registrationReferenced in:

- * 1. Introduction (2)
- * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
- * 7.1. Registering a new credential

#relying-partyReferenced in:

- * 1. Introduction (2) (3) (4) (5) (6) (7)
- * 1.1.3. Other use cases and configurations
- * 2.3. Relying Parties
- * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13) (14) (15) (16) (17) (18) (19) (20) (21) (22) (23) (24) (25) (26) (27) (28) (29) (30)
- * 5. Web Authentication API (2) (3) (4) (5) (6) (7)
- * 5.1. PublicKeyCredential Interface (2)
- * 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
- * 5.1.4. Use an existing credential to make an assertion - PublicKeyCredential's [[Get]](options) method (2)
- * 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method (2) (3) (4)
- * 5.1.6. Availability of User-Verifying Platform Authenticator - PublicKeyCredential's isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3)
- * 5.2. Authenticator Responses (interface AuthenticatorResponse)
- * 5.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse) (2)
- * 5.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse)
- * 5.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions) (2) (3) (4) (5) (6) (7)
- * 5.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity) (2) (3)
- * 5.4.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRpEntity) (2)
- * 5.4.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria) (2) (3)
- * 5.4.5. Authenticator Attachment enumeration (enum AuthenticatorAttachment) (2) (3) (4)
- * 5.4.6. Attestation Conveyance Preference enumeration (enum AttestationConveyancePreference) (2) (3) (4) (5) (6) (7)
- * 5.5. Options for Assertion Generation (dictionary PublicKeyCredentialRequestOptions)
- * 5.8.1. Client data used in WebAuthn signatures (dictionary CollectedClientData) (2) (3) (4)
- * 5.8.4. Authenticator Transport enumeration (enum AuthenticatorTransport) (2)
- * 5.8.6. User Verification Requirement enumeration (enum UserVerificationRequirement) (2) (3) (4)
- * 6. WebAuthn Authenticator model (2)
- * 6.1. Authenticator data (2)
- * 6.1.1. Signature Counter Considerations (2) (3) (4) (5) (6)
- * 6.2.1. The authenticatorMakeCredential operation (2) (3) (4) (5) (6)
- * 6.2.2. The authenticatorGetAssertion operation (2) (3)
- * 6.3. Attestation (2) (3) (4) (5) (6)

4991	Compromise (2) (3) (4) (5) (6)
4992	* 6. Relying Party Operations (2) (3) (4)
4993	* 6.1. Registering a new credential (2) (3) (4) (5) (6) (7) (8) (9)
4994	(10) (11) (12) (13)
4995	* 6.2. Verifying an authentication assertion (2) (3) (4) (5)
4996	* 7.4. Android Key Attestation Statement Format
4997	* 8. WebAuthn Extensions (2) (3) (4)
4998	* 8.2. Defining extensions (2)
4999	* 8.3. Extending request parameters (2) (3) (4)
5000	* 8.6. Example Extension (2) (3)
5001	* 9.1. FIDO Appid Extension (appid) (2)
5002	* 9.2. Simple Transaction Authorization Extension (txAuthSimple)
5003	* 9.4. Authenticator Selection Extension (authnSel) (2) (3)
5004	* 9.5. Supported Extensions Extension (exts) (2)
5005	* 9.6. User Verification Index Extension (uvi)
5006	* 9.7. Location Extension (loc) (2)
5007	* 10.2. WebAuthn Extension Identifier Registrations (2)
5008	* 11.1. Registration (2) (3) (4) (5)
5009	* 11.2. Registration Specifically with Platform Authenticator (2) (3)
5010	* 11.3. Authentication (2) (3) (4) (5)
5011	* 11.4. Decommissioning (2)
5012	
5013	#relying-party-identifierReferenced in:
5014	* 4. Web Authentication API
5015	* 4.4. Options for Credential Creation (dictionary)
5016	MakePublicKeyCredentialOptions)
5017	* 4.5. Options for Assertion Generation (dictionary)
5018	PublicKeyCredentialRequestOptions)
5019	* 5. WebAuthn Authenticator model
5020	
5021	#rp-idReferenced in:
5022	* 3. Terminology (2) (3) (4) (5) (6)
5023	* 4. Web Authentication API (2) (3) (4) (5)
5024	* 4.1.3. Create a new credential - PublicKeyCredential's
5025	[[Create]](options) method (2)
5026	* 4.1.4. Use an existing credential to make an assertion -
5027	PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5028	method (2)
5029	* 4.4.1. Public Key Entity Description (dictionary)
5030	PublicKeyCredentialEntity)
5031	* 5. WebAuthn Authenticator model
5032	* 5.1. Authenticator data (2) (3) (4) (5) (6)
5033	* 5.2.1. The authenticatorMakeCredential operation (2) (3)
5034	* 5.2.2. The authenticatorGetAssertion operation (2) (3)
5035	* 6.1. Registering a new credential (2)
5036	* 6.2. Verifying an authentication assertion (2)
5037	* 7.4. Android Key Attestation Statement Format
5038	* 7.6. FIDO U2F Attestation Statement Format (2) (3)
5039	
5040	#public-key-credentialReferenced in:
5041	* 1. Introduction (2) (3) (4) (5)
5042	* 3. Terminology (2) (3) (4) (5) (6) (7) (8)
5043	* 4. Web Authentication API (2) (3) (4)
5044	* 4.1. PublicKeyCredential Interface
5045	* 4.1.3. Create a new credential - PublicKeyCredential's
5046	[[Create]](options) method
5047	* 4.1.4. Use an existing credential to make an assertion -
5048	PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5049	method (2)
5050	* 4.2.1. Information about Public Key Credential (interface)
5051	AuthenticatorAttestationResponse)
5052	* 4.4.1. Public Key Entity Description (dictionary)
5053	PublicKeyCredentialEntity)
5054	* 4.4.3. Authenticator Selection Criteria (dictionary)

6093	* 6.3.5.1. Privacy
6094	* 6.3.5.2. Attestation Certificate and Attestation Certificate CA
6095	Compromise (2) (3) (4) (5) (6)
6096	* 7. Relying Party Operations (2) (3) (4)
6097	* 7.1. Registering a new credential (2) (3) (4) (5) (6) (7) (8) (9)
6098	(10) (11) (12)
6099	* 7.2. Verifying an authentication assertion (2) (3) (4) (5) (6) (7)
6100	(8)
6101	* 8.4. Android Key Attestation Statement Format
6102	* 9. WebAuthn Extensions (2) (3) (4)
6103	* 9.2. Defining extensions (2)
6104	* 9.3. Extending request parameters (2) (3) (4)
6105	* 9.6. Example Extension (2) (3)
6106	* 10.1. FIDO Appid Extension (appid) (2)
6107	* 10.2. Simple Transaction Authorization Extension (txAuthSimple)
6108	* 10.4. Authenticator Selection Extension (authnSel) (2) (3)
6109	* 10.5. Supported Extensions Extension (exts) (2)
6110	* 10.6. User Verification Index Extension (uvi)
6111	* 10.7. Location Extension (loc) (2)
6112	* 11.2. WebAuthn Extension Identifier Registrations (2)
6113	* 12.1. Registration (2) (3) (4) (5)
6114	* 12.2. Registration Specifically with User Verifying Platform
6115	Authenticator (2) (3)
6116	* 12.3. Authentication (2) (3) (4) (5)
6117	* 12.5. Decommissioning (2)
6118	* 13.1. Cryptographic Challenges
6119	
6120	#relying-party-identifierReferenced in:
6121	* 4. Terminology
6122	* 5. Web Authentication API
6123	* 5.4. Options for Credential Creation (dictionary)
6124	MakePublicKeyCredentialOptions)
6125	* 5.5. Options for Assertion Generation (dictionary)
6126	PublicKeyCredentialRequestOptions)
6127	* 6. WebAuthn Authenticator model
6128	
6129	#rp-idReferenced in:
6130	* 4. Terminology (2) (3) (4) (5)
6131	* 5. Web Authentication API (2) (3) (4) (5)
6132	* 5.1.3. Create a new credential - PublicKeyCredential's
6133	[[Create]](origin, options, sameOriginWithAncestors) method (2)
6134	* 5.1.4.1. PublicKeyCredential's
6135	[[DiscoverFromExternalSource]](origin, options,
6136	sameOriginWithAncestors) method (2)
6137	* 5.4.2. RP Parameters for Credential Generation (dictionary)
6138	PublicKeyCredentialRpEntity)
6139	* 6. WebAuthn Authenticator model
6140	* 6.1. Authenticator data (2) (3) (4) (5) (6)
6141	* 6.1.1. Signature Counter Considerations
6142	* 6.2.1. The authenticatorMakeCredential operation (2)
6143	* 6.2.2. The authenticatorGetAssertion operation (2)
6144	* 7.1. Registering a new credential (2)
6145	* 7.2. Verifying an authentication assertion
6146	* 8.4. Android Key Attestation Statement Format
6147	* 8.6. FIDO U2F Attestation Statement Format

6148
6149
6150
6151
6152
6153
6154
6155
6156
6157
6158
6159
6160
6161
6162
6163
6164
6165
6166
6167
6168
6169
6170
6171
6172
6173
6174
6175
6176
6177
6178
6179
6180
6181
6182
6183
6184
6185
6186
6187
6188
6189
6190
6191
6192
6193
6194
6195
6196
6197
6198
6199
6200
6201
6202

#test-of-user-presenceReferenced in:

- * 4. Terminology (2) (3) (4) (5) (6)
- * 6.2.1. The authenticatorMakeCredential operation
- * 6.2.2. The authenticatorGetAssertion operation
- * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
- * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)

#user-consentReferenced in:

- * 1. Introduction (2)
- * 4. Terminology (2)
- * 5. Web Authentication API
- * 5.1.4. Use an existing credential to make an assertion - PublicKeyCredential's [[Get]](options) method
- * 5.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse)
- * 5.4.6. Attestation Conveyance Preference enumeration (enum AttestationConveyancePreference)
- * 6. WebAuthn Authenticator model (2) (3)
- * 6.2.1. The authenticatorMakeCredential operation (2) (3) (4) (5) (6)
- * 6.2.2. The authenticatorGetAssertion operation (2) (3) (4) (5)
- * 11.2. WebAuthn Extension Identifier Registrations

#user-handleReferenced in:

- * 4. Terminology
- * 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method
- * 5.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse)
- * 5.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions)
- * 5.4.3. User Account Parameters for Credential Generation (dictionary PublicKeyCredentialUserEntity)
- * 6.2.1. The authenticatorMakeCredential operation
- * 6.2.2. The authenticatorGetAssertion operation

#user-verificationReferenced in:

- * 1. Introduction
- * 4. Terminology (2) (3) (4) (5) (6) (7) (8) (9)
- * 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
- * 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method (2) (3)
- * 5.1.6. Availability of User-Verifying Platform Authenticator - PublicKeyCredential's isUserVerifyingPlatformAuthenticatorAvailable() method (2) (3) (4) (5)
- * 5.4.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria)
- * 5.5. Options for Assertion Generation (dictionary PublicKeyCredentialRequestOptions)
- * 5.8.6. User Verification Requirement enumeration (enum

5093
5094 #concept-user-presentReferenced in:
5095 * 3. Terminology
5096 * 5.1. Authenticator data (2) (3)
5097
5098 #upReferenced in:
5099 * 5.1. Authenticator data
5100
5101 #concept-user-verifiedReferenced in:
5102 * 3. Terminology
5103 * 5.1. Authenticator data (2) (3)
5104
5105 #uvReferenced in:
5106 * 5.1. Authenticator data

5107
5108 #webauthn-clientReferenced in:
5109 * 3. Terminology (2)

5110
5111 #web-authentication-apiReferenced in:
5112 * 1. Introduction (2) (3)
5113 * 3. Terminology (2)
5114
5115 #publickeycredentialReferenced in:
5116 * 1. Introduction
5117 * 4.1. PublicKeyCredential Interface (2) (3) (4) (5) (6) (7) (8)
5118 * 4.1.3. Create a new credential - PublicKeyCredential's
5119 [[Create]](options) method (2) (3) (4) (5) (6)
5120 * 4.1.4. Use an existing credential to make an assertion -
5121 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5122 method (2) (3)
5123 * 4.1.5. Platform Authenticator Availability - PublicKeyCredential's
5124 isPlatformAuthenticatorAvailable() method
5125 * 4.7.3. Credential Descriptor (dictionary

5126 PublicKeyCredentialDescriptor)
5127 * 5.2.1. The authenticatorMakeCredential operation
5128 * 6. Relying Party Operations
5129 * 6.2. Verifying an authentication assertion
5130
5131 #dom-publickeycredential-rawidReferenced in:
5132 * 4.1. PublicKeyCredential Interface
5133 * 6.2. Verifying an authentication assertion
5134
5135 #dom-publickeycredential-responseReferenced in:
5136 * 4.1. PublicKeyCredential Interface
5137 * 4.1.3. Create a new credential - PublicKeyCredential's
5138 [[Create]](options) method
5139 * 4.1.4. Use an existing credential to make an assertion -
5140 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5141 method
5142 * 6.2. Verifying an authentication assertion
5143
5144 #dom-publickeycredential-clientextensionresultsReferenced in:
5145 * 4.1. PublicKeyCredential Interface
5146 * 4.1.3. Create a new credential - PublicKeyCredential's
5147 [[Create]](options) method
5148 * 4.1.4. Use an existing credential to make an assertion -

6203 UserVerificationRequirement) (2) (3) (4)
6204 * 6.2.1. The authenticatorMakeCredential operation (2) (3)
6205 * 6.2.2. The authenticatorGetAssertion operation (2) (3)
6206 * 10.2. Simple Transaction Authorization Extension (txAuthSimple)
6207 * 10.3. Generic Transaction Authorization Extension (txAuthGeneric)
6208 * 12.2. Registration Specifically with User Verifying Platform
6209 Authenticator
6210
6211 #concept-user-presentReferenced in:
6212 * 4. Terminology
6213 * 6.1. Authenticator data (2) (3)
6214
6215 #upReferenced in:
6216 * 6.1. Authenticator data
6217
6218 #concept-user-verifiedReferenced in:
6219 * 4. Terminology
6220 * 6.1. Authenticator data (2) (3)
6221
6222 #uvReferenced in:
6223 * 5.8.6. User Verification Requirement enumeration (enum
6224 UserVerificationRequirement) (2)
6225 * 6.1. Authenticator data
6226
6227 #webauthn-clientReferenced in:
6228 * 4. Terminology (2)
6229 * 6.2.1. The authenticatorMakeCredential operation
6230 * 6.2.2. The authenticatorGetAssertion operation
6231
6232 #web-authentication-apiReferenced in:
6233 * 1. Introduction (2) (3)
6234 * 4. Terminology (2)
6235
6236 #publickeycredentialReferenced in:
6237 * 1. Introduction
6238 * 5.1. PublicKeyCredential Interface (2) (3) (4) (5) (6) (7) (8)
6239 * 5.1.3. Create a new credential - PublicKeyCredential's
6240 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6241 * 5.1.4.1. PublicKeyCredential's
6242 [[DiscoverFromExternalSource]](origin, options,
6243 sameOriginWithAncestors) method (2)
6244 * 5.1.5. Store an existing credential - PublicKeyCredential's
6245 [[Store]](credential, sameOriginWithAncestors) method (2)
6246 * 5.1.6. Availability of User-Verifying Platform Authenticator -
6247 PublicKeyCredential's
6248 isUserVerifyingPlatformAuthenticatorAvailable() method
6249 * 5.8.3. Credential Descriptor (dictionary
6250 PublicKeyCredentialDescriptor)
6251 * 7. Relying Party Operations
6252 * 7.2. Verifying an authentication assertion

6253
6254 #dom-publickeycredential-rawidReferenced in:
6255 * 5.1. PublicKeyCredential Interface
6256 * 7.2. Verifying an authentication assertion
6257
6258 #dom-publickeycredential-getclientextensionresultsReferenced in:
6259 * 5.1. PublicKeyCredential Interface
6260 * 9.4. Client extension processing

6261
6262 #dom-publickeycredential-responseReferenced in:
6263 * 5.1. PublicKeyCredential Interface
6264 * 5.1.3. Create a new credential - PublicKeyCredential's
6265 [[Create]](origin, options, sameOriginWithAncestors) method
6266 * 5.1.4.1. PublicKeyCredential's

5149 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5150 method
5151 * 8.4. Client extension processing
5152
5153 #dom-publickeycredential-identifier-slotReferenced in:
5154 * 4.1. PublicKeyCredential Interface (2)
5155 * 4.1.3. Create a new credential - PublicKeyCredential's
5156 [[Create]](options) method
5157 * 4.1.4. Use an existing credential to make an assertion -
5158 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5159 method

5160
5161 #dom-credentialcreationoptions-publickeyReferenced in:
5162 * 4.1.3. Create a new credential - PublicKeyCredential's
5163 [[Create]](options) method (2) (3)
5164
5165 #dom-credentialrequestoptions-publickeyReferenced in:
5166 * 4.1.4. Use an existing credential to make an assertion -
5167 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5168 method (2) (3)
5169
5170 #dom-publickeycredential-create-slotReferenced in:
5171 * 4.1. PublicKeyCredential Interface

5172
5173 #dom-publickeycredential-create-options-optionsReferenced in:
5174 * 6.1. Registering a new credential

5175
5176 #dom-publickeycredential-discoverfromexternalsource-slotReferenced in:
5177 * 4.1. PublicKeyCredential Interface

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 * 5.1. PublicKeyCredential Interface
6298 * 5.6. Abort operations with AbortSignal (2) (3) (4) (5)
6299 * 6.2.1. The authenticatorMakeCredential operation
6300

6301 #dom-publickeycredential-create-origin-options-sameoriginwithancestors-
6302 originReferenced in:
6303 * 5.1.3. Create a new credential - PublicKeyCredential's
6304 [[Create]](origin, options, sameOriginWithAncestors) method
6305

6306 #dom-publickeycredential-create-origin-options-sameoriginwithancestors-
6307 optionsReferenced in:
6308 * 7.1. Registering a new credential
6309

6310 #effective-user-verification-requirement-for-credential-creationReferen-
6311 ced in:
6312 * 6.2.1. The authenticatorMakeCredential operation
6313

6314 #credentialcreationdata-attestationobjectresultReferenced in:
6315 * 5.1.3. Create a new credential - PublicKeyCredential's
6316 [[Create]](origin, options, sameOriginWithAncestors) method
6317

6318 #credentialcreationdata-clientdatajsonresultReferenced in:
6319 * 5.1.3. Create a new credential - PublicKeyCredential's
6320 [[Create]](origin, options, sameOriginWithAncestors) method
6321

6322 #credentialcreationdata-attestationconveyancepreferenceoptionReferenced
6323 in:
6324 * 5.1.3. Create a new credential - PublicKeyCredential's
6325 [[Create]](origin, options, sameOriginWithAncestors) method
6326

6327 #credentialcreationdata-clientextensionresultsReferenced in:
6328 * 5.1.3. Create a new credential - PublicKeyCredential's
6329 [[Create]](origin, options, sameOriginWithAncestors) method
6330

6331 #dom-publickeycredential-collectfromcredentialstore-slotReferenced in:
6332 * 5.1.4. Use an existing credential to make an assertion -
6333 PublicKeyCredential's [[Get]](options) method
6334

6335 #dom-publickeycredential-discoverfromexternalsource-slotReferenced in:
6336 * 5.1. PublicKeyCredential Interface

5178
5179
5180
5181
5182
5183
5184
5185
5186
5187
5188
5189
5190
5191
5192
5193
5194
5195
5196
5197
5198
5199
5200
5201
5202
5203
5204

#authenticatorresponseReferenced in:
* 4.1. PublicKeyCredential Interface (2)
* 4.2. Authenticator Responses (interface AuthenticatorResponse) (2)
* 4.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse) (2)
* 4.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse) (2)
#dom-authenticatorresponse-clientdatajsonReferenced in:
* 4.1.3. Create a new credential - PublicKeyCredential's [[Create]](options) method (2)
* 4.1.4. Use an existing credential to make an assertion - PublicKeyCredential's [[DiscoverFromExternalSource]](options) method (2)
* 4.2. Authenticator Responses (interface AuthenticatorResponse)
* 4.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse)
* 4.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse)
* 6.1. Registering a new credential (2)
* 6.2. Verifying an authentication assertion
#authenticatorattestationresponseReferenced in:
* 4.1. PublicKeyCredential Interface
* 4.1.3. Create a new credential - PublicKeyCredential's [[Create]](options) method (2)

6337
6338
6339
6340
6341
6342
6343
6344
6345
6346
6347
6348
6349
6350
6351
6352
6353
6354
6355
6356
6357
6358
6359
6360
6361
6362
6363
6364
6365
6366
6367
6368
6369
6370
6371
6372
6373
6374
6375
6376
6377
6378
6379
6380
6381
6382
6383
6384
6385
6386
6387
6388
6389
6390
6391
6392
6393
6394
6395
6396
6397
6398
6399
6400
6401
6402
6403
6404
6405
6406

* 5.1.4. Use an existing credential to make an assertion - PublicKeyCredential's [[Get]](options) method
* 5.6. Abort operations with AbortSignal (2) (3) (4) (5)
* 6.2.2. The authenticatorGetAssertion operation
#dom-publickeycredential-discoverfromexternalsource-origin-options-sameoriginwithancestors-originReferenced in:
* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method
#effective-user-verification-requirement-for-assertionReferenced in:
* 6.2.2. The authenticatorGetAssertion operation
#assertioncreationdata-credentialidresultReferenced in:
* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method (2) (3)
#assertioncreationdata-clientdatajsonresultReferenced in:
* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method
#assertioncreationdata-authenticatordataresultReferenced in:
* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method
#assertioncreationdata-signatureresultReferenced in:
* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method
#assertioncreationdata-userhandlerresultReferenced in:
* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method
#assertioncreationdata-clientextensionresultsReferenced in:
* 5.1.4.1. PublicKeyCredential's [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) method
#authenticatorresponseReferenced in:
* 5.1. PublicKeyCredential Interface (2)
* 5.2. Authenticator Responses (interface AuthenticatorResponse) (2)
* 5.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse) (2)
* 5.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse) (2)
#dom-authenticatorresponse-clientdatajsonReferenced in:
* 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)
* 5.2. Authenticator Responses (interface AuthenticatorResponse)
* 5.2.1. Information about Public Key Credential (interface AuthenticatorAttestationResponse)
* 5.2.2. Web Authentication Assertion (interface AuthenticatorAssertionResponse)
* 7.1. Registering a new credential (2)
* 7.2. Verifying an authentication assertion
#authenticatorattestationresponseReferenced in:
* 5.1. PublicKeyCredential Interface
* 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method

5205 * 4.2.1. Information about Public Key Credential (interface
5206 AuthenticatorAttestationResponse) (2)
5207 * 6. Relying Party Operations
5208 * 6.1. Registering a new credential (2) (3)
5209
5210 #dom-authenticatorattestationresponse-attestationobjectReferenced in:
5211 * 4.1.3. Create a new credential - PublicKeyCredential's
5212 [[Create]](options) method
5213 * 4.2.1. Information about Public Key Credential (interface
5214 AuthenticatorAttestationResponse)
5215 * 6.1. Registering a new credential
5216
5217 #authenticatorassertionresponseReferenced in:
5218 * 3. Terminology
5219 * 4.1. PublicKeyCredential Interface
5220 * 4.1.4. Use an existing credential to make an assertion -
5221 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5222 method
5223 * 4.2.2. Web Authentication Assertion (interface
5224 AuthenticatorAssertionResponse) (2)
5225 * 6. Relying Party Operations
5226
5227 #dom-authenticatorassertionresponse-authenticatordataReferenced in:
5228 * 4.1.4. Use an existing credential to make an assertion -
5229 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5230 method (2)
5231 * 4.2.2. Web Authentication Assertion (interface
5232 AuthenticatorAssertionResponse)
5233 * 6.2. Verifying an authentication assertion
5234
5235 #dom-authenticatorassertionresponse-signatureReferenced in:
5236 * 4.1.4. Use an existing credential to make an assertion -
5237 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5238 method (2)
5239 * 4.2.2. Web Authentication Assertion (interface
5240
5241 AuthenticatorAssertionResponse)
5242 * 6.2. Verifying an authentication assertion
5243
5244 #dictdef-publickeycredentialparametersReferenced in:
5245 * 4.3. Parameters for Credential Generation (dictionary
5246 PublicKeyCredentialParameters)
5247 * 4.4. Options for Credential Creation (dictionary
5248 MakePublicKeyCredentialOptions) (2)
5249
5250 #dom-publickeycredentialparameters-typeReferenced in:
5251 * 4.1.3. Create a new credential - PublicKeyCredential's
5252 [[Create]](options) method (2)
5253 * 4.3. Parameters for Credential Generation (dictionary
5254 PublicKeyCredentialParameters)
5255
5256 #dom-publickeycredentialparameters-algReferenced in:
5257 * 4.1.3. Create a new credential - PublicKeyCredential's
5258 [[Create]](options) method
5259 * 4.3. Parameters for Credential Generation (dictionary
5260 PublicKeyCredentialParameters)
5261
5262 #dictdef-makepublickeycredentialoptionsReferenced in:
5263 * 4.1.1. CredentialCreationOptions Extension
5264 * 4.1.3. Create a new credential - PublicKeyCredential's
5265 [[Create]](options) method
5266 * 4.4. Options for Credential Creation (dictionary
MakePublicKeyCredentialOptions)

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:
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 #dom-makepublickeycredentialoptions-rpReferenced in:
6477 * 5.1.3. Create a new credential - PublicKeyCredential's
6478 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
6479 (4) (5) (6)
6480 * 5.4. Options for Credential Creation (dictionary
6481 MakePublicKeyCredentialOptions)
6482
6483 #dom-makepublickeycredentialoptions-userReferenced in:
6484 * 5.1.3. Create a new credential - PublicKeyCredential's
6485 [[Create]](origin, options, sameOriginWithAncestors) method
6486 * 5.4. Options for Credential Creation (dictionary
6487 MakePublicKeyCredentialOptions)
6488 * 7.1. Registering a new credential
6489
6490 #dom-makepublickeycredentialoptions-challengeReferenced in:
6491 * 5.1.3. Create a new credential - PublicKeyCredential's
6492 [[Create]](origin, options, sameOriginWithAncestors) method
6493 * 5.4. Options for Credential Creation (dictionary
6494 MakePublicKeyCredentialOptions)
6495
6496 #dom-makepublickeycredentialoptions-pubkeycredparamsReferenced in:
6497 * 5.1.3. Create a new credential - PublicKeyCredential's
6498 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6499 * 5.4. Options for Credential Creation (dictionary
6500 MakePublicKeyCredentialOptions)
6501
6502 #dom-makepublickeycredentialoptions-timeoutReferenced in:
6503 * 5.1.3. Create a new credential - PublicKeyCredential's
6504 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6505 * 5.4. Options for Credential Creation (dictionary
6506 MakePublicKeyCredentialOptions)
6507
6508 #dom-makepublickeycredentialoptions-excludecredentialsReferenced in:
6509 * 5.1.3. Create a new credential - PublicKeyCredential's
6510 [[Create]](origin, options, sameOriginWithAncestors) method
6511 * 5.4. Options for Credential Creation (dictionary
6512 MakePublicKeyCredentialOptions)
6513
6514 #dom-makepublickeycredentialoptions-authenticatorselectionReferenced
6515 in:
6516 * 5.1.3. Create a new credential - PublicKeyCredential's
6517 [[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
6518 (4) (5) (6)
6519 * 5.4. Options for Credential Creation (dictionary
6520 MakePublicKeyCredentialOptions)
6521 * 6.2.1. The authenticatorMakeCredential operation
6522
6523 #dom-makepublickeycredentialoptions-attestationReferenced in:
6524 * 5.1.3. Create a new credential - PublicKeyCredential's
6525 [[Create]](origin, options, sameOriginWithAncestors) method
6526 * 5.4. Options for Credential Creation (dictionary
6527 MakePublicKeyCredentialOptions)
6528
6529 #dom-makepublickeycredentialoptions-extensionsReferenced in:
6530 * 5.1.3. Create a new credential - PublicKeyCredential's
6531 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6532 * 5.4. Options for Credential Creation (dictionary
6533 MakePublicKeyCredentialOptions)
6534 * 9.3. Extending request parameters
6535
6536 #dictdef-publickeycredentialentityReferenced in:
6537 * 5.4.1. Public Key Entity Description (dictionary
6538 PublicKeyCredentialEntity) (2)
6539 * 5.4.2. RP Parameters for Credential Generation (dictionary
6540 PublicKeyCredentialRpEntity)
6541

```


(dictionary PublicKeyCredentialUserEntity)
* 5.2.1. The authenticatorMakeCredential operation

#dom-publickeycredentialentity-idReferenced in:
* 4.1.3. Create a new credential - PublicKeyCredential's [[Create]](options) method (2) (3) (4) (5)
* 4.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions) (2) (3)
* 4.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)
* 5.2.1. The authenticatorMakeCredential operation (2)

#dom-publickeycredentialentity-nameReferenced in:
* 4.1.3. Create a new credential - PublicKeyCredential's [[Create]](options) method (2)
* 4.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions) (2)
* 4.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

#dom-publickeycredentialentity-iconReferenced in:
* 4.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

#dictdef-publickeycredentialuserentityReferenced in:
* 4.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions) (2)
* 4.4.2. User Account Parameters for Credential Generation (dictionary PublicKeyCredentialUserEntity) (2)
* 5.2.1. The authenticatorMakeCredential operation

#dom-publickeycredentialuserentity-displaynameReferenced in:
* 4.1.3. Create a new credential - PublicKeyCredential's [[Create]](options) method
* 4.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions)
* 4.4.2. User Account Parameters for Credential Generation (dictionary PublicKeyCredentialUserEntity)

#dictdef-authenticatorselectioncriteriaReferenced in:
* 4.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions) (2)
* 4.4.3. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria) (2)

#dom-authenticatorselectioncriteria-aaReferenced in:

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

#dom-publickeycredentialentity-nameReferenced in:
* 5.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions) (2)
* 5.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)
* 6.2.1. The authenticatorMakeCredential operation (2)

#dom-publickeycredentialentity-iconReferenced in:
* 5.4.1. Public Key Entity Description (dictionary PublicKeyCredentialEntity)

#dictdef-publickeycredentialrpentityReferenced in:
* 5.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions) (2)
* 5.4.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRpEntity) (2)
* 6.2.1. The authenticatorMakeCredential operation

#dom-publickeycredentialrpentity-idReferenced in:
* 5.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions)
* 5.4.2. RP Parameters for Credential Generation (dictionary PublicKeyCredentialRpEntity)
* 6.2.1. The authenticatorMakeCredential operation (2) (3) (4)

#dictdef-publickeycredentialuserentityReferenced in:
* 5.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions) (2)
* 5.4.3. User Account Parameters for Credential Generation (dictionary PublicKeyCredentialUserEntity) (2)
* 6.2.1. The authenticatorMakeCredential operation

#dom-publickeycredentialuserentity-idReferenced in:
* 5.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions)
* 5.4.3. User Account Parameters for Credential Generation (dictionary PublicKeyCredentialUserEntity)
* 6.2.1. The authenticatorMakeCredential operation

#dom-publickeycredentialuserentity-displaynameReferenced in:
* 4. Terminology
* 5.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions)
* 5.4.3. User Account Parameters for Credential Generation (dictionary PublicKeyCredentialUserEntity)
* 6.2.1. The authenticatorMakeCredential operation

#dictdef-authenticatorselectioncriteriaReferenced in:
* 5.4. Options for Credential Creation (dictionary MakePublicKeyCredentialOptions) (2)
* 5.4.4. Authenticator Selection Criteria (dictionary AuthenticatorSelectionCriteria) (2)

#dom-authenticatorselectioncriteria-authenticatorattachmentReferenced

5373 * 4.1.3. Create a new credential - PublicKeyCredential's
5374 [[Create]](options) method
5375 * 4.4.3. Authenticator Selection Criteria (dictionary

5376 AuthenticatorSelectionCriteria)
5377
5378 #dom-authenticatorselectioncriteria-rkReferenced in:
5379 * 4.1.3. Create a new credential - PublicKeyCredential's
5380 [[Create]](options) method (2)
5381 * 4.4.3. Authenticator Selection Criteria (dictionary
5382 AuthenticatorSelectionCriteria)

5383
5384 #dom-authenticatorselectioncriteria-uvReferenced in:
5385 * 4.1.3. Create a new credential - PublicKeyCredential's
5386 [[Create]](options) method
5387 * 4.4.3. Authenticator Selection Criteria (dictionary
5388 AuthenticatorSelectionCriteria)
5389
5390 #enumdef-authenticatorattachmentReferenced in:
5391 * 4.4.3. Authenticator Selection Criteria (dictionary
5392 AuthenticatorSelectionCriteria) (2)
5393 * 4.4.4. Authenticator Attachment enumeration (enum
5394 AuthenticatorAttachment) (2)
5395
5396 #platform-authenticatorsReferenced in:
5397 * 4.1.5. Platform Authenticator Availability - PublicKeyCredential's
5398 isPlatformAuthenticatorAvailable() method (2) (3) (4) (5)
5399 * 4.4.4. Authenticator Attachment enumeration (enum

5400 AuthenticatorAttachment) (2)
5401 * 11.1. Registration
5402 * 11.2. Registration Specifically with Platform Authenticator (2)

5403
5404 #roaming-authenticatorsReferenced in:
5405 * 1.1.3. Other use cases and configurations
5406 * 4.4.4. Authenticator Attachment enumeration (enum
5407 AuthenticatorAttachment) (2)
5408 * 11.1. Registration
5409
5410 #platform-attachmentReferenced in:
5411 * 4.4.4. Authenticator Attachment enumeration (enum
5412 AuthenticatorAttachment)
5413
5414 #cross-platform-attachedReferenced in:
5415 * 4.4.4. Authenticator Attachment enumeration (enum
5416 AuthenticatorAttachment) (2)
5417

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-requireresidentkeyReferenced 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-serververificationReferenced 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
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 #cross-platform-attachedReferenced in:
6646 * 5.4.5. Authenticator Attachment enumeration (enum
6647 AuthenticatorAttachment) (2)
6648
6649 #attestation-conveyanceReferenced in:
6650 * 4. Terminology
6651 * 5.4. Options for Credential Creation (dictionary
6652 MakePublicKeyCredentialOptions)
6653 * 5.4.6. Attestation Conveyance Preference enumeration (enum
6654 AttestationConveyancePreference)
6655
6656 #enumdef-attestationconveyancepreferenceReferenced in:
6657 * 5.4. Options for Credential Creation (dictionary
6658 MakePublicKeyCredentialOptions) (2)
6659 * 5.4.6. Attestation Conveyance Preference enumeration (enum
6660 AttestationConveyancePreference) (2)
6661
6662 #dom-attestationconveyancepreference-noneReferenced in:
6663 * 5.4. Options for Credential Creation (dictionary
6664 MakePublicKeyCredentialOptions)
6665 * 5.4.6. Attestation Conveyance Preference enumeration (enum
6666 AttestationConveyancePreference)
6667
6668 #dom-attestationconveyancepreference-indirectReferenced in:

5418 #dictdef-publickeycredentialrequestoptionsReferenced in:
5419 * 4.1.2. CredentialRequestOptions Extension
5420 * 4.5. Options for Assertion Generation (dictionary

5421 PublicKeyCredentialRequestOptions) (2)
5422 * 6.2. Verifying an authentication assertion
5423
5424 #dom-publickeycredentialrequestoptions-challengeReferenced in:
5425 * 4.1.4. Use an existing credential to make an assertion -
5426 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5427 method
5428 * 4.5. Options for Assertion Generation (dictionary
5429 PublicKeyCredentialRequestOptions) (2)
5430
5431 #dom-publickeycredentialrequestoptions-timeoutReferenced in:
5432 * 4.1.4. Use an existing credential to make an assertion -
5433 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5434 method (2)
5435 * 4.5. Options for Assertion Generation (dictionary
5436 PublicKeyCredentialRequestOptions)
5437
5438 #dom-publickeycredentialrequestoptions-rpidReferenced in:
5439 * 4.1.4. Use an existing credential to make an assertion -
5440 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5441 method (2) (3) (4)
5442 * 4.5. Options for Assertion Generation (dictionary
5443 PublicKeyCredentialRequestOptions)
5444 * 9.1. FIDO AppId Extension (appid)
5445
5446 #dom-publickeycredentialrequestoptions-allowcredentialsReferenced in:
5447 * 4.1.4. Use an existing credential to make an assertion -
5448 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5449 method (2) (3) (4)
5450 * 4.5. Options for Assertion Generation (dictionary

5451 PublicKeyCredentialRequestOptions)
5452
5453 #dom-publickeycredentialrequestoptions-extensionsReferenced in:
5454 * 4.1.4. Use an existing credential to make an assertion -
5455 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5456 method (2)
5457 * 4.5. Options for Assertion Generation (dictionary
5458 PublicKeyCredentialRequestOptions)
5459
5460 #typedefdef-authenticationextensionsReferenced in:
5461 * 4.1. PublicKeyCredential Interface (2)
5462 * 4.1.3. Create a new credential - PublicKeyCredential's
5463 [[Create]](options) method
5464 * 4.1.4. Use an existing credential to make an assertion -
5465 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5466 method
5467 * 4.4. Options for Credential Creation (dictionary
5468 MakePublicKeyCredentialOptions) (2)
5469 * 4.5. Options for Assertion Generation (dictionary

6669 * 5.4.6. Attestation Conveyance Preference enumeration (enum
6670 AttestationConveyancePreference)
6671
6672 #dom-attestationconveyancepreference-directReferenced in:
6673 * 5.4.6. Attestation Conveyance Preference enumeration (enum
6674 AttestationConveyancePreference)
6675
6676 #dictdef-publickeycredentialrequestoptionsReferenced in:
6677 * 5.1.2. CredentialRequestOptions Extension
6678 * 5.1.4.1. PublicKeyCredential's
6679 [[DiscoverFromExternalSource]](origin, options,
6680 sameOriginWithAncestors) method
6681 * 5.5. Options for Assertion Generation (dictionary
6682 PublicKeyCredentialRequestOptions) (2)
6683 * 7.2. Verifying an authentication assertion
6684
6685 #dom-publickeycredentialrequestoptions-challengeReferenced in:
6686 * 5.1.4.1. PublicKeyCredential's
6687 [[DiscoverFromExternalSource]](origin, options,
6688 sameOriginWithAncestors) method
6689 * 5.5. Options for Assertion Generation (dictionary
6690 PublicKeyCredentialRequestOptions) (2)
6691 * 13.1. Cryptographic Challenges
6692
6693 #dom-publickeycredentialrequestoptions-timeoutReferenced in:
6694 * 5.1.4.1. PublicKeyCredential's
6695 [[DiscoverFromExternalSource]](origin, options,
6696 sameOriginWithAncestors) method (2)
6697 * 5.5. Options for Assertion Generation (dictionary
6698 PublicKeyCredentialRequestOptions)
6699
6700 #dom-publickeycredentialrequestoptions-rpidReferenced in:
6701 * 5.1.4.1. PublicKeyCredential's
6702 [[DiscoverFromExternalSource]](origin, options,
6703 sameOriginWithAncestors) method (2) (3) (4)
6704 * 5.5. Options for Assertion Generation (dictionary
6705 PublicKeyCredentialRequestOptions)
6706 * 10.1. FIDO AppId Extension (appid)
6707
6708 #dom-publickeycredentialrequestoptions-allowcredentialsReferenced in:
6709 * 5.1.4.1. PublicKeyCredential's
6710 [[DiscoverFromExternalSource]](origin, options,
6711 sameOriginWithAncestors) method (2) (3) (4)
6712 * 5.5. Options for Assertion Generation (dictionary
6713 PublicKeyCredentialRequestOptions)
6714
6715 #dom-publickeycredentialrequestoptions-userverificationReferenced in:
6716 * 5.1.4.1. PublicKeyCredential's
6717 [[DiscoverFromExternalSource]](origin, options,
6718 sameOriginWithAncestors) method (2)
6719 * 5.5. Options for Assertion Generation (dictionary
6720 PublicKeyCredentialRequestOptions)
6721
6722 #dom-publickeycredentialrequestoptions-extensionsReferenced in:
6723 * 5.1.4.1. PublicKeyCredential's
6724 [[DiscoverFromExternalSource]](origin, options,
6725 sameOriginWithAncestors) method (2)
6726 * 5.5. Options for Assertion Generation (dictionary
6727 PublicKeyCredentialRequestOptions)
6728
6729 #typedefdef-authenticationextensionsReferenced in:
6730 * 5.1. PublicKeyCredential Interface
6731 * 5.1.3. Create a new credential - PublicKeyCredential's
6732 [[Create]](origin, options, sameOriginWithAncestors) method
6733 * 5.1.4.1. PublicKeyCredential's
6734 [[DiscoverFromExternalSource]](origin, options,
6735 sameOriginWithAncestors) method
6736 * 5.4. Options for Credential Creation (dictionary
6737 MakePublicKeyCredentialOptions) (2)
6738 * 5.5. Options for Assertion Generation (dictionary

5470 PublicKeyCredentialRequestOptions) (2)
5471 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5472 CollectedClientData) (2)
5473
5474 #dictdef-collectedclientdataReferenced in:
5475 * 4.1.3. Create a new credential - PublicKeyCredential's
5476 [[Create]](options) method
5477 * 4.1.4. Use an existing credential to make an assertion -
5478 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5479 method
5480 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5481 CollectedClientData) (2)
5482
5483 #client-dataReferenced in:
5484 * 4.2. Authenticator Responses (interface AuthenticatorResponse)
5485 * 5. WebAuthn Authenticator model (2) (3) (4)
5486 * 5.1. Authenticator data (2)
5487 * 6.1. Registering a new credential
5488 * 6.2. Verifying an authentication assertion
5489 * 8. WebAuthn Extensions
5490 * 8.4. Client extension processing
5491 * 8.6. Example Extension

5492
5493 #dom-collectedclientdata-challengeReferenced in:
5494 * 4.1.3. Create a new credential - PublicKeyCredential's
5495 [[Create]](options) method
5496 * 4.1.4. Use an existing credential to make an assertion -
5497 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5498 method
5499 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5500 CollectedClientData)
5501 * 6.1. Registering a new credential
5502 * 6.2. Verifying an authentication assertion
5503
5504 #dom-collectedclientdata-originReferenced in:
5505 * 4.1.3. Create a new credential - PublicKeyCredential's
5506 [[Create]](options) method
5507 * 4.1.4. Use an existing credential to make an assertion -
5508 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5509 method
5510 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5511 CollectedClientData)
5512 * 6.1. Registering a new credential
5513 * 6.2. Verifying an authentication assertion
5514
5515 #dom-collectedclientdata-hashalgorithmReferenced in:
5516 * 4.1.3. Create a new credential - PublicKeyCredential's
5517 [[Create]](options) method
5518 * 4.1.4. Use an existing credential to make an assertion -
5519 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5520 method
5521 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5522 CollectedClientData) (2)
5523 * 6.1. Registering a new credential
5524 * 6.2. Verifying an authentication assertion
5525
5526 #dom-collectedclientdata-tokenbindingidReferenced in:
5527 * 4.1.3. Create a new credential - PublicKeyCredential's
5528 [[Create]](options) method

6739 PublicKeyCredentialRequestOptions) (2)
6740 * 5.8.1. Client data used in WebAuthn signatures (dictionary
6741 CollectedClientData) (2)
6742
6743 #dictdef-collectedclientdataReferenced in:
6744 * 5.1.3. Create a new credential - PublicKeyCredential's
6745 [[Create]](origin, options, sameOriginWithAncestors) method
6746 * 5.1.4.1. PublicKeyCredential's
6747 [[DiscoverFromExternalSource]](origin, options,
6748 sameOriginWithAncestors) method
6749 * 5.8.1. Client data used in WebAuthn signatures (dictionary
6750 CollectedClientData) (2)
6751
6752 #client-dataReferenced in:
6753 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
6754 * 6. WebAuthn Authenticator model (2) (3) (4)
6755 * 6.1. Authenticator data (2)
6756 * 7.1. Registering a new credential
6757 * 7.2. Verifying an authentication assertion
6758 * 9. WebAuthn Extensions
6759 * 9.4. Client extension processing
6760 * 9.6. Example Extension
6761
6762 #dom-collectedclientdata-typeReferenced in:
6763 * 5.1.3. Create a new credential - PublicKeyCredential's
6764 [[Create]](origin, options, sameOriginWithAncestors) method
6765 * 5.1.4.1. PublicKeyCredential's
6766 [[DiscoverFromExternalSource]](origin, options,
6767 sameOriginWithAncestors) method
6768 * 5.8.1. Client data used in WebAuthn signatures (dictionary
6769 CollectedClientData)
6770 * 7.1. Registering a new credential
6771 * 7.2. Verifying an authentication assertion
6772
6773 #dom-collectedclientdata-challengeReferenced in:
6774 * 5.1.3. Create a new credential - PublicKeyCredential's
6775 [[Create]](origin, options, sameOriginWithAncestors) method
6776 * 5.1.4.1. PublicKeyCredential's
6777 [[DiscoverFromExternalSource]](origin, options,
6778 sameOriginWithAncestors) method
6779 * 5.8.1. Client data used in WebAuthn signatures (dictionary
6780 CollectedClientData)
6781 * 7.1. Registering a new credential
6782 * 7.2. Verifying an authentication assertion
6783
6784 #dom-collectedclientdata-originReferenced in:
6785 * 5.1.3. Create a new credential - PublicKeyCredential's
6786 [[Create]](origin, options, sameOriginWithAncestors) method
6787 * 5.1.4.1. PublicKeyCredential's
6788 [[DiscoverFromExternalSource]](origin, options,
6789 sameOriginWithAncestors) method
6790 * 5.8.1. Client data used in WebAuthn signatures (dictionary
6791 CollectedClientData)
6792 * 7.1. Registering a new credential
6793 * 7.2. Verifying an authentication assertion
6794
6795 #dom-collectedclientdata-hashalgorithmReferenced in:
6796 * 5.1.3. Create a new credential - PublicKeyCredential's
6797 [[Create]](origin, options, sameOriginWithAncestors) method
6798 * 5.1.4.1. PublicKeyCredential's
6799 [[DiscoverFromExternalSource]](origin, options,
6800 sameOriginWithAncestors) method
6801 * 5.8.1. Client data used in WebAuthn signatures (dictionary
6802 CollectedClientData) (2)
6803 * 7.1. Registering a new credential
6804 * 7.2. Verifying an authentication assertion
6805
6806 #dom-collectedclientdata-tokenbindingidReferenced in:
6807 * 5.1.3. Create a new credential - PublicKeyCredential's
6808 [[Create]](origin, options, sameOriginWithAncestors) method


```
5529 * 4.1.4. Use an existing credential to make an assertion -
5530 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5531 method
5532 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5533 CollectedClientData)
5534 * 6.1. Registering a new credential
5535 * 6.2. Verifying an authentication assertion
5536
5537 #dom-collectedclientdata-clientextensionsReferenced in:
5538 * 4.1.3. Create a new credential - PublicKeyCredential's
5539 [[Create]](options) method
5540 * 4.1.4. Use an existing credential to make an assertion -
5541 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5542 method
5543 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5544 CollectedClientData)
5545 * 6.1. Registering a new credential
5546 * 6.2. Verifying an authentication assertion
5547 * 8.4. Client extension processing
5548
5549 #dom-collectedclientdata-authenticatorextensionsReferenced in:
5550 * 4.1.3. Create a new credential - PublicKeyCredential's
5551 [[Create]](options) method
5552 * 4.1.4. Use an existing credential to make an assertion -
5553 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5554 method
5555 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5556 CollectedClientData)
5557 * 6.1. Registering a new credential
5558 * 6.2. Verifying an authentication assertion
5559
5560 #collectedclientdata-json-serialized-client-dataReferenced in:
5561 * 4.1.3. Create a new credential - PublicKeyCredential's
5562 [[Create]](options) method
5563 * 4.1.4. Use an existing credential to make an assertion -
5564 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5565 method
5566 * 4.2. Authenticator Responses (interface AuthenticatorResponse)
5567 * 4.2.1. Information about Public Key Credential (interface
5568 AuthenticatorAttestationResponse) (2)
5569 * 4.2.2. Web Authentication Assertion (interface
5570 AuthenticatorAssertionResponse)
5571 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5572 CollectedClientData)
5573
5574 #collectedclientdata-hash-of-the-serialized-client-dataReferenced in:
5575 * 4.1.3. Create a new credential - PublicKeyCredential's
5576 [[Create]](options) method (2)
5577 * 4.1.4. Use an existing credential to make an assertion -
5578 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5579 method (2)
5580 * 4.2.1. Information about Public Key Credential (interface
5581 AuthenticatorAttestationResponse)
5582 * 4.2.2. Web Authentication Assertion (interface
5583 AuthenticatorAssertionResponse)
5584 * 4.7.1. Client data used in WebAuthn signatures (dictionary
5585 CollectedClientData)
5586 * 5. WebAuthn Authenticator model
5587 * 5.2.1. The authenticatorMakeCredential operation (2)
5588 * 5.2.2. The authenticatorGetAssertion operation (2) (3)
5589 * 5.3.2. Attestation Statement Formats (2)
5590 * 5.3.4. Generating an Attestation Object
5591 * 6.1. Registering a new credential
5592 * 7.2. Packed Attestation Statement Format (2)
5593 * 7.3. TPM Attestation Statement Format (2)
5594 * 7.4. Android Key Attestation Statement Format (2)
5595 * 7.5. Android SafetyNet Attestation Statement Format
5596 * 7.6. FIDO U2F Attestation Statement Format (2)
5597
5598 #enumdef-publickeycredentialtypeReferenced in:
```

```
6809 * 5.1.4.1. PublicKeyCredential's
6810 [[DiscoverFromExternalSource]](origin, options,
6811 sameOriginWithAncestors) method
6812 * 5.8.1. Client data used in WebAuthn signatures (dictionary
6813 CollectedClientData)
6814 * 7.1. Registering a new credential
6815 * 7.2. Verifying an authentication assertion
6816
6817 #dom-collectedclientdata-clientextensionsReferenced in:
6818 * 5.1.3. Create a new credential - PublicKeyCredential's
6819 [[Create]](origin, options, sameOriginWithAncestors) method
6820 * 5.1.4.1. PublicKeyCredential's
6821 [[DiscoverFromExternalSource]](origin, options,
6822 sameOriginWithAncestors) method
6823 * 5.8.1. Client data used in WebAuthn signatures (dictionary
6824 CollectedClientData)
6825 * 7.1. Registering a new credential
6826 * 7.2. Verifying an authentication assertion
6827 * 9.4. Client extension processing
6828
6829 #dom-collectedclientdata-authenticatorextensionsReferenced in:
6830 * 5.1.3. Create a new credential - PublicKeyCredential's
6831 [[Create]](origin, options, sameOriginWithAncestors) method
6832 * 5.1.4.1. PublicKeyCredential's
6833 [[DiscoverFromExternalSource]](origin, options,
6834 sameOriginWithAncestors) method
6835 * 5.8.1. Client data used in WebAuthn signatures (dictionary
6836 CollectedClientData)
6837 * 7.1. Registering a new credential
6838 * 7.2. Verifying an authentication assertion
6839
6840 #collectedclientdata-json-serialized-client-dataReferenced in:
6841 * 5.1.3. Create a new credential - PublicKeyCredential's
6842 [[Create]](origin, options, sameOriginWithAncestors) method
6843 * 5.1.4.1. PublicKeyCredential's
6844 [[DiscoverFromExternalSource]](origin, options,
6845 sameOriginWithAncestors) method
6846 * 5.2. Authenticator Responses (interface AuthenticatorResponse)
6847 * 5.2.1. Information about Public Key Credential (interface
6848 AuthenticatorAttestationResponse) (2)
6849 * 5.2.2. Web Authentication Assertion (interface
6850 AuthenticatorAssertionResponse)
6851 * 5.8.1. Client data used in WebAuthn signatures (dictionary
6852 CollectedClientData)
6853
6854 #collectedclientdata-hash-of-the-serialized-client-dataReferenced in:
6855 * 5.1.3. Create a new credential - PublicKeyCredential's
6856 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6857 * 5.1.4.1. PublicKeyCredential's
6858 [[DiscoverFromExternalSource]](origin, options,
6859 sameOriginWithAncestors) method (2)
6860 * 5.2.1. Information about Public Key Credential (interface
6861 AuthenticatorAttestationResponse)
6862 * 5.2.2. Web Authentication Assertion (interface
6863 AuthenticatorAssertionResponse)
6864 * 5.8.1. Client data used in WebAuthn signatures (dictionary
6865 CollectedClientData)
6866 * 6. WebAuthn Authenticator model
6867 * 6.2.1. The authenticatorMakeCredential operation
6868 * 6.2.2. The authenticatorGetAssertion operation (2)
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:
```



```
5599 * 4.1.3. Create a new credential - PublicKeyCredential's
5600 [[Create]](options) method (2)
5601 * 4.3. Parameters for Credential Generation (dictionary
5602 PublicKeyCredentialParameters)
5603 * 4.7.2. Credential Type enumeration (enum PublicKeyCredentialType)
5604 * 4.7.3. Credential Descriptor (dictionary
5605 PublicKeyCredentialDescriptor)
5606 * 5.2.1. The authenticatorMakeCredential operation (2) (3)
5607
5608 #dom-publickeycredentialtype-public-keyReferenced in:
5609 * 4.7.2. Credential Type enumeration (enum PublicKeyCredentialType)
5610
5611 #dictdef-publickeycredentialdescriptorReferenced in:
5612 * 4.4. Options for Credential Creation (dictionary
5613
5614 MakePublicKeyCredentialOptions) (2)
5615 * 4.5. Options for Assertion Generation (dictionary
5616 PublicKeyCredentialRequestOptions) (2) (3)
5617 * 4.7.3. Credential Descriptor (dictionary
5618 PublicKeyCredentialDescriptor)
5619 * 5.2.1. The authenticatorMakeCredential operation
5620
5621 #dom-publickeycredentialdescriptor-transportsReferenced in:
5622 * 4.1.3. Create a new credential - PublicKeyCredential's
5623 [[Create]](options) method (2)
5624 * 4.1.4. Use an existing credential to make an assertion -
5625 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5626 method (2)
5627
5628 #dom-publickeycredentialdescriptor-typeReferenced in:
5629 * 4.1.4. Use an existing credential to make an assertion -
5630 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5631 method
5632 * 4.7.3. Credential Descriptor (dictionary
5633 PublicKeyCredentialDescriptor)
5634
5635 #dom-publickeycredentialdescriptor-idReferenced in:
5636 * 4.1.4. Use an existing credential to make an assertion -
5637 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5638 method
5639 * 4.7.3. Credential Descriptor (dictionary
5640 PublicKeyCredentialDescriptor)
5641
5642 #enumdef-authenticatortransportReferenced in:
5643 * 4.7.3. Credential Descriptor (dictionary
5644 PublicKeyCredentialDescriptor)
5645 * 4.7.4. Authenticator Transport enumeration (enum
5646 AuthenticatorTransport)
5647
5648 #dom-authenticatortransport-usbReferenced in:
5649 * 4.7.4. Authenticator Transport enumeration (enum
5650 AuthenticatorTransport)
5651
5652 #dom-authenticatortransport-nfcReferenced in:
5653 * 4.7.4. Authenticator Transport enumeration (enum
5654 AuthenticatorTransport)
5655
5656 #dom-authenticatortransport-bleReferenced in:
5657 * 4.7.4. Authenticator Transport enumeration (enum
5658 AuthenticatorTransport)
5659
5660 #typedefdef-cosealgorithmIdentifierReferenced in:
5661 * 4.1.3. Create a new credential - PublicKeyCredential's
```

```
6879 * 5.1.3. Create a new credential - PublicKeyCredential's
6880 [[Create]](origin, options, sameOriginWithAncestors) method (2)
6881 * 5.3. Parameters for Credential Generation (dictionary
6882 PublicKeyCredentialParameters)
6883 * 5.8.2. Credential Type enumeration (enum PublicKeyCredentialType)
6884 * 5.8.3. 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-transportsReferenced 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
```

5661 [[Create]](**options**) method
5662 * 4.3. Parameters for Credential Generation (dictionary
5663 PublicKeyCredentialParameters)
5664 * 4.7.5. Cryptographic Algorithm Identifier (typedef
5665 COSEAlgorithmIdentifier)
5666 * 5.2.1. The authenticatorMakeCredential operation
5667 * 5.3.1. Attest**ation** data

#attestation-signatureReferenced in:
* 3. Terminology
* 5. WebAuthn Authenticator model (2) (3)
* 5.3. Attestation
* 7.6. FIDO U2F Attestation Statement Format

#assertion-signatureReferenced in:
* 5. WebAuthn Authenticator model (2)
* 5.2.2. The authenticatorGetAssertion operation (2) (3) (4) (5) (6)

#authenticator-dataReferenced in:
* 4.2.1. Information about Public Key Credential (interface
AuthenticatorAttestationResponse) (2)
* 4.2.2. Web Authentication Assertion (interface
AuthenticatorAssertionResponse)
* 5. WebAuthn Authenticator model (2)
* 5.1. Authenticator data (2) (3) (4) (5) (6) (7) (8)
* 5.2.1. The authenticatorMakeCredential operation (2)
* 5.2.2. The authenticatorGetAssertion operation (2) (3) (4)
* 5.3. Attestation (2)
* 5.3.1. Attestation data

6949 [[Create]](**origin, options, sameOriginWithAncestors**) method
6950 * 5.3. Parameters for Credential Generation (dictionary
6951 PublicKeyCredentialParameters)
6952 * 5.8.5. Cryptographic Algorithm Identifier (typedef
6953 COSEAlgorithmIdentifier)
6954 * 6.2.1. The authenticatorMakeCredential operation
6955 * 6.3.1. Attested credential data
6956 * 8.2. Packed Attestation Statement Format
6957 * 8.3. TPM Attestation Statement Format

#enumdef-userverificationrequirementReferenced in:
* 5.4.4. Authenticator Selection Criteria (dictionary
AuthenticatorSelectionCriteria) (2)
* 5.5. Options for Assertion Generation (dictionary
PublicKeyCredentialRequestOptions) (2)
* 5.8.6. User Verification Requirement enumeration (enum
UserVerificationRequirement)

#dom-userverificationrequirement-requiredReferenced in:
* 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)
* 5.8.6. User Verification Requirement enumeration (enum
UserVerificationRequirement)

#dom-userverificationrequirement-preferredReferenced in:
* 5.1.3. Create a new credential - PublicKeyCredential's
[[Create]](**origin, options, sameOriginWithAncestors**) method
* 5.1.4.1. PublicKeyCredential's
[[DiscoverFromExternalSource]](**origin, options,**
sameOriginWithAncestors) method
* 5.8.6. User Verification Requirement enumeration (enum
UserVerificationRequirement)

#dom-userverificationrequirement-discouragedReferenced in:
* 5.1.3. Create a new credential - PublicKeyCredential's
[[Create]](**origin, options, sameOriginWithAncestors**) method
* 5.1.4.1. PublicKeyCredential's
[[DiscoverFromExternalSource]](**origin, options,**
sameOriginWithAncestors) method
* 5.8.6. User Verification Requirement enumeration (enum
UserVerificationRequirement)

#attestation-signatureReferenced in:
* 4. Terminology
* 6. WebAuthn Authenticator model (2) (3)
* 6.3. Attestation
* 7.1. Registering a new credential
* 8.6. FIDO U2F Attestation Statement Format

#assertion-signatureReferenced in:
* 6. WebAuthn Authenticator model (2)
* 6.2.2. The authenticatorGetAssertion operation (2) (3)

#authenticator-dataReferenced in:
* 5.1.4.1. PublicKeyCredential's
[[DiscoverFromExternalSource]](**origin, options,**
sameOriginWithAncestors) method
* 5.2.1. Information about Public Key Credential (interface
AuthenticatorAttestationResponse) (2)
* 5.2.2. Web Authentication Assertion (interface
AuthenticatorAssertionResponse)
* 6. WebAuthn Authenticator model (2)
* 6.1. Authenticator data (2) (3) (4) (5) (6) (7) (8) (9)
* 6.1.1. Signature Counter Considerations (2)
* 6.2.1. The authenticatorMakeCredential operation
* 6.2.2. The authenticatorGetAssertion operation
* 6.3. Attestation (2)

5690	* 5.3.2. Attestation Statement Formats (2)
5691	* 5.3.4. Generating an Attestation Object (2) (3)
5692	* 5.3.5.3. Attestation Certificate Hierarchy
5693	* 6.1. Registering a new credential (2)
5694	* 7.5. Android SafetyNet Attestation Statement Format
5695	* 8.5. Authenticator extension processing (2)
5696	* 8.6. Example Extension (2)
5697	* 9.6. User Verification Index Extension (uvi)
5698	* 9.7. Location Extension (loc)
5699	* 9.8. User Verification Method Extension (uvm)
5700	
5701	#authenticatormakecredentialReferenced in:
5702	* 3. Terminology (2) (3)
5703	* 4.1.3. Create a new credential - PublicKeyCredential's
5704	[[Create]](options) method (2)
5705	* 5. WebAuthn Authenticator model
5706	* 5.2.3. The authenticatorCancel operation (2)
5707	* 8. WebAuthn Extensions
5708	* 8.2. Defining extensions
5709	
5710	#authenticatorgetassertionReferenced in:
5711	* 3. Terminology (2) (3)
5712	* 4.1.4. Use an existing credential to make an assertion -
5713	PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5714	method (2) (3) (4)
5715	* 5. WebAuthn Authenticator model
5716	* 5.1. Authenticator data

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)
7072	
7073	#authenticatormakecredentialReferenced in:
7074	* 4. Terminology (2) (3) (4)
7075	* 5.1.3. Create a new credential - PublicKeyCredential's
7076	[[Create]](origin, options, sameOriginWithAncestors) method (2)
7077	* 6. WebAuthn Authenticator model
7078	* 6.2.3. The authenticatorCancel operation (2)
7079	* 9. WebAuthn Extensions
7080	* 9.2. Defining extensions
7081	
7082	#authenticatorgetassertionReferenced in:
7083	* 4. Terminology (2) (3)
7084	* 5.1.4.1. PublicKeyCredential's
7085	[[DiscoverFromExternalSource]](origin, options,
7086	sameOriginWithAncestors) method (2) (3) (4)
7087	* 6. WebAuthn Authenticator model
7088	* 6.1. Authenticator data

5717	* 5.2.3. The authenticatorCancel operation (2)
5718	* 8. WebAuthn Extensions
5719	* 8.2. Defining extensions
5720	
5721	#authenticatorcancelReferenced in:
5722	* 4.1.3. Create a new credential - PublicKeyCredential's
5723	[[Create]](options) method (2) (3)
5724	* 4.1.4. Use an existing credential to make an assertion -
5725	PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5726	method (2) (3)
5727	
5728	#attestation-objectReferenced in:
5729	* 3. Terminology
5730	* 4. Web Authentication API
5731	* 4.2.1. Information about Public Key Credential (interface
5732	AuthenticatorAttestationResponse) (2)
5733	* 4.4. Options for Credential Creation (dictionary
5734	MakePublicKeyCredentialOptions) (2)
5735	* 5.2.1. The authenticatorMakeCredential operation (2)
5736	* 5.3. Attestation (2) (3)
5737	* 5.3.1. Attestation data
5738	* 5.3.4. Generating an Attestation Object (2) (3) (4)
5739	* 6.1. Registering a new credential
5740	
5741	#attestation-statementReferenced in:
5742	* 3. Terminology
5743	* 4.2.1. Information about Public Key Credential (interface
5744	AuthenticatorAttestationResponse) (2) (3)
5745	* 5.3. Attestation (2) (3) (4) (5) (6) (7) (8)
5746	* 5.3.2. Attestation Statement Formats (2) (3)
5747	
5748	#attestation-statement-formatReferenced in:
5749	* 4.2.1. Information about Public Key Credential (interface
5750	AuthenticatorAttestationResponse)
5751	* 4.7.4. Authenticator Transport enumeration (enum
5752	AuthenticatorTransport)
5753	* 5.3. Attestation (2) (3) (4) (5) (6) (7)
5754	* 5.3.2. Attestation Statement Formats (2) (3) (4)
5755	* 5.3.4. Generating an Attestation Object (2)
5756	
5757	#attestation-typeReferenced in:
5758	* 5.3. Attestation (2) (3) (4) (5) (6)
5759	* 5.3.2. Attestation Statement Formats
5760	
5761	#attestation-dataReferenced in:
5762	* 5.1. Authenticator data (2) (3) (4) (5) (6) (7)
5763	* 5.2.1. The authenticatorMakeCredential operation
5764	* 5.2.2. The authenticatorGetAssertion operation
5765	* 5.3. Attestation (2)
5766	* 5.3.3. Attestation Types
5767	* 6.1. Registering a new credential (2)
5768	* 7.3. TPM Attestation Statement Format
5769	* 7.4. Android Key Attestation Statement Format

7089	* 6.1.1. Signature Counter Considerations (2) (3)
7090	* 6.2.3. The authenticatorCancel operation (2)
7091	* 9. WebAuthn Extensions
7092	* 9.2. Defining extensions
7093	
7094	#authenticatorcancelReferenced in:
7095	* 5.1.3. Create a new credential - PublicKeyCredential's
7096	[[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
7097	(4)
7098	* 5.1.4.1. PublicKeyCredential's
7099	[[DiscoverFromExternalSource]](origin, options,
7100	sameOriginWithAncestors) method (2) (3) (4)
7101	* 6.2.1. The authenticatorMakeCredential operation
7102	* 6.2.2. The authenticatorGetAssertion operation
7103	
7104	#attestation-objectReferenced in:
7105	* 4. Terminology (2) (3)
7106	* 5. Web Authentication API
7107	* 5.2.1. Information about Public Key Credential (interface
7108	AuthenticatorAttestationResponse) (2)
7109	* 5.4. Options for Credential Creation (dictionary
7110	MakePublicKeyCredentialOptions) (2)
7111	* 6.2.1. The authenticatorMakeCredential operation (2)
7112	* 6.3. Attestation (2) (3)
7113	* 6.3.1. Attested credential data
7114	* 6.3.4. Generating an Attestation Object (2)
7115	* 7.1. Registering a new credential
7116	
7117	#attestation-statementReferenced in:
7118	* 4. Terminology (2)
7119	* 5.1.3. Create a new credential - PublicKeyCredential's
7120	[[Create]](origin, options, sameOriginWithAncestors) method (2) (3)
7121	* 5.2.1. Information about Public Key Credential (interface
7122	AuthenticatorAttestationResponse) (2) (3)
7123	* 5.4.6. Attestation Conveyance Preference enumeration (enum
7124	AttestationConveyancePreference) (2) (3) (4) (5) (6) (7)
7125	* 6.3. Attestation (2) (3) (4) (5) (6) (7) (8)
7126	* 6.3.2. Attestation Statement Formats (2) (3) (4)
7127	* 7.1. Registering a new credential
7128	
7129	#attestation-statement-formatReferenced in:
7130	* 5.2.1. Information about Public Key Credential (interface
7131	AuthenticatorAttestationResponse)
7132	* 5.8.4. Authenticator Transport enumeration (enum
7133	AuthenticatorTransport)
7134	* 6.2.1. The authenticatorMakeCredential operation
7135	* 6.3. Attestation (2) (3) (4) (5) (6) (7)
7136	* 6.3.2. Attestation Statement Formats (2) (3) (4)
7137	* 6.3.4. Generating an Attestation Object
7138	* 7.1. Registering a new credential (2)
7139	
7140	#attestation-typeReferenced in:
7141	* 5.1.3. Create a new credential - PublicKeyCredential's
7142	[[Create]](origin, options, sameOriginWithAncestors) method
7143	* 6.3. Attestation (2) (3) (4) (5) (6)
7144	* 6.3.2. Attestation Statement Formats (2)
7145	
7146	#attested-credential-dataReferenced in:
7147	* 5.1.3. Create a new credential - PublicKeyCredential's
7148	[[Create]](origin, options, sameOriginWithAncestors) method
7149	* 6.1. Authenticator data (2) (3) (4) (5)
7150	* 6.2.1. The authenticatorMakeCredential operation
7151	* 6.3. Attestation (2)
7152	* 6.3.1. Attested credential data
7153	* 6.3.3. Attestation Types
7154	
7155	#aaguidReferenced in:
7156	* 4. Terminology
7157	* 5.1.3. Create a new credential - PublicKeyCredential's
7158	[[Create]](origin, options, sameOriginWithAncestors) method (2) (3)

5770
5771
5772
5773
5774
5775
5776
5777
5778
5779
5780
5781
5782
5783
5784
5785
5786
5787
5788
5789
5790
5791
5792
5793
5794
5795
5796
5797
5798
5799
5800

#signing-procedureReferenced in:
* 5.3.2. Attestation Statement Formats

#authenticator-data-for-the-attestationReferenced in:
* 7.2. Packed Attestation Statement Format
* 7.3. TPM Attestation Statement Format
* 7.4. Android Key Attestation Statement Format (2)
* 7.5. Android SafetyNet Attestation Statement Format
* 7.6. FIDO U2F Attestation Statement Format

#authenticator-data-claimed-to-have-been-used-for-the-attestationReferenced in:
* 7.2. Packed Attestation Statement Format
* 7.3. TPM Attestation Statement Format
* 7.4. Android Key Attestation Statement Format (2)
* 7.6. FIDO U2F Attestation Statement Format

#basic-attestationReferenced in:
* 5.3.5.1. Privacy

#self-attestationReferenced in:
* 3. Terminology (2) (3) (4)
* 5.3. Attestation (2)
* 5.3.2. Attestation Statement Formats
* 5.3.3. Attestation Types
* 5.3.5.2. Attestation Certificate and Attestation Certificate CA Compromise
* 6.1. Registering a new credential (2) (3)
* 7.2. Packed Attestation Statement Format (2)
* 7.6. FIDO U2F Attestation Statement Format

7159
7160
7161
7162
7163
7164
7165
7166
7167
7168
7169
7170
7171
7172
7173
7174
7175
7176
7177
7178
7179
7180
7181
7182
7183
7184
7185
7186
7187
7188
7189
7190
7191
7192
7193
7194
7195
7196
7197
7198
7199
7200
7201
7202
7203
7204
7205
7206
7207
7208
7209
7210
7211
7212
7213
7214
7215
7216
7217
7218
7219
7220
7221
7222
7223
7224
7225
7226
7227
7228

* 5.4.6. Attestation Conveyance Preference enumeration (enum AttestationConveyancePreference)
* 7.1. Registering a new credential
* 8.2. Packed Attestation Statement Format
* 8.3. TPM Attestation Statement Format

#credentialidlengthReferenced in:
* 6.1. Authenticator data

#credentialidReferenced in:
* 5.1.3. Create a new credential - PublicKeyCredential's [[Create]](origin, options, sameOriginWithAncestors) method
* 6.1. Authenticator data
* 7.1. Registering a new credential

#credentialpublickeyReferenced in:
* 6.1. Authenticator data
* 7.1. Registering a new credential
* 8.2. Packed Attestation Statement Format
* 8.3. TPM Attestation Statement Format
* 8.4. Android Key Attestation Statement Format

#signing-procedureReferenced in:
* 6.3.2. Attestation Statement Formats
* 6.3.4. Generating an Attestation Object

#authenticator-data-for-the-attestationReferenced in:
* 8.2. Packed Attestation Statement Format
* 8.3. TPM Attestation Statement Format
* 8.4. Android Key Attestation Statement Format (2)
* 8.5. Android SafetyNet Attestation Statement Format
* 8.6. FIDO U2F Attestation Statement Format

#verification-procedure-inputsReferenced in:
* 8.2. Packed Attestation Statement Format
* 8.3. TPM Attestation Statement Format
* 8.4. Android Key Attestation Statement Format
* 8.5. Android SafetyNet Attestation Statement Format
* 8.6. FIDO U2F Attestation Statement Format

#authenticator-data-claimed-to-have-been-used-for-the-attestationReferenced in:
* 8.4. Android Key Attestation Statement Format

#attestation-trust-pathReferenced in:
* 6.3.2. Attestation Statement Formats
* 8.2. Packed Attestation Statement Format (2) (3)
* 8.3. TPM Attestation Statement Format
* 8.4. Android Key Attestation Statement Format
* 8.5. Android SafetyNet Attestation Statement Format
* 8.6. FIDO U2F Attestation Statement Format

#basic-attestationReferenced in:
* 6.3.5.1. Privacy
* 8.4. Android Key Attestation Statement Format
* 8.5. Android SafetyNet Attestation Statement Format
* 8.6. FIDO U2F Attestation Statement Format

#self-attestationReferenced in:
* 4. Terminology (2) (3) (4)
* 5.4.6. Attestation Conveyance Preference enumeration (enum AttestationConveyancePreference)
* 6.3. Attestation (2)
* 6.3.2. Attestation Statement Formats
* 6.3.3. Attestation Types
* 6.3.5.2. Attestation Certificate and Attestation Certificate CA Compromise
* 7.1. Registering a new credential (2) (3)
* 8.2. Packed Attestation Statement Format (2)
* 8.6. FIDO U2F Attestation Statement Format

5801 #privacy-caReferenced in:
5802 * 5.3.5.1. Privacy
5803

5804 #elliptic-curve-based-direct-anonymous-attestationReferenced in:
5805 * 5.3.5.1. Privacy
5806

5807 #ecdaaReferenced in:
5808 * 5.3.2. Attestation Statement Formats
5809 * 5.3.3. Attestation Types
5810 * 5.3.5.2. Attestation Certificate and Attestation Certificate CA
5811 Compromise
5812 * 6.1. Registering a new credential
5813 * 7.2. Packed Attestation Statement Format (2)
5814 * 7.3. TPM Attestation Statement Format (2)
5815

5816 #attestation-statement-format-identifierReferenced in:
5817 * 5.3.2. Attestation Statement Formats
5818 * 5.3.4. Generating an Attestation Object
5819

5820 #identifier-of-the-ecdaa-issuer-public-keyReferenced in:
5821 * 6.1. Registering a new credential
5822 * 7.2. Packed Attestation Statement Format
5823 * 7.3. TPM Attestation Statement Format (2)
5824

5825 #ecdaa-issuer-public-keyReferenced in:
5826 * 5.3.2. Attestation Statement Formats
5827 * 5.3.5.1. Privacy
5828 * 6.1. Registering a new credential
5829 * 7.2. Packed Attestation Statement Format (2) (3)
5830

5831 #registration-extensionReferenced in:
5832 * 4.1.3. Create a new credential - PublicKeyCredential's
5833 [[Create]](options) method
5834 * 8. WebAuthn Extensions (2) (3) (4) (5) (6)
5835 * 8.6. Example Extension
5836 * 9.2. Simple Transaction Authorization Extension (txAuthSimple)
5837 * 9.3. Generic Transaction Authorization Extension (txAuthGeneric)
5838 * 9.4. Authenticator Selection Extension (authnSel)
5839 * 9.5. Supported Extensions Extension (exts)
5840 * 9.6. User Verification Index Extension (uvi)
5841 * 9.7. Location Extension (loc)
5842 * 9.8. User Verification Method Extension (uvm)
5843 * 10.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
5844 (6) (7)
5845

5846 #authentication-extensionReferenced in:
5847 * 4.1.4. Use an existing credential to make an assertion -
5848 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5849 method
5850 * 8. WebAuthn Extensions (2) (3) (4) (5) (6)
5851 * 8.6. Example Extension
5852 * 9.1. FIDO Appid Extension (appid)
5853 * 9.2. Simple Transaction Authorization Extension (txAuthSimple)
5854 * 9.3. Generic Transaction Authorization Extension (txAuthGeneric)
5855 * 9.6. User Verification Index Extension (uvi)
5856 * 9.7. Location Extension (loc)
5857 * 9.8. User Verification Method Extension (uvm)
5858 * 10.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
5859 (6)
5860

5861 #client-extensionReferenced in:
5862 * 4.1.3. Create a new credential - PublicKeyCredential's
5863 [[Create]](options) method
5864

7229 #privacy-caReferenced in:
7230 * 5.1.3. Create a new credential - PublicKeyCredential's
7231 [[Create]](origin, options, sameOriginWithAncestors) method
7232 * 5.4.6. Attestation Conveyance Preference enumeration (enum
7233 AttestationConveyancePreference)
7234 * 6.3.5.1. Privacy
7235 * 8.3. TPM Attestation Statement Format
7236 * 8.6. FIDO U2F Attestation Statement Format
7237

7238 #elliptic-curve-based-direct-anonymous-attestationReferenced in:
7239 * 6.3.5.1. Privacy
7240

7241 #ecdaaReferenced in:
7242 * 6.3.2. Attestation Statement Formats
7243 * 6.3.3. Attestation Types
7244 * 6.3.5.2. Attestation Certificate and Attestation Certificate CA
7245 Compromise
7246 * 7.1. Registering a new credential
7247 * 8.2. Packed Attestation Statement Format (2)
7248 * 8.3. TPM Attestation Statement Format (2) (3)
7249

7250 #attestation-statement-format-identifierReferenced in:
7251 * 6.3.2. Attestation Statement Formats
7252 * 6.3.4. Generating an Attestation Object
7253

7254 #identifier-of-the-ecdaa-issuer-public-keyReferenced in:
7255 * 7.1. Registering a new credential
7256 * 8.2. Packed Attestation Statement Format
7257 * 8.3. TPM Attestation Statement Format (2)
7258

7259 #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 * 11.2. WebAuthn Extension Identifier Registrations (2) (3) (4) (5)
7278 (6) (7)
7279

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

```
5865 * 4.1.4. Use an existing credential to make an assertion -
5866 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5867 method
5868 * 4.6. Authentication Extensions (typedef AuthenticationExtensions)
5869 * 8. WebAuthn Extensions
5870 * 8.2. Defining extensions
5871 * 8.4. Client extension processing
5872
5873 #authenticator-extensionReferenced in:
5874 * 4.1.3. Create a new credential - PublicKeyCredential's
5875 [[Create]](options) method
5876 * 4.1.4. Use an existing credential to make an assertion -
5877 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5878 method
5879 * 4.6. Authentication Extensions (typedef AuthenticationExtensions)
5880 * 8. WebAuthn Extensions (2) (3)
5881 * 8.2. Defining extensions (2)
5882 * 8.3. Extending request parameters
5883 * 8.5. Authenticator extension processing
5884
5885 #extension-identifierReferenced in:
5886 * 4.1. PublicKeyCredential Interface
5887 * 4.1.3. Create a new credential - PublicKeyCredential's
5888 [[Create]](options) method
5889 * 4.1.4. Use an existing credential to make an assertion -
5890 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5891 method
5892 * 5.1. Authenticator data
5893 * 8. WebAuthn Extensions (2)
5894 * 8.2. Defining extensions
5895 * 8.3. Extending request parameters
5896 * 8.4. Client extension processing (2)
5897 * 8.5. Authenticator extension processing (2)
5898 * 8.6. Example Extension
5899 * 9.5. Supported Extensions Extension (exts) (2)
5900 * 9.7. Location Extension (loc)
5901 * 10.2. WebAuthn Extension Identifier Registrations
5902
5903 #client-extension-inputReferenced in:
5904 * 8. WebAuthn Extensions (2) (3)
5905 * 8.2. Defining extensions
5906 * 8.3. Extending request parameters (2) (3) (4) (5) (6)
5907 * 8.4. Client extension processing (2) (3) (4)
5908 * 8.6. Example Extension
5909
5910 #authenticator-extension-inputReferenced in:
5911 * 8. WebAuthn Extensions (2) (3) (4) (5)
5912 * 8.2. Defining extensions
5913 * 8.3. Extending request parameters (2) (3)
5914 * 8.4. Client extension processing
5915 * 8.5. Authenticator extension processing (2) (3)
5916
5917 #client-extension-processingReferenced in:
5918 * 4.1. PublicKeyCredential Interface
5919 * 4.1.3. Create a new credential - PublicKeyCredential's
5920 [[Create]](options) method (2)
5921 * 4.1.4. Use an existing credential to make an assertion -
5922 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5923 method (2)
5924 * 8. WebAuthn Extensions (2) (3) (4)
5925 * 8.2. Defining extensions
5926
5927 #client-extension-outputReferenced in:
5928 * 4.1. PublicKeyCredential Interface
5929 * 4.1.3. Create a new credential - PublicKeyCredential's
5930 [[Create]](options) method (2)
```

```
7299 * 5.1.4.1. PublicKeyCredential's
7300 [[DiscoverFromExternalSource]](origin, options,
7301 sameOriginWithAncestors) method
7302 * 5.7. Authentication Extensions (typedef AuthenticationExtensions)
7303 * 9. WebAuthn Extensions
7304 * 9.2. Defining extensions
7305 * 9.4. Client extension processing
7306
7307 #authenticator-extensionReferenced in:
7308 * 5.1.3. Create a new credential - PublicKeyCredential's
7309 [[Create]](origin, options, sameOriginWithAncestors) method
7310 * 5.1.4.1. PublicKeyCredential's
7311 [[DiscoverFromExternalSource]](origin, options,
7312 sameOriginWithAncestors) method
7313 * 5.7. Authentication Extensions (typedef AuthenticationExtensions)
7314 * 9. WebAuthn Extensions (2) (3)
7315 * 9.2. Defining extensions (2)
7316 * 9.3. Extending request parameters
7317 * 9.5. Authenticator extension processing
7318
7319 #extension-identifierReferenced in:
7320 * 5.1. PublicKeyCredential Interface
7321 * 5.1.3. Create a new credential - PublicKeyCredential's
7322 [[Create]](origin, options, sameOriginWithAncestors) method
7323 * 5.1.4.1. PublicKeyCredential's
7324 [[DiscoverFromExternalSource]](origin, options,
7325 sameOriginWithAncestors) method
7326 * 6.1. Authenticator data
7327 * 6.2.1. The authenticatorMakeCredential operation (2)
7328 * 6.2.2. The authenticatorGetAssertion operation (2)
7329 * 9. WebAuthn Extensions (2)
7330 * 9.2. Defining extensions
7331 * 9.3. Extending request parameters
7332 * 9.4. Client extension processing (2)
7333 * 9.5. Authenticator extension processing (2)
7334 * 9.6. Example Extension
7335 * 10.5. Supported Extensions Extension (exts) (2)
7336 * 10.7. Location Extension (loc)
7337 * 11.2. WebAuthn Extension Identifier Registrations
7338
7339 #client-extension-inputReferenced in:
7340 * 9. WebAuthn Extensions (2) (3)
7341 * 9.2. Defining extensions
7342 * 9.3. Extending request parameters (2) (3) (4) (5) (6)
7343 * 9.4. Client extension processing (2) (3) (4)
7344 * 9.6. Example Extension
7345
7346 #authenticator-extension-inputReferenced in:
7347 * 6.2.1. The authenticatorMakeCredential operation
7348 * 6.2.2. The authenticatorGetAssertion operation
7349 * 9. WebAuthn Extensions (2) (3) (4) (5)
7350 * 9.2. Defining extensions
7351 * 9.3. Extending request parameters (2) (3)
7352 * 9.4. Client extension processing
7353 * 9.5. Authenticator extension processing (2) (3)
7354
7355 #client-extension-processingReferenced in:
7356 * 5.1. PublicKeyCredential Interface
7357 * 5.1.3. Create a new credential - PublicKeyCredential's
7358 [[Create]](origin, options, sameOriginWithAncestors) method (2)
7359 * 5.1.4.1. PublicKeyCredential's
7360 [[DiscoverFromExternalSource]](origin, options,
7361 sameOriginWithAncestors) method (2)
7362 * 9. WebAuthn Extensions (2) (3) (4)
7363 * 9.2. Defining extensions
7364
7365 #client-extension-outputReferenced in:
7366 * 5.1. PublicKeyCredential Interface
7367 * 5.1.3. Create a new credential - PublicKeyCredential's
7368 [[Create]](origin, options, sameOriginWithAncestors) method (2)
```

5931 * 4.1.4. Use an existing credential to make an assertion -
5932 PublicKeyCredential's [[DiscoverFromExternalSource]](options)
5933 method (2)
5934 * 8. WebAuthn Extensions (2) (3)
5935 * 8.2. Defining extensions (2) (3)
5936 * 8.4. Client extension processing (2) (3)
5937 * 8.6. Example Extension

5938
5939 #authenticator-extension-processingReferenced in:
5940 * 8. WebAuthn Extensions
5941 * 8.2. Defining extensions
5942 * 8.5. Authenticator extension processing

5943
5944 #authenticator-extension-outputReferenced in:
5945 * 5.1. Authenticator data
5946 * 8. WebAuthn Extensions (2) (3)
5947 * 8.2. Defining extensions (2) (3)
5948 * 8.4. Client extension processing
5949 * 8.5. Authenticator extension processing
5950 * 8.6. Example Extension
5951 * 9.5. Supported Extensions Extension (exts)
5952 * 9.6. User Verification Index Extension (uvi)
5953 * 9.7. Location Extension (loc)
5954 * 9.8. User Verification Method Extension (uvm)

5955
5956 #typedefdef-authenticatorselectionlistReferenced in:
5957 * 9.4. Authenticator Selection Extension (authnSel)
5958

5959 #typedefdef-aaguidReferenced in:
5960 * 9.4. Authenticator Selection Extension (authnSel)
5961

7369 * 5.1.4.1. PublicKeyCredential's
7370 [[DiscoverFromExternalSource]](origin, options,
7371 sameOriginWithAncestors) method (2)
7372 * 9. WebAuthn Extensions (2) (3)
7373 * 9.2. Defining extensions (2) (3)
7374 * 9.4. Client extension processing (2) (3)
7375 * 9.6. Example Extension

7376
7377 #authenticator-extension-processingReferenced in:
7378 * 6.2.1. The authenticatorMakeCredential operation
7379 * 6.2.2. The authenticatorGetAssertion operation
7380 * 9. WebAuthn Extensions
7381 * 9.2. Defining extensions
7382 * 9.5. Authenticator extension processing

7383
7384 #authenticator-extension-outputReferenced in:
7385 * 6.1. Authenticator data
7386 * 9. WebAuthn Extensions (2) (3)
7387 * 9.2. Defining extensions (2) (3)
7388 * 9.4. Client extension processing
7389 * 9.5. Authenticator extension processing
7390 * 9.6. Example Extension
7391 * 10.5. Supported Extensions Extension (exts)
7392 * 10.6. User Verification Index Extension (uvi)
7393 * 10.7. Location Extension (loc)
7394 * 10.8. User Verification Method Extension (uvm)

7395
7396 #typedefdef-authenticatorselectionlistReferenced in:
7397 * 10.4. Authenticator Selection Extension (authnSel)
7398

7399 #typedefdef-aaguidReferenced in:
7400 * 10.4. Authenticator Selection Extension (authnSel)
7401