alliance

1 UAF Protocol Specification

2 Specification Set: fido-uaf-v1.0-rd-20140209 REVIEW DRAFT

3 Editors:

- 4 Dr. Rolf Lindemann, Nok Nok Labs, Inc.
- 5 Davit Baghdasaryan, Nok Nok Labs, Inc.
- 6 Eric Tiffany, FIDO Alliance

7 Contributors:

8 Dirk Balfanz, Google

9 Abstract:

- 10 The goal of the Universal Authentication Framework is to provide a unified and extensi-
- 11 ble authentication mechanism that supplants passwords while avoiding the shortcom-
- 12 ings of current alternative authentication approaches. This approach is designed to al-
- 13 low the Relying Party to choose the best available authentication mechanism for a par-
- 14 ticular end user or interaction, while preserving the option for the Relying Party to lever-
- age emerging device security capabilities in the future without requiring additional inte-
- 16 gration effort.
- 17 This document describes the FIDO architecture in detail, it defines the flow and content
- 18 of all UAF protocol messages and presents the rationale behind the design choices.

Copyright © 2014 FIDO Alliance

19 Status:

- 20 This Specification has been prepared by FIDO Alliance, Inc. This is a Review Draft
- 21 Specification and is not intended to be a basis for any implementations as the
- 22 **Specification may change.** Permission is hereby granted to use the Specification
- solely for the purpose of reviewing the Specification. No rights are granted to prepare
- 24 derivative works of this Specification. Entities seeking permission to reproduce portions
- of this Specification for other uses must contact the FIDO Alliance to determine whether
- 26 an appropriate license for such use is available.
- 27 Implementation of certain elements of this Specification may require licenses under third
- 28 party intellectual property rights, including without limitation, patent rights. The FIDO AI-
- 29 liance, Inc. and its Members and any other contributors to the Specification are not, and
- 30 shall not be held, responsible in any manner for identifying or failing to identify any or all
- 31 such third party intellectual property rights.
- 32 THIS FIDO ALLIANCE SPECIFICATION IS PROVIDED "AS IS" AND WITHOUT ANY
- 33 WARRANTY OF ANY KIND, INCLUDING, WITHOUT LIMITATION, ANY EXPRESS OR
- 34 IMPLIED WARRANTY OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS
- 35 FOR A PARTICULAR PURPOSE.
- 36 Copyright © 2014 FIDO Alliance, Inc. All rights reserved.

Table of Contents

1 Notation	
1.1 Key Words	7
2 Overview	
2.1 Scope	
2.2 Architecture	8
2.3 Protocol Conversation	9
2.3.1 Registration	
2.3.2 Authentication	
2.3.3 Transaction Confirmation	
2.3.4 Deregistration	
3 Protocol Details	
3.1 Shared Structures and Types	
3.1.1 Version	
3.1.2 Operation Header	14
3.1.3 Type of Authenticator Attestation ID (AAID)	14
3.1.4 Type of KeyID	
3.1.5 Type of ServerChallenge	16
3.1.6 Type of FinalChallengeParams	16
<u>3.1.7 Type of TLSData</u>	16
<u>3.1.8 Type of JwkKey</u>	
3.1.9 Type of Extension	
3.1.10 Type of TrustedApps	
3.1.11 Type of Policy	20
3.2 Version Negotiation	23

3.3 Registration Operation24
3.3.1 Type of RegisterRequest26
3.3.2 Type of RegisterResponse27
3.3.3 Processing Rules
3.3.3.1 Registration Request Generation Rules for FIDO Server
3.3.3.2 Registration Request Processing Rules for FIDO Client
3.3.3.3 Registration Request Processing Rules for FIDO Authenticator
3.3.3.4 Registration Response Generation Rules for FIDO Client
3.3.3.5 Registration Response Processing Rules for FIDO Server
3.4 Authentication Operation32
3.4.1 Type of AuthenticationRequest
3.4.2 Type of AuthenticationResponse
3.4.3 Processing Rules
3.4.3.1 Authentication Request Generation Rules for FIDO Server
3.4.3.2 Authentication Request Processing Rules for FIDO Client37
3.4.3.3 Authentication Request Processing Rules for FIDO Authenticator37
3.4.3.4 Authentication Response Generation Rules for FIDO Client
3.4.3.5 Authentication Response Processing Rules for FIDO Server
3.5 Deregistration Operation
3.5.1 Type of DeregistrationRequest40
3.5.2 Processing Rules
3.5.2.1 Deregistration Request Generation Rules for FIDO Server
3.5.2.2 Deregistration Request Processing Rules for FIDO Client
3.5.2.3 Deregistration Request Processing Rules for FIDO Authenticator
4 Considerations
4.1 Protocol Core Design Considerations

4.1.1 Authenticator Metadata4
4.1.2 Authenticator Attestation4
4.1.2.1 Basic Attestation
4.1.3 Error Handling
4.1.4 Assertion Schemes
4.1.5 Username in Authenticator40
4.1.6 TLS Protected Communication
4.2 Implementation Considerations4
4.2.1 Server Challenge and Random Numbers4
4.3 Security Considerations4
4.3.1 FIDO Authenticator Security
4.3.2 Cryptographic Algorithms
4.3.3 Application Isolation
4.3.3.1 AppID and FacetID Assertion
4.3.3.2 Isolation using KHAccessToken53
4.3.4 TLS Binding
4.3.5 Personas
4.3.6 ServerData and KeyHandle5
4.3.7 Authenticator Information retrieved through UAF Client API vs. Metadata50
4.3.8 Policy Verification50
4.3.9 Replay Attack Protection50
4.3.10 Protection against Cloned Authenticators5
4.4 Interoperability Considerations
5 UAF Supported Assertion Schemes
5.1 Assertion Scheme "UAFV1TLV"60
5.1.1 KeyRegistrationData

5.1.2 SignedData	
<u>6 Definitions</u>	
Bibliography	

37 1 Notation

- ³⁸ Type names, attribute names and element names are written in *italics*.
- 39 String literals are enclosed in "", e.g. "UAF-TLV".
- 40 In formulas we use "|" to denote byte wise concatenation operations.
- 41 UAF specific terminology used in this document is defined in [FIDOGlossary].

42 **1.1 Key Words**

- 43 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
- 44 "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this doc-
- 45 ument are to be interpreted as described in [RFC2119].

46 2 Overview

The goal of this Universal Authentication Framework is to provide a unified and extensi-47 ble authentication mechanism that supplants passwords while avoiding the shortcom-48 ings of current alternative authentication approaches. The design goal of the protocol is 49 to enable Relying Parties to leverage the diverse and heterogeneous set of security ca-50 pabilities available on end users' devices via a single, unified protocol. This approach is 51 designed to allow the Relying Party to choose the best available authentication mecha-52 nism for a particular end user or interaction, while preserving the option for the Relying 53 Party to leverage emerging device security capabilities in the future without requiring 54 additional integration effort. 55

56 **2.1 Scope**

57 This document describes FIDO architecture in detail and defines the UAF protocol as a 58 network protocol. It defines the flow and content of all UAF messages and presents the

59 rationale behind the design choices.

60 Particular application-level bindings are outside the scope of this document. This docu-61 ment is not intended to answer questions such as:

- What does an HTTP binding look like for UAF?
- How can a web application communicate to FIDO Client?
- How can FIDO Client communicate to FIDO enabled Authenticators?
- The answers to these questions can be found other UAF specifications, e.g. UAFAppAPI&Binding] [UAFASM] [UAFAuthnrCommands].

67 2.2 Architecture

⁶⁸ The following diagram depicts the entities involved in UAF protocol.

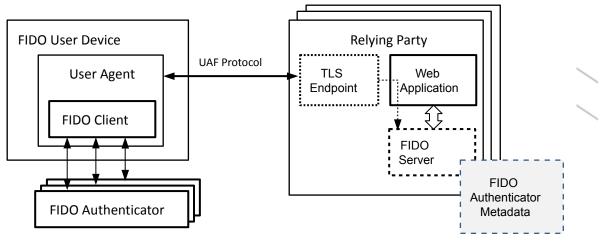


Figure 2.1: The UAF Architecture

- 69 Of these entities, only these three directly create and/or process UAF protocol mes-70 sages:
- **FIDO Server**, running on the Relying Party's infrastructure
- **FIDO Client**, part of the User Agent and running on the FIDO user device
- **FIDO Authenticator**, integrated into the FIDO user device
- 74 It is assumed in this document that a FIDO Server has access to the FIDO Authentica-
- tor Metadata [UAFAuthnrMetadata] describing all the Authenticators it will interact with.

76 2.3 Protocol Conversation

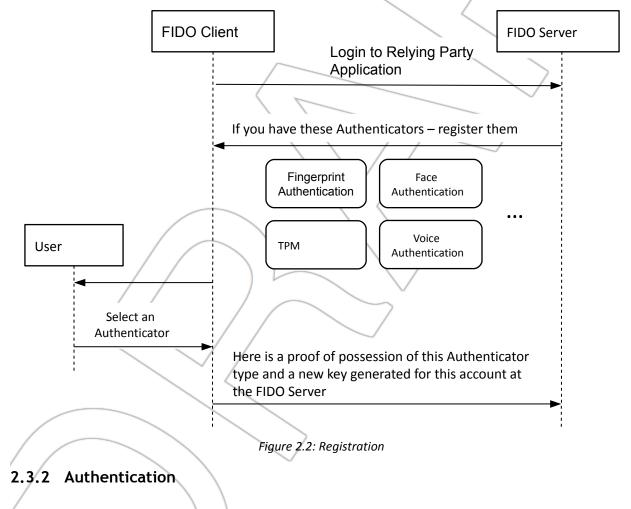
- The core UAF protocol consists of four conceptual conversations between FIDO Clientand FIDO Server.
- Registration: UAF allows the Relying Party to register a FIDO Authenticator with the user's account at the relying party. The Relying Party can specify a policy for supporting various FIDO Authenticator types. FIDO Client will only register existing FIDO Authenticators in accordance with that policy.
- Authentication: UAF allows the Relying Party to prompt the end user to authenticate using a previously registered FIDO Authenticator. This authentication can be invoked any time, at the Relying Party's discretion.
- Transaction Confirmation: In addition to providing a general authentication
 prompt, UAF provides support for prompting the user to confirm a specific trans action. This prompt includes the ability to communicate additional information to
 the client for secure display to the end user. The goal of this additional authenti cation operation is to enable Relying Parties to ensure that the user is confirming
 a specified set of the transaction details.

- **Deregistration:** The Relying Party can trigger the deletion of the Authentication Key material.
- 94 Although this document defines the FIDO Server as the initiator of requests, in a real
- 95 world deployment the first UAF operation will always follow User Agent's request (e.g.,
- 96 an HTTP request) to Relying Party.
- The following section give a brief overview of the protocol conversation for individual operations. More detailed descriptions can be found in the sections Registration Opera-
- tion, Authentication Operation, Authentication Operation, and Deregistration Operation.

100 2.3.1 Registration

102

101 The following diagram shows the message flows for the Registration operation.



103 The following diagram depicts the message flows for the Authentication operation.

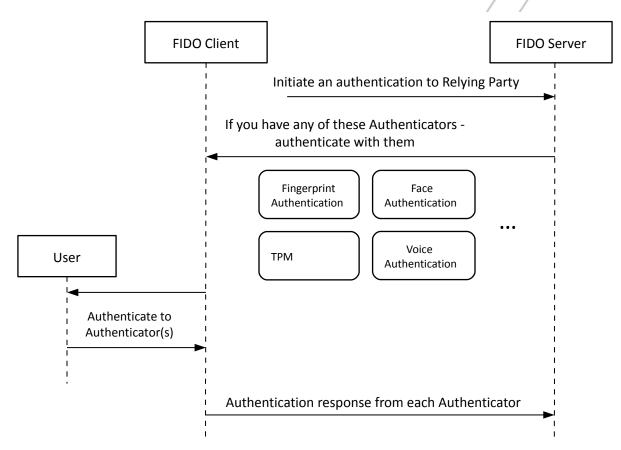
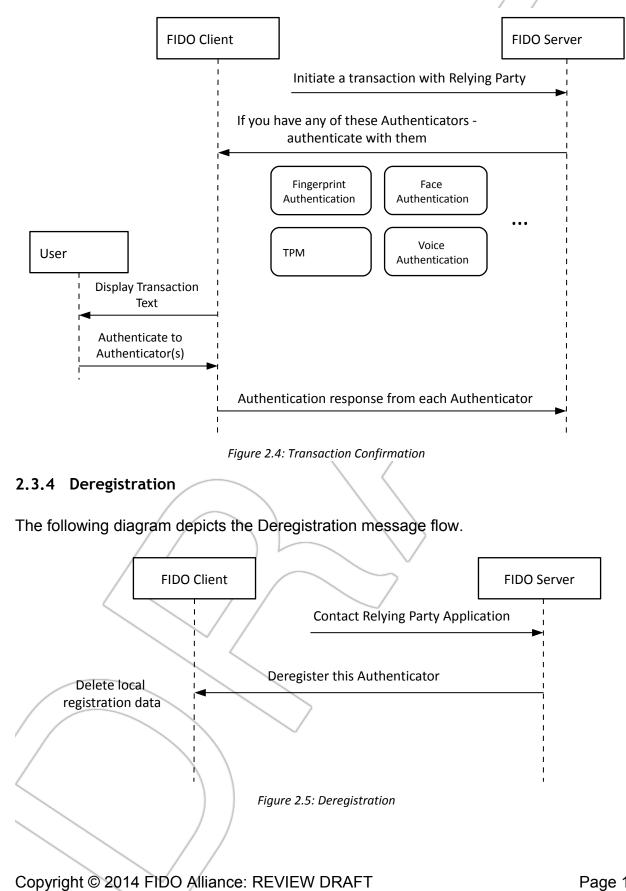


Figure 2.3: Authentication

- 104 2.3.3 Transaction Confirmation
- 105 The following figure depicts the Transaction Confirmation message flow.

106

107



108 3 Protocol Details

109 This section provides a detailed description of operations supported by the UAF Proto-110 col.

111 Support of all protocol elements is mandatory for conforming software, unless stated 112 otherwise.

- All string literals in this specification are constructed from UNICODE codepoints
 within the set U+0000..U+007F. Unless otherwise specified, protocol messages
 are transferred with a UTF-8 content encoding.
- All data used in this protocol MUST be exchanged using a secure protocol (such as TLS/HTTPS) established between FIDO Client and Relying Party; details are specified in section TLS Protected Communication.
- Unless otherwise specified the fields in UAF messages MUST be non-empty and
 if a list/array is provided it MUST have at least one entry.
- The notation base64url(byte[8..64]) reads as 8-64 bytes of data encoded in base64url, "Base 64 Encoding with URL and Filename Safe Alphabet"
 [RFC4648].
- The notation *string[5]* reads as a five-character UTF-8 formatted string of the type indicated in the declaration, typically a WebIDL [WebIDL] DOMString.
- All strings are case-sensitive unless noted otherwise.

Unless explicitly specified the "MUST" keyword applies to all steps described in this
 document

- 129 This document uses WebIDL to define UAF protocol messages. Implementations
- MUST serialize the UAF protocol messages for transmission using JSON [RFC4627]
 using UTF8 encoding.
- 132 **3.1 Shared Structures and Types**
- 133 This section defines types and structures shared by various operations.
- 134 3.1.1 Version

135 Represents a generic version with major and minor fields.

- 136 dictionary Version {
 137 int mj; // Mandatory.
- 138 int mn; // Mandatory.
- 139 }

- 140 Description:
- **mj**: Major version
- 142 **mn**: Minor version

143 3.1.2 Operation Header

144 Represents a UAF Message Request and Response header

145 146 147 148 149 150 151	<pre>dictionary OperationHeader { Version upv; // Mandatory. DOMString op; // Mandatory. Must be "Reg", "Auth" or "Dereg" DOMString appID; // Mandatory. string[1512]. DOMString serverData; // Optional, string[11536] Extension[] exts; // Optional. }</pre>
152	Description:
153 154	 upv: UAF protocol version. Major version must be 1 and minor version must be 0.
155 156	 op: Name of FIDO operation this message relates to. Note that "Auth" is used for both authentication and transaction confirmation.
157 158 159 160 161	• appID : The application id that the RP would like to assert. The new key pair that the UAF Authenticator generates will be associated with this appID. It MUST be an URI with HTTPS protocol as FIDO Client will use it to load the list of FacetIDs using this URI. Security Relevance: The AppID is used by the FIDO Client to verify the eligibility of an application to trigger use of a specific Uauth key.
162 163 164 165 166 167 168 169	• serverData : A session id created by the RP. The RP can opaquely store things like expiration times for the registration session, protocol version used and other useful information there. This data is opaque to FIDO Client and servers MAY reject a response that lacks or contains unauthorized modifications to this data. Servers that depend on it SHOULD apply and verify a cryptographically secure Message Authentication Code (MAC) to <i>serverData</i> and ensure it is cryptographically bound to other relevant portions of the message such as the <i>ServerChallenge</i> , see also section ServerData and KeyHandle.

- 170 **exts**: List of UAF Message Extensions.
- 171 3.1.3 Type of Authenticator Attestation ID (AAID)
- 172 typedef DOMString AAID;

// string[9]

173 Description:

- AAID: Each Authenticator MUST have an AAID to identify UAF enabled Authenti-174 cator models globally. Only Authenticators from the same vendor, of the same 175 Model, and with identical security characteristics may share the same AAID (see 176 Security Considerations). 177 The AAID is a string with following format – "V#M", where 178 • "#" is a separator 179 "V" indicates the Authenticator Vendor Code. This code consists of 4 hex 180 digits. 181 "M" indicates the Authenticator Model Code. This code consists of 4 hex 182 digits. 183 The Augmented BNF [ABNF] for the AAID: 184 0 4(HEXDIG) "#" 4(HEXDIG) 185 Note: HEXDIG is case insensitive, i.e. "03EF" and "03ef" are identical. 186 The FIDO Alliance is responsible for assigning Authenticator Vendor Codes. 187 • Authenticator Vendors are responsible for assigning model codes to their Au-188 thenticators. Authenticator Vendors MUST assign unique AAIDs to Authenti-189 cators with different security characteristics. 190 Fixing firmware/software bugs, adding new firmware/software features, or 191 0 changing the underlying hardware protection mechanisms will typically 192 change the security characteristics of an Authenticator and hence would 193 require a new AAID be used. 194
- 195 3.1.4 Type of KeylD

196 typedef DOMString KeyID; // base64url(byte[32...2048])

- 197 Description:
- *KeyID* is a unique identifier (within the scope of an AAID) used to refer to a specific Uauth.key. It is generated by the Authenticator and registered with a FIDO Server.
- The (AAID, KeyID) tuple MUST uniquely identify an Authenticator's registration
 for a relying party. Whenever a FIDO Server wants to provide specific information
 to a particular Authenticator it MUST use the (AAID, KeyID) tuple.
- *KeyID* must be base64url encoded within the UAF message (see above).
- *KeyID* may be used by Roaming Authenticators which don't have internal storage
 and need to store the generated Uauth keys in wrapped form (see also section
 ServerData and KeyHandle) on a FIDO Server.
- During an authentication operation FIDO Server has to provide the *KeyID* back to
 the Authenticator for the latter to unwrap the Uauth.priv key and generate a sig nature using it.

• The exact structure and content of a *KeyID* is implementation-specific.

212 3.1.5 Type of ServerChallenge

- 213 typedef DOMString ServerChallenge; // base64url(byte[8...64])
- 214 Description:
- ServerChallenge is a server-provided random challenge. Security Relevance:
 The challenge is used by the FIDO Server to verify whether an incoming re sponse is new or has already been processed. See section Replay Attack Pro tection for more details.
- The ServerChallenge should be mixed into the entropy pool of the Authenticator.
 Security Relevance: The FIDO Server SHOULD provide a challenge containing strong cryptographic randomness whenever possible. [Server Challenge and Random Numbers]
- The minimum challenge length of 8 bytes follows the requirement in [SP 800-63-1] and is equivalent to the 20 decimal digits as required in [RFC6287].
- The maximum length has been defined such that SHA-512 output can be used without truncation.
- 227 3.1.6 Type of FinalChallengeParams

228	dictionary FinalChallengeParam	ns {		\sim /
229	DOMString appID;	11	Mandatory,	string[1512].
230	ServerChallenge challenge;	~ 11	Mandatory.	
231	DOMString facetID;	TX	Mandatory,	string[1512].
232	TLSData tlsData;	< <i>11</i>	Mandatory	
233	}		-	

234 Description:

235

- The appID is taken from the Operation Header (see section Operation Header).
- The *challenge* is taken from Operation Header (see section Operation Header).
- The *facetID* is determined by FIDO Client and depends on the calling application (see section Type of TrustedApps and section AppID and FacetID Assertion for more details). Security Relevance: The *facetID* is determined by the FIDO Client and verified against the *TrustedApps* retrieved using the *appID*.
- The *tlsData* contains the TLS information to be sent by FIDO Client to the FIDO Server.
- 243 3.1.7 Type of TLSData

```
244 dictionary TLSData {
```

245 DOMString serverEndPoint;

// Mandatory, not empty. base64url

246 247 248 249	DOMSt	ring tlsServerCertificate; // Optional, not empty if present ring tlsUnique; // Mandatory, not empty. base64url ring cid_pubkey; // Optional, base64url encoded JwsKey
250	Descrip	tion:
251 252 253	Г	<i>"LSData</i> contains channel binding information [RFC5056]. Security Relevance: The <i>TLSData</i> is verified by the FIDO Server in order to detect and prevent MITM ttacks.
254	• 1	The field serverEndPoint must be set
255 256	C	to the base64url encoding of the hash of the TLS server certificate if this is available. The hash function is to be selected as follows:
257 258 259		 if the certificate's signatureAlgorithm uses a single hash function, and that hash function is either MD5 [RFC1321] or SHA-1 [RFC6234], then use SHA-256 [FIPS180-4];
260 261 262		 if the certificate's signatureAlgorithm uses a single hash function and that hash function neither MD5 nor SHA-1, then use the hash function associ- ated with the certificate's signatureAlgorithm;
263 264 265 266		 if the certificate's signatureAlgorithm uses no hash functions or uses multi- ple hash functions, then this channel binding type's channel bindings are undefined at this time (updates to is channel binding type may occur to ad- dress this issue if it ever arises).
267 268 269	С	to "None" if the TLS server certificate is not available to the processing entity (e.g., the FIDO Client) or the hash function cannot be determined as described.
270	• 1	he field <i>tlsServerCertificate</i> is optional.
271 272	C	This field must be set to the string "None" if the TLS server certificate is not available to the FIDO Client.
273 274	C	This field can be absent if (and only if) the data is available to the FIDO Client, but the FIDO Client decides not to make it available.
275 276 277	C	This field must be set to the base64url encoding of the DER encoded TLS server certificate if this data is available to the FIDO Client and the FIDO Client decides to make this data available.
278 279		The <i>tlsUnique</i> field must be set to the base64url encoded TLS channel Finished tructure or it must be set to "None" if this data is not available [RFC5929].
280	/ •/ T	he field <i>cid_pubkey</i>
281 282		is absent if the client TLS stack doesn't provide ChannelID [ChannelID] infor- mation to the processing entity (e.g., the web browser or client application).
283 284	C	must be set to "None" if ChannelID information is supported by the client-side TLS stack but has not been signaled by the TLS server.

```
Otherwise, it must be set to the base64url encoded serialized [RFC4627]
285
            0
                JwkKey structure using UTF8 encoding (see Type of JwkKey, below).
286
     Further requirements:
287
         1. If TLS Channel ID data is accessible to the web browser or client application, it
288
            SHALL be relayed to and used by FIDO Client.
289
         2. TLS Channel ID SHALL be supported by FIDO Server. However, it can only be
290
            used by FIDO Server if the related Web Server supports it.
291
         3. If TLS binding data according to [RFC5929] is accessible to the FIDO Client, it
292
            SHALL be used by FIDO Client. Depending on the constraints given by the oper-
293
            ating environment, the FIDO Server may or may not evaluate it.
294
     3.1.8 Type of JwkKey
295
296
     dictionary JwkKey {
       DOMString kty;
297
                         // Set key type to "EC".
                         // Set to "P-256".
298
       DOMString crv;
299
       DOMString x;
                         // Mandatory, not empty. base64url(byte[32])
                         // Mandatory, not empty. base64url(byte[32])
300
       DOMString y;
301
     }
     Description:
302

    JwkKey is a dictionary representing a JSON Web Key encoding of an Elliptic

303
            Curve public key [JWK]. This public key is the Channel ID public key minted by
304
            the client TLS stack for the particular Relying Party. [ChannellD] stipulates
305
            using only a particular elliptic curve, and the particular coordinate type
306
         • The field kty denotes the key type used for Channel ID. At this time only elliptic
307
            curve is supported by [ChannelID], so it must be set to "EC" [JWA].
308
         • The field crv denotes the elliptic curve on which this public key is defined. At this
309
            time only P-256 is supported by [ChannelID], so it must be set to "P-256"
310
         • The field x contains the base64url-encoding of the x coordinate of the public key
311
            (big-endian, 32-byte value).
312
           The field y contains the base64url-encoding of the y coordinate of the public key
313
            (big-endian, 32-byte value).
314
     3.1.9 Type of Extension
315
     dictionary Extension {
316
                                 // Mandatory. string[1..32].
317
       DOMString id;
       DOMString data;
                                // Mandatory. base64url(byte[1..8192]).
318
       boolean fail if_unknown;// Mandatory.
319
320
     }
     Description:
321
```

322	Generic extensions used in various operations.
323	The field <i>id</i> identifies the extension.
324 325	 The field data contains arbitrary data with a semantics agreed between Server and Client.
326 327	 The field fail_if_unknown indicates whether unknown extensions should be ignored (fail_if_unknown=false) or should lead to an error (fail_if_unknown=true).
328	3.1.10 Type of TrustedApps
329 330 331 332	<pre>dictionary TrustedApps { Version version; // Mandatory. DOMString[] ids; // Mandatory. Each list element is string[1512]. }</pre>
333	Description:
334 335 336	 TrustedApps represents a structure holding a list of FacetIDs trusted by the RP (see section AppID and FacetID Assertion). A HTTP GET query to the AppID (which is a URI) MUST return a JSON object with this structure:
337	$\{$
338	"alg": "B64S256",
339	"ids": [
340	"https://login.acme.com/",
341	"android:apk-key-hash:2jmj7l5rSw0yVb/vlWAYkK/YBwk",
342	"ios:bundle-id:com.acme.app"
343	
344	}
345	• The field version. Major must be set to 1 and minor must be set to 0.
346 347	 The field <i>ids</i> contains list of FacetIDs (see also section Type of FinalChal- lengeParams). Each list element is string[1512].
348 349 350 351	 In the Web case, the facetID is the Web Origin [RFC6454] of the web page triggering the FIDO operation, written as a URI with an empty path. Default ports are omitted. E.g. https://login.mycorp.com/
352 353 354	 In the Android case, the facetID is derived from the sha1 hash of the APK signing certificate [APK-Signing], i.e. it is the URI android:apk-key-hash:<sha1_hash-of-apk-signing-cert></sha1_hash-of-apk-signing-cert>
355	The sha1 hash can be computed as follows:

```
keytool -exportcert -alias androiddebugkey -keystore <path-to-apk-sign-
356
                   ing-keystore> &>2 /dev/null | openssl sha1 -binary | openssl base64 |
357
                   sed 's/=//g'
358

    In the iOS case, the facetID is the BundleID [BundleID], i.e. it is the URI.

359
                   ios:bundle-id:<ios-bundle-id-of-app>
360
         1. This list MUST NOT contain more than one Web Origin facetID.
361
         2. The AppID (i.e. the URL to fetch the TrustedApps object) MUST be a HTTPS
362
            URL.
363
         The TrustedApps object MUST be directly returned to a HTTPS GET request, i.e.
364
            not using any form of redirection.
365
         4. The TrustedApps object MUST be returned as object with MIME-Type "vnd.fi-
366
            do.trusted-apps+json"
367
     3.1.11 Type of Policy
368
369
     dictionary MatchCriteria {
370
       AAID aaid;
                                                   // Optional
       KeyID[] keyIDList;
                                                   // Optional
371
       unsigned long long authenticationFactor; // Optional, set of bit flags
372
       unsigned long long keyProtection;
                                                  // Optional, set of bit flags
373
374
       unsigned long long attachment;
                                                   // Optional, set of bit flags
                                                   // Optional, set of bit flags
375
       unsigned long long secureDisplay;
       unsigned long[] supportedAuthAlgs;
                                                   // Optional
376
       DOMString[] supportedSchemes;
                                                   // Optional
377
                                                   // Optional
378
       Extension[] exts;
379
     }
380
     dictionary Policy {
381
       MatchCriteria[][] accepted;
                                          Mandatory
                                        //
382
       MatchCriteria[]/disallowed;
                                        // Mandatory
383
     }
     Description:
384
            The dictionary MatchCriteria represents the matching criteria to be used in the
385
            server policy:
386
            • The field aaid contains the AAID if matching is restricted to a single AAID.
387
               The field keyIDList contains a list of the matching Authenticator KeyIDs if
388
            \cap
                matching is restricted to a set of KeyID instances. (see [FIDORegistry])
389
               The field authenticationFactor contains one or more bit flags if matching is re-
390
            0
                stricted by the authentication factor. (see [FIDORegistry])
391
               The field keyProtection contains one or more bit flags if matching is restricted
            0
392
                by the key protection. (see [FIDORegistry])
393
```

394	 The field attachment contains one or more bit flags if matching is restricted by
395	the attachment type. (see [FIDORegistry])
396	 The field secureDisplay contains one or more bit flags if matching is restricted
397	by the type of the secure display. (see [FIDORegistry])
398	 The field supportedAuthAlgs is an array containing values of supported au-
399	thentication algorithm TAG values (see [FIDORegistry], prefix
400	UAF_ALG_SIGN) if matching is restricted by the supported authentication al-
401	gorithms.
402	 The field supportedSchemes contains a list of supported encoding schemes
403	the authenticators use for KeyRegistrationData and SignedData if matching is
404	restricted by the supported schemes. See section UAF Supported Assertion
405	Schemes for details.
406	 The field exts contains a list of extensions.
407 408	 The dictionary <i>Policy</i> contains a specification of accepted Authenticators and a specification of disallowed Authenticators.
409	 The field accepted is a two dimensional array describing the required authen-
410	ticator characteristics for the server to accept a registration/authentication for
411	a particular purpose. This two dimensional array can be seen as a list of sets.
412	List elements (i.e. the sets) are alternatives (OR condition). All elements
413	within a set must be combined:
414	 The first array index indicates OR conditions (i.e. the list). Any set of au-
415	thenticator(s) satisfying these <i>MatchCriteria</i> in the first index is acceptable
416	to the server for registration/authentication.
417	 Sub-arrays of <i>MatchCriteria</i> in the second index (i.e. the set) indicate that
418	multiple authenticators (i.e. each set element) must be registered/authenti-
419	cated to be accepted by the server.
420	The <i>MatchCriteria</i> array represents ordered preferences by the server.
421	Servers SHOULD put their most preferred authenticators first, and FIDO
422	Clients SHOULD respect those preferences, either by presenting authentica-
423	tor registration/authentication options to the user in the same order, or by of-
424	fering to register/authenticate only the most preferred authenticator(s).
425	 Any authenticator that matches any of <i>MatchCriteria</i> contained in the field dis-
426	allowed MUST be excluded from eligibility for registration/authentication, re-
427	gardless of whether it matches any accepted <i>MatchCriteria</i> or not.
428	FIDO Client MUST follow the following rules while parsing server policy:
429	During registration:
430	 Policy.accepted is a list of combinations. Each combination indicates a list of
431	criteria for authenticators that the server wants the user to register. A typical
432	combination for registration contains a single criteria.

432 combination for registration contains a single criteria.

433 434	0	Follow the priority of items in <i>Policy.accepted</i> [][]. The lists are ordered with highest priority first.
435 436	0	Choose the combination who's criteria matches best with currently available authenticators
437		 Collect information about available authenticators
438		 Ignore authenticators which match the Policy.disallowed criteria
439 440		 Match collected information with the matching criteria imposed in the pol- icy
441 442	0	Guide the user to register the authenticators specified in the chosen combina- tion
443 •	Dι	uring authentication and transaction confirmation:
444 445 446	0	Note that <i>Policy.accepted</i> is a list of combinations. Each combination indi- cates a criteria which is enough to completely authenticate the current pend- ing operation
447 448	0	Follow the priority of items in <i>Policy.accepted</i> [][]. The lists are ordered with highest priority first.
449 450	0	Choose the combination who's criteria matches best with currently available authenticators
451		 Collect information about available authenticators
452		 Ignore authenticators which meet the Policy.disallowed criteria
453 454		 Match collected information with the matching criteria imposed in the pol- icy
455 456	0	Guide the user to authenticate with the authenticators specified in chosen combination
457 458		 A pending operation will be approved by Server only after all criteria of a single combination are entirely met

459 Example 1: Policy allowing either a FPS based or a Face Recognition based Authenticator 460 (generic) 461 { 462 "accepted": [[{ "authenticationFactor": 0x02}], 463 [{ "authenticationFactor": 0x10}] 464] 465 }

466 Note that in the simple example the same result could be achieved by simply combining467 the authenticationFactor bitflags.

```
Example 2: Policy allowing either a FPS based or a Face Recognition based Authenticator
468
     (short)
469
470
      {
          "accepted": [ [{ "authenticationFactor": 0x12}] ]
471
472
      The next example requires two Authenticators to be used:
473
      Example 3: Policy requiring a FPS based and a Face Recognition based Authenticator
474
     (generic)
475
476
      {
477
          "accepted": [ [{ "authenticationFactor": 0x02},
                          { "authenticationFactor": 0x10 }]
478
479
          1
480
     }
     Other criteria can be specified in addition to the authenticationFactor:
481
     Example 4: Policy requiring the combination of two bound Authenticators
482
483
      {
          "accepted": [ [{ "authenticationFactor": 0x02, "attachment": 0x01},
484
                          { "authenticationFactor": 0x10, "attachment": 0x01}]
485
486
          ]
487
```

488 **3.2 Version Negotiation**

502

- In the UAF protocol we have the UAF protocol version, the version of KeyRegistra tionData and SignedData objects (identified by the respective tags, see [FIDORegistry]),
 and the ASM version, see [UAFASM].
- The KeyRegistrationData and SignedData objects have to be parsed and verified by the FIDO Server. This verification is only possible if the FIDO Server understands the encoding and the content of KeyRegistrationData and SignedData. Each UAF protocol version supports a set of KeyRegistrationData and SignedData versions. Similarly each of the ASM Versions supports a set of KeyRegistrationData and SignedData and SignedData versions.
- As a consequence the FIDO Client must select the Authenticators which will generate
 the appropriate versions of KeyRegistrationData and SignedData.
- 499 Version negotiation is based on the following rules:
- The FIDO Client creates a set of version pairs ASM Version (av) and UAF Protocol Version (upv) as follows:
 - 1. Add all pairs supported by the FIDO Client into it.
- Intersect this set of pairs with the set of upv included in UAF Message (i.e.
 keep only those pairs where the upv value is also contained in the UAF Messos
 sage).

- Look into Authenticators available locally which are allowed by the Policy defined in the message. Remove any pair from the set which contains an av not supported by the ASM of any available authenticator.
- Select the Authenticator to be used from the ones supporting an av included in
 the set. Remove all pairs from the set which contain an av not supported by the
 selected Authenticator.
- 512 4. Look into remaining set and select "highest" version pair.
- 513 Notes:
- "Highest" of two pairs is defined as follows: Take the pair where the upv is highest. In all these pairs look into the one with highest av.
- Each one version in the pair (upv, av) consists of Major and Minor version. The
 comparison of two versions follows SAML proposal, i.e. compare the Major versions and if they are equal compare the Minor versions.
- Each UAF message contains a version field upv. UAF Protocol Version negotiation is always between FIDO Client and FIDO Server.

521 3.3 Registration Operation

- 522 The Registration operation allows the FIDO Server and the FIDO Authenticator to agree
- 523 on an Authentication Key.

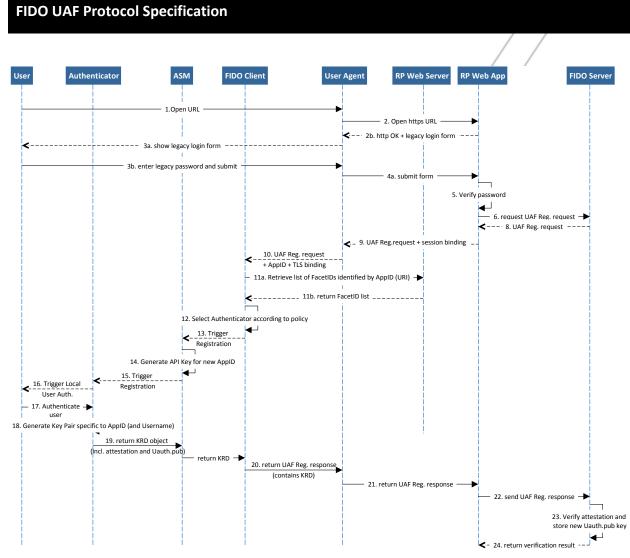


Figure 3.1: Sequence Diagram of UAF Registration

524 The following diagram depicts the cryptographic data flow for the Registration se-525 quence.

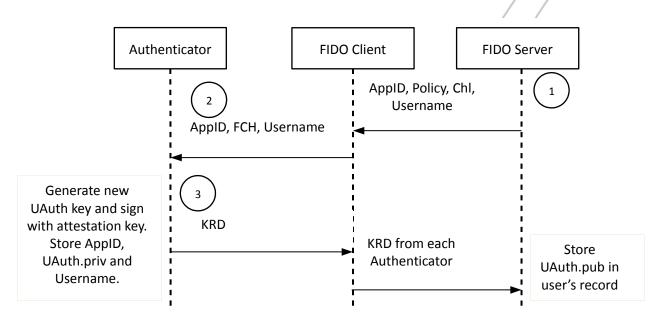


Figure 3.2: Cryptographic data flow of Registration

526 The FIDO Server sends the AppID (see section AppID and FacetID Assertion), the Au-

527 thenticator Policy (see section Type of Policy), the server generated Challenge (Chl,

see section Type of ServerChallenge) and the Username to the FIDO Client.

529 The FIDO Client computes the Final Challenge Params (FCH) from the Server Chal-

530 lenge and some other values (see section Type of FinalChallengeParams) and sends

the AppID, the FCH and the Username to the Authenticator.

532 The Authenticator creates the KeyRegistrationData object (KRD, see [UAFAuthnrCom-

mands]) containing the FCH, the newly generated user public key (Uauth.pub) and

- some other values and signs it using the Attestation private key. This KRD object is
- 535 cryptographically verified by the FIDO Server.

536 3.3.1 Type of RegisterRequest

537	dictionary RegisterRequest {		
538	OperationHeader header;	// Mandatory,	header.op must be "Reg"
539	ServerChallenge challenge; /	// Mandatory.	
540	DOMString username;	<pre>// Mandatory,</pre>	string[1128].
541	Policy policy;	// Mandatory	
542	}		

543 object[] uafRegisterRequest; // Mandat

// Mandatory, not more than one element per version

- 544 Description:
- 545 RegisterRequest contains a single registration request:
- 546 The field header contains the operation header for a registration operation.
- 547 The field challenge contains the server provided value.

- The field username contains a human-readable user name intended to allow
 the user to distinguish and select from among different accounts even at the
 same relying party.
- 551 The field policy defines which types of Authenticators are acceptable for this 552 registration operation.
- In general, a single UAF registration message may convey multiple versions of
 Registration requests. It may not contain more than one element per version.

```
555 Example 5: UAF Register Request
```

```
556
     [{
          "header": { "op": "Reg", "upv": { "mj": 1, "mn": 0 }, "appID": "https://mycorp.-
557
558
     com/fido"},
          "challenge": "qwudh827hddbawd8qbdqj3bduq3duq56t324zwasdq4wrt",
559
560
          "username": "banking personal",
561
          "policy": {
              "accepted": [[{
562
563
                  "authenticationFactor": 0000000000001ff,
564
                  "keyProtection": 000000000000000,
565
                  "attachment": 0000000000000ff,
                  "secureDisplay": 000000000000001e,
566
567
                  "supportedSchemes": "UAFV1TLV"}]],
              "disallowed": {"aaid": "1234#5678"}
568
569
          }
570
     }1
```

571 3.3.2 Type of RegisterResponse

```
dictionary AuthenticatorRegistrationAssertion {
572
573
       AAID aaid;
                                // Mandatory.
       DOMString attestationCertificateChain; // Optional. base64url(byte[1..])
574
575
       DOMString scheme;
                                // Mandatory
                                // Mandatory. base64url(byte[1..4096])
576
       DOMString krd;
       Extension [] exts;
577
                                // Optional
```

578 }

```
579 dictionary RegisterResponse {
580 OperationHeader header; // Mandatory, OperationHeader.op must be "Reg"
581 DOMString fcParams; // Mandatory, base64url encoded FinalChallengeParams
582 AuthenticatorRegistrationAssertion[] assertions; // Mandatory.
```

583 }

```
584 RegisterResponse uafRegisterResponse;
```

```
585 Description:
```

586 • 587		ne dictionary <i>AuthenticatorRegistrationAssertion</i> contains the Authenticator's sponse to a <i>uafRegisterRequest</i> message:
588 589	0	The field <i>aaid</i> contains the AAID of the specific Authenticator being used for registration.
590 591 592 593	0	The field <i>attestationCertificateChain</i> contains the Authenticator's Attestation certificate chain (excluding the root, which is included in the Authenticator Metadata [UAFAuthnrMetadata) formatted as defined for the "x5c" type of JSON Web Key [JWK].
594 595	0	The field <i>scheme</i> contains the name of the Assertion Scheme used to encode KRD. See section UAF Supported Assertion Schemes for details.
596 597 598 599	0	The field <i>krd</i> contains the <i>KeyRegistrationData</i> structure that contains the newly generated Uauth.pub signed with the Attestation Private Key. This structure is produced by Authenticator and is used only in this Registration operation. Its format can vary from one Registration Scheme to another.
600	0	The field exts contains Extensions prepared by Authenticator
601 • 602		ne dictionary <i>RegisterResponse</i> contains all fields related to the registration re- conse:
603 604	0	The field <i>header</i> contains the operation header related to a UAF registration operation.
605 606 607 608	0	The field <i>fcParams</i> is the base64url encoded serialized [RFC4627] FinalChal- lengeParams using UTF8 encoding (see section Type of FinalChal- lengeParams) which contains all parameters required for the server to verify the Final Challenge.
609 610	0	The field <i>assertions</i> contains the response data for each Authenticator being registered.
611 •	Uá	afRegisterResponse contains the UAF Registration Response message.

612 3.3.3 Processing Rules

613 **3.3.3.1** Registration Request Generation Rules for FIDO Server

614 The policy contains a 2-dimensional array of allowed *MatchCriteria* (see Type of Policy).

This array can be considered a list (first dimension) of sets (second dimension) of Au-

616 thenticators (identified by *MatchCriteria*). All Authenticators in a specific set must be

registered simultaneously in order to match the policy. But any of those sets in the list

- are valid, i.e. the list elements are alternatives.
- Construct appropriate registration policy p

 For each set of Authenticators a (to be registered simultaneously) do
 Create MatchCriteria object m
 If <i>m.aaid</i> is provided - no other fields, except keyID, attachment and exts, MUST be provided
 If <i>m.aaid</i> is not provided - at least <i>m.supportedAuthAlgs</i> and <i>m.supported-Schemes</i> MUST be provided
 If this set of Authenticators is considered disallowed, append p to m.disal- lowed. Note: Server MUST include already registered AAIDs and KeyIDs into p.disallowed to hint the Client to not suggest registering these again
 If this set of Authenticators is considered accepted, append p to m.ac- cepted, e.g. m.accepted[n] = set.
 Create a RegisterRequest object r with appropriate r.header
 Generate a random challenge and assign it to <i>r.challenge</i>
 Assign the username of the user to be registered to <i>r.username</i>
Assign <i>p</i> to <i>r.policy</i>
 Append r to the array o of message with various versions (uafRegisterRequest)
Send <i>o</i> to the FIDO Client
3.3.3.2 Registration Request Processing Rules for FIDO Client
 Choose the message <i>m</i> with major version 1 and minor version 0
Parse the message <i>m</i>
 If a mandatory field in UAF message is not present or a field doesn't corre- spond to its type and value - reject the operation
 Filter the available Authenticators with the given policy and present the filtered Authenticators to User. Make sure to not include already registered Authentica- tors for this user specified in <i>RegRequest.policy.disallowed</i>[].keyID
 Follow the priorities in server's policy and drive user experience based on these priorities.
 Obtain FacetID of the requesting Application. Resolve AppID URI and make sure that this FacetID is listed in TrustedApps.
 If FacetID is not in TrustedApps – reject the operation
Obtain TLS data if it is available
• Create a FinalChallengeParams structure fcp and set fcp.appID, fcp.challenge, fcp.facetID, and fcp.tlsData appropriately. Serialize [RFC4627] fcp using UTF8 encoding and base64url encode it.

- FinalChallenge = base64url(serialize(utf8encode(*fcp*)))
- For each authenticator that matches UAF protocol version (see section Version Negotiation) and user agrees to register:
- 657 O Add AppID, FinalChallenge, and all other required fields to the ASMRequest
 658 [UAFASM].
- Send ASMRequest to the ASM
- 660 3.3.3.3 Registration Request Processing Rules for FIDO Authenticator
- 661 See [UAFAuthnrCommands], section "Register Command".

662 3.3.3.4 Registration Response Generation Rules for FIDO Client

- Create a *uafRegisterResponse* message
- Copy uafRegisterRequest.header into uafRegisterResponse.header
- Set uafRegisterResponse.fcParams to FinalChallenge (base64url encoded serialized and utf8 encoded FinalChallengeParams)
- Append the response from each Authenticator into *uafRegisterResponse.assertions*
- Send *uafRegisterResponse* message to FIDO Server
- 670 3.3.3.5 Registration Response Processing Rules for FIDO Server

671	NOTE
672	<i>Exact error codes returned by FIDO server are not listed in the current specification.</i>
673	They will be listed in the next revision.

- Parse the message
- If protocol version (*uafRegisterResponse.header.upv*) is not supported re ject the operation
- 677 o If a mandatory field in UAF message is not present or a field doesn't corre 678 spond to its type and value reject the operation
- Verify that *uafRegisterResponse.header.serverData*, if used, passes any implementation-specific checks against its validity.
- base64url decode uafRegisterResponse.fcParams and convert it into an object
 (fcp)

683	Verify each field in <i>fcp</i> and make sure it is valid:
684	 Make sure <i>fcp.appID</i> corresponds to the one stored in FIDO Server
685 686	 Make sure <i>fcp.challenge</i> has really been generated by FIDO Server for this operation and it is not expired
687	 Make sure <i>fcp.facetID</i> is in the local list of "trusted FacetIDs"
688	 Make sure <i>fcp.tlsData</i> is as expected [TLS Binding]
689	 Reject the response if any of these checks fails
690	 For each assertion a in uafRegisterResponse.assertions
691 692	 Locate Authenticator specific authentication algorithms from Authenticator Metadata [UAFAuthnrMetadata]
693 694	 Parse TLV data from <i>a.krd</i> and make sure it has all the mandatory fields (indi- cated in Authenticator Metadata) it is supposed to have
695 696 697 698	 Hash uafRegisterResponse.fcParams using hashing algorithm suitable for this authenticator type. Look up the hash algorithm in Authenticator Metadata, field AuthenticationAlgs. It is the hash algorithm associated with the first entry related to a constant with prefix UAF_ALG_SIGN.
699	 FCHash = hash(uafRegisterResponse.fcParams)
700	 Make sure that a.krd.FinalChallenge == FCHash
701	 If comparison fails - continue with next assertion
702 703	 if entry AttestationRootCertificate for this AAID in the Metadata [UAFAuthn- rMetadata] contains at least one element:
704 705	 Obtain a.krd.Certificate and related certificate chain from a.attestationCer- tificateChain
706 707	 Obtain all entries of AttestationRootCertificate for a.aaid in Authenticator Metadata, field AttestationRootCertificate.
708 709 710	 Verify krd.Certificate and the entire certificate chain up to the Attestation Root Certificate using Certificate Path Validation as specified in [RFC5280]
711	 If verification fails – continue with next assertion
712	 Verify krd.Signature using krd.Certificate
713	 If verification fails – continue with next assertion
714	 if entry AttestationRootCertificate for this AAID in the Metadata is empty
715	 Verify krd.Signature using krd.PublicKey
716	 If verification fails – continue with next assertion
717	 Verify a.aaid == krd.AAID

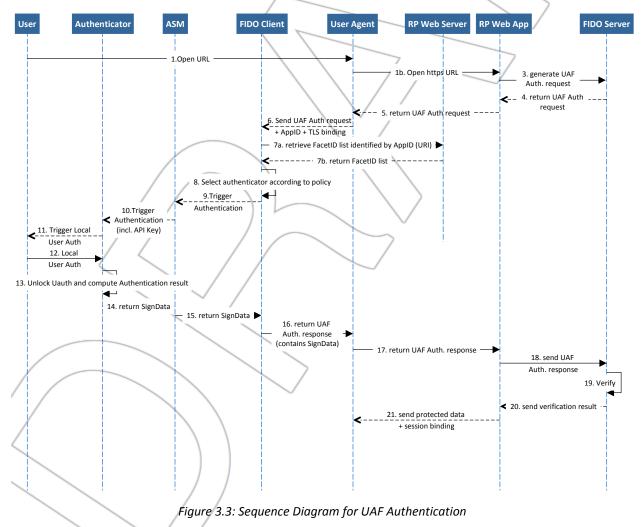
 Make sure that the set of successfully verified assertions meets the originally im posed policy If they don't meet the policy - treat the response as insufficient and reject For each positively verified assertion a 	ntinue with next assertion
	ne set of successfully verified assertions meets the originally im-
• For each positively verified assertion a	neet the policy - treat the response as insufficient and reject
	ly verified assertion a
 Store a.krd.PublicKey, a.krd.KeyID, a.krd.SignCounter, a.krd.authentica- torVersion and a.krd.AAID into a record associated with the user's identity . an entry with the same pair of AAID and KeyID already exists then fail (shoul never occur). 	d <i>a.krd.AAID</i> into a record associated with the user's identity . If the same pair of AAID and KeyID already exists then fail (should

727 3.4 Authentication Operation

728 During this operation FIDO Server asks FIDO Client to authenticate user with specified

729 Authenticators and return an authentication response. In order for this operation to suc-

raceed the Authenticator and Relying Party MUST have a previously shared registration.



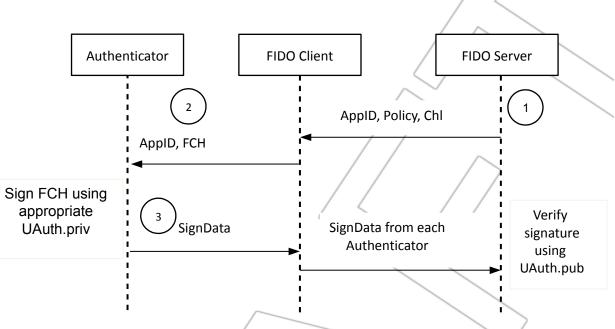


Figure 3.4: Cryptographic data flow of the Authentication message sequence

- 731 Diagram of cryptographic flow:
- 732 The FIDO Server sends the AppID (see section AppID and FacetID Assertion), the Au-
- thenticator Policy (see section Type of Policy) and the server generated Challenge (Chl,
- see section Type of ServerChallenge) to the FIDO Client.
- The FIDO Client computes the Final Challenge Params (FCH) from the Server Chal-
- lenge and some other values (see section Type of FinalChallengeParams) and sends
 the AppID and FCH to the Authenticator.
- 738 The Authenticator creates the SignedData object (see [UAFAuthnrCommands]) contain-
- ing the FCH and some other values and signs it using the UAuth.priv key. This Signed-
- 740 Data object is cryptographically verified by the FIDO Server.
- 741 **3.4.1** Type of AuthenticationRequest

742	dictionary Transaction {
743	DOMString contentType; // Mandatory
744	DOMString content; // Mandatory. base64url(byte[18192])
745	
746	dictionary AuthenticationRequest {
747	OperationHeader header; // Mandatory, header.op must be "Auth"
748	ServerChallenge challenge; // Mandatory
749	<pre>Transaction transaction; // Optional</pre>
750	Policy policy; // Mandatory
751	}

752	object[] uafAuthRequest; // Mandatory, not more than one element per version
753	Description:
754 755	 The dictionary <i>Transaction</i> contains the Transaction Text provided by the FIDO Server:
756 757	 The field contentType contains the Content-type according to [RFC2049], should be either "text/plain" or "image/png" [RFC2083].
758 759	 If the type is "text/plain" then the content MUST be ASCII encoded text with a maximum of 200 characters.
760	NOTE
761	The current specification only supports ASCII encoded transation text. The next revi-
762	sion will support internationalized text.
763 764	 The field <i>content</i> contains the content of the transaction according to the con- tent type.
765 766	 The dictionary AuthenticationRequest contains the UAF Authentication Request Message:
767 768	 The field <i>header</i> contains the operation header for an authentication opera- tion.
769	 The field <i>challenge</i> contains the server-provided value.
770 771	 The field <i>transaction</i> contains the transaction data to be explicitly confirmed by the user.
772 773	 The field <i>policy</i> defines which types of Authenticators are acceptable for this authentication operation.
774 775 776	• <i>uafAuthRequest</i> Represents UAF Authentication request message. In general, a single message may convey multiple versions of Authentication requests. It may not contain more than one element per version.



```
777
     Example 6: UAF Authentication Request
778
      [{
779
          "header": {"op": "Auth", "upv": { "mj": 1, "mn": 0 }, "appID":
780
      "https://mycorp.com/fido"},
781
          "challenge": "triz786ighwer8764g6574234515reg45z",
782
          "policy": {
783
              "accepted": [[{
784
                   "authenticationFactor": 0000000000001ff,
785
                   "keyProtection": 000000000000000e,
                   "attachment": 0000000000000ff,
786
787
                   "secureDisplay": 000000000000001e,
                   "supportedSchemes": "UAFV1TLV"}]],
788
789
              "disallowed": {"aaid": "1234#5678"}
790
          }
791
     }]
     3.4.2 Type of AuthenticationResponse
792
793
     dictionary AuthenticatorSignAssertion {
794
       AAID aaid;
                                    // Mandatory
795
       KeyID keyID;
                                    // Mandatory
       DOMString signedData; // Mandatory, e.g. "UAFV1TLV"
DOMString signedData; // Mandatory. base64url(byte[1..4096])
796
797
798
       Extension[] exts;
                                    // Optional
799
     }
800
     dictionary AuthenticationResponse {
                                   // Mandatory, header.op must be "Auth"
801
       OperationHeader header;
802
       DOMString fcParams;
                                    // Mandatory, base64url encoded FinalChallengeParams
       AuthenticatorSignAssertion[] assertions; // Mandatory
803
804
     }
     Description:
805
            The dictionary AuthenticatorSignAssertion represents the Authenticator specific
806
            response:
807
               The field aaid contains the Authenticator's AAID.
808
            0
               The field keyID contains the unique KeyID related to Uauth.priv.
809
            0
               The field scheme contains the name of the Assertion Scheme used to encode
810
            0
                signedData (e.g. "UAFV1TLV").
811
            • The field signedData is a structure that contains cryptographic signature gen-
812
                erated using Uauth priv. Such a structure is produced by an Authenticator and
813
                is used only in Authentication operations.
814
                The field exts contains extensions prepared by an Authenticator.
815
```

816 817	 The dictionary AuthenticationResponse contains the UAF Authentication Re- sponse Message:
818 819	 The field <i>header</i> contains the operation header for the authentication opera- tion.
820 821 822 823	 The field <i>fcParams</i> is the base64url encoded <i>serialized</i> [<i>RFC4627</i>] <i>Fi-nalChallengeParams</i> in UTF8 encoding (see section Type of FinalChallengeParams) which contains all parameters required for the server to verify the Final Challenge.
824 825	 The field assertions contains the list of authenticator responses related to this authentication operation.
826	Example 7: UAF Authentication Response
827	{
828	"header": {"op": "Auth", "upv": {
829	"fcParams": "eyJhcHBJRCI6Imh0dHBzOi8vbXljb3JwLmNvbS9maWRvIiwgImNoYWxsZW5nZSI6I-
830	j U0N j k 4 emhmZGtzamdoODc2dWpoZ2hqNyIsICJmYWN1dElEIjoiYW5kcm9pZDphcGsta2V5LWhhc2g6Mmpta-barrow and the second statement of
831	jdsNXJTdzB5VmIvdmxXQVlrSy9ZQndrIiwgInRsc0RhdGEiOiIifQ",
832	"assertions": [
833	{"AAID":"1234#abcd", "keyID": "1234def", "scheme": "UAFV1TLV",
834	"signedData": ""},
835	{"AAID":"1234#abce", "keyID": "fa73fg", "scheme": "UAFV1TLV",
836	"signedData":""}]
837	}

838 3.4.3 Processing Rules

- 839 **3.4.3.1** Authentication Request Generation Rules for FIDO Server
- Generate a random challenge
- Construct appropriate authentication policy
- 842 o If *MatchCritera.aaid* is provided then no other fields, except *keyID*, *attach-* 843 *ment* and *exts*, MUST be provided
- 844 o If MatchCritera.aaid is not provided then at least supportedAuthenticationAlgs
 845 and supportedSchemes MUST be provided
- In case of step-up authentication (i.e. in the case where it is expected the
 user is already known due to a previous authentication step) every item in
 Policy.accepted MUST include the AAID and KeyID of the Authenticator reg istered for this account in order to avoid ambiguities when having multiple accounts at this relying party.
- Create an authentication request message for each supported version by putting
 generated data into these, assemble all these messages into an array and send
 to FIDO Client

854 **3.4.3.2** Authentication Request Processing Rules for FIDO Client

855	•	Choose the message with major version 1 and minor version 0
856	•	Parse the message <i>m</i>
857 858		 If a mandatory field in the UAF message is not present or a field doesn't cor- respond to its type and value then reject the operation
859 860	•	Filter available Authenticators with the given policy and present the filtered list to User.
861	•	Let the user select the preferred Authenticator.
862 863	•	If <i>AuthRequest.policy.accepted</i> list is empty then suggest any registered Authen- ticator to the user for authentication
864 865	•	Obtain FacetID of the requesting Application. Resolve AppID URI and make sure that this FacetID is listed in TrustedApps.
866		 If FacetID is not in TrustedApps then reject the operation
867	•	Obtain TLS data if its available
868 869 870	•	Create a FinalChallengeParams structure <i>fcp</i> and set <i>fcp</i> . <i>AppID</i> , <i>fcp</i> . <i>challenge</i> , <i>fcp</i> . <i>facetID</i> , and <i>fcp</i> . <i>tlsData</i> appropriately. Serialize [RFC4627] fcp using UTF8 encoding and base64url encode it.
871		 FinalChallenge = base64url(serialize(utf8encode(fcp)))
872 873 874	•	For each authenticator that supports an Authenticator Interface Version AIV com- patible with message version <i>AuthRequest.header.upv</i> (see section Version Ne- gotiation) and user agrees to authenticate with:
875 876		 Add AppID, FinalChallenge, KeyID, Transaction Text (if present), and all other required fields to the ASMRequest.
877		 Send the ASMRequest to the ASM

878 3.4.3.3 Authentication Request Processing Rules for FIDO Authenticator

879 See [UAFAuthnrCommands], section "Sign Command".

880 **3.4.3.4** Authentication Response Generation Rules for FIDO Client

- Create an AuthResponse message
- Copy AuthRequest.header into AuthResponse.header

- Fill out AuthResponse.FinalChallengeParams with appropriate fields and then
 stringify it
- Append the response from each Authenticator into *AuthResponse.assertions*
- Send AuthResponse message to the FIDO Server

887 **3.4.3.5** Authentication Response Processing Rules for FIDO Server

888		NOTE
889		Exact error codes returned by FIDO server are not listed in the current specification.
890		They will be listed in the next revision.
891	•	Parse the message
892		 If protocol version is not supported – reject the operation
893 894		 If a mandatory field in UAF message is not present or a field doesn't corre- spond to its type and value - reject the operation
895 896	•	Verify that <i>AuthResponse .header.serverData</i> , if used, passes any implementa- tion-specific checks against its validity.
897	٠	base64url decode AuthResponse.fcParams and convert into an object (fcp)
898	•	Verify each field in fcp and make sure it's valid:
899		 Make sure AppID corresponds to the one stored in FIDO Server
900		 Make sure FacetID is in "trusted FacetIDs"
901		 Make sure TLSData is as expected [TLS Binding]
902 903		 Make sure ServerChallenge is a really generated by FIDO Server and is not expired
904		 Reject the response if any of these checks fails
905	٠	For each assertions in AuthResponse.assertions
906		 Locate Uauth.pub public key associated with AuthResponse.assertions.keyID
907		If such record doesn't exist - continue with next assertion
908 909	/ /	• Verify the AuthResponse.assertions.aaid against the AAID stored in the FIDO Server database at time of Registration.
910	1	 If comparison fails – continue with next assertion
911 912		 Locate Authenticator specific authentication algorithms from Authenticator Metadata (field AuthenticationAlgs)

913 914	 Parse AuthResponse.assertions.signedData and make sure it has all the mandatory fields (indicates in Authenticator Metadata) it's supposed to have
915 916 917	 Check the Sign Counter and make sure it is either not supported by the Au- thenticator or it has incremented (compared to the value stored in the user's record)
918	 If didn't increment - continue with next assertion
919 920 921 922	 Hash AuthResponse.FinalChallengeParams using the hashing algorithm suit- able for this authenticator type. Look up the hash algorithm in Authenticator Metadata, field AuthenticationAlgs. It is the hash algorithm associated with the first entry related to a constant with prefix UAF_ALG_SIGN.
923	 FCHash = hash(AuthResponse.FinalChallengeParams)
924	 Make sure that signedData.FinalChallenge == FCHash
925	 If comparison fails – continue with next assertion
926	 If authenticationMode == 2
927	 Make sure there is a transaction cached on Relying Party side
928	 If not – continue with next assertion
929 930	 Hash the cached transaction using hashing algorithm suitable for this au- thenticator (same hash algorithm as used for FinalChallenge)
931	 cachedTransHash = hash(cachedTransaction)
932	Make sure that the cachedTransHash == signedData.TransactionHash
933	 If comparison fails – continue with next assertion
934 935	 Use Uauth.pub key and appropriate authentication algorithm to verify the sig- nature included in SignedData
936	 If signature verification fails – continue with next assertion
937 938	 Make sure that the set of successfully verified assertions meets the originally imposed policy
939	 If they don't meet the policy – treat the response as insufficient and reject

- 940 3.5 Deregistration Operation
- This operation allows FIDO Server to ask the FIDO Authenticator to delete keys related to the particular relying party. The FIDO Server should trigger this operation when the user removes his account at the relying party.
- 944 Note: there is no deregistration response object.

945 3.5.1 Type of DeregistrationRequest

```
946
     dictionary DeregisterAuthenticator {
947
       AAID aaid;
                        // Mandatory
948
       KeyID keyID;
                         // Mandatory
949
     }
950
     dictionary DeregistrationRequest {
                                 // Mandatory, header.op must be "Dereg"
951
       OperationHeader header;
       DeregisterAuthenticator[] authenticators; // Mandatory
952
953
     }
     object[] uafDeregRequest; // Mandatory, not more than one element per version
954
     Description:
955
           The dictionary DeregisterAuthenticator contains the data required to identify the
956
            Authenticator to be deregistered:
957
               The field aaid contains the Authenticator's AAID.
958
            0
            • The field keyID contains the unique KeyID related to Uauth priv. Note: we as-
959
               sume keyID to be unique within the scope of an AAID only.
960
           The dictionary DeregistrationRequest contains the UAF Deregistration Request
961
            Message:
962
            • The field header contains the operation header related to the deregistration
963
               request".
964
               The field authenticators contains the list of the Authenticators to be deregis-
965
            0
               tered.
966
            uafDeregReguest represents UAF Deregistration request message. In general, a
967
            single message may convey multiple versions of deregistration requests. It may
968
            not contain more than one element per version.
969
```

970	Example	e 8: UAF Deregistration Request
971	[{	
972	"he	ader": {"op": "Dereg", "upv": { "mj": 1, "mn": 0 }, "appID": "https://mycorp
973	com/fid	o"},
974	"au	thenticators": [
975	{	"aaid":"1234#abcd",
976		"keyID": "14a504423f582727ea15c96d67200727f350dc8cc2289ed8106f3b6b7ee3ebb8"},
977	{	"aaid":"1234#abce",
978		"keyID": "84a2f881a2ee7866b8fd4db94d00279a2b485b635823fcfadef79eef0c7771e4"}
979]	
980	}]	

981 3.5.2 Processing Rules

982 3.5.2.1 Deregistration Request Generation Rules for FIDO Server

- Create a deregistration request message m with major version of *m.header.upv* set to 1 and minor version set to 0
- For each Authenticator to be deregistered
- 986 Create *DeregisterAuthenticator* object *o* for Authenticator to be deregistered
- 987 Set *o.aaid* and *o.keyID* appropriately
- 988 Append o the *m.authenticators*
- 989 o delete related entry in FIDO Server's account database
- Send message to FIDO Client
- 991 **3.5.2.2** Deregistration Request Processing Rules for FIDO Client
- Choose the message with major version 1 and minor version 0
- Parse the message

998

999

- 994 o If a mandatory field in *uafDeregRequest* message is not present or a field
 995 doesn't correspond to its type and value reject the operation
- For each Authenticator that supports an Authenticator Interface Version (AIV)
 compatible with the message version uafDeregRequest.header.upv:
 - Create Deregister Command for Authenticator, containing AppID and uafDeregRequest.keyID.
- 1000 Send Deregister Command to Authenticator

- 1001 **3.5.2.3** Deregistration Request Processing Rules for FIDO Authenticator
- 1002 See [UAFAuthnrCommands], section "Deregister Command".

1003 **4 Considerations**

1004 This is the considerations section. In this section the contents is informative by default, 1005 normative clauses are clearly marked as follows:

Normative

This is a normative clause

1006 **4.1 Protocol Core Design Considerations**

1007 This section describes the important design elements used in the protocol.

1008 4.1.1 Authenticator Metadata

- 1009 It is assumed that FIDO Server has access to a list of all supported Authenticators and 1010 their corresponding Metadata. Authenticator Metadata [UAFAuthnrMetadata] contains 1011 information such as:
- Supported Registration and Authentication Schemes
- Authentication Factor, Installation type, supported content-types and other supplementary information, etc.
- 1015 In order to make a decision about which Authenticators are appropriate for a specific
- 1016 transaction, FIDO Server looks up the list of Authenticator Metadata by AAID and re-
- 1017 trieves the required information from it.

Normative

Each entry in the Authenticator Metadata repository MUST be identified with a unique Authenticator Attestation ID (AAID).

1018 4.1.2 Authenticator Attestation

- Authenticator Attestation is the process of validating Authenticator model identity during registration. It allows Relying Parties to cryptographically verify that the Authenticator reported by FIDO Client is really what it claims to be.
- 1022 Using Authenticator attestation, a relying party "example-rp.com" will be able to verify
- 1023 that the Authenticator model of the "example-Authenticator", reported with AAID
- 1024 "1234#5678", is not malware running on the FIDO User Device but is really a Authenti-
- 1025 cator of model "1234#5678".

1026 FIDO Authenticators SHOULD support "Basic Attestation" described below. New Attes-1027 tation mechanisms MAY be added to the protocol over time.

FIDO Authenticators not providing sufficient protection for Attestation keys (non-attested
 Authenticators) MUST use the Uauth.priv key in order to formally generate the same
 Key Registration Data object as attested Authenticators. This behavior MUST be prop-

1031 erly declared in the Authenticator Metadata [UAFAuthnrMetadata].

1032 4.1.2.1 Basic Attestation

FIDO Servers MUST have access to a trust anchor for verifying attestation public keys (i.e. Attestation Certificate trust store) and Authenticators MUST provide its attestation signature during the registration process. The attestation trust anchor is shared with FIDO Servers out of band (as part of the Metadata). This sharing process is out of scope of this document.

1038	NOTE
1039	The protection measures of the Authenticator's attestation private key depend on
1040	the specific Authenticator model's implementation.

The FIDO Server must load the appropriate Authenticator Attestation Root Certificate from its trust store based on the AAID provided in KeyRegistrationData. The remainder of the Attestation Certificate Chain is included in the UAF Registration Response (field *AttestationCertificateChain*) and potentially the *KeyRegistrationData*. These two partial chains must be combined. Off-loading portions of the Attestation Certificate Chain from the Authenticator reduces its memory requirements.

In this Basic Attestation model, a large number of Authenticators share the same Attes tation certificate and Attestation Private Key in order to provide non-linkability (see sec tion Protocol Core Design Considerations). Authenticators can only be identified on a

1050 production batch level or an AAID level by their Attestation Certificate, and not individu-

- ally. A large number of Authenticators sharing the same Attestation Certificate provides
- 1052 better privacy, but also makes the related private key a more attractive attack target.

Normative

A given set of Authenticators sharing the same manufacturer and essential characteristics MUST NOT be issued a new Attestation Key before at least 100,000 devices are issued the previous shared key.

Normative

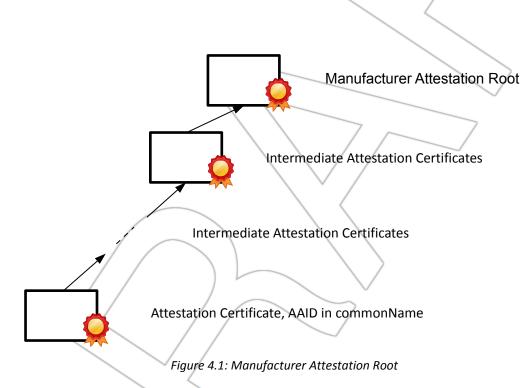
Either (a) the manufacturer attestation root certificate or (b) the root certificate related to the AAID MUST be specified in the Authenticator Metadata (see section Authenti-

cator Metadata).

This root certificate MUST be dedicated to the issuance of Authenticator Attestation certificates.

In the case (a), the root certificate might cover multiple Authenticator types (i.e. multiple AAIDs). The AAID MUST be specified in the SubjectDN CommonName (oid 2.5.4.3) of the Attestation Certificate. In the case (b) this is not required as the root certificate only covers a single AAID.

The FIDO Server MUST verify and validate the attestation certificate chain according to [RFC5280], section 6 "Certificate Path Validation". Certificate revocation status SHOULD be checked (e.g. using OCSP [RFC2560] or CRL based validation [RFC5280]).



1053 4.1.3 Error Handling

FIDO Server will inform the calling Relying Party Web Application Server (see Figure
 4.4: FIDO Interoperability Overview) about any error conditions encountered when gen erating or processing UAF messages through their proprietary API.

1057 FIDO Authenticators will inform the FIDO Client (see Figure 4.4: FIDO Interoperability

1058 Overview) about any error conditions encountered when processing commands through

the Authenticator Specific Module (ASM). See [UAFASM] and [UAFAuthnrCommands]for details.

1061 4.1.4 Assertion Schemes

1062 UAF Protocol is designed to be compatible with a variety of existing Authenticators 1063 (TPMs, Fingerprint Sensors, Secure Elements, etc.) and also future Authenticators de-1064 signed for FIDO. Therefore extensibility is a core capability designed into the protocol.

- 1065 It is considered that there are two particular aspects that need careful extensibility.1066 These are:
- Cryptographic key provisioning (Registration Assertions)
- Cryptographic authentication and signature (Authentication Assertion)
- 1069 The combination of Registration and Authentication Assertion is called an Assertion 1070 Scheme.
- 1071 The UAF protocol allows plugging in new Assertion Schemes. See also section UAF 1072 Supported Assertion Schemes.
- 1073 The Registration Assertion defines how and in which format a cryptographic key is ex-1074 changed between the Authenticator and the FIDO Server.
- 1075 The Authentication Assertion defines how and in which format the Authenticator gener-1076 ates a cryptographic signature.
- 1077 The generally-supported Assertion Schemes are defined in [FIDORegistry].

1078 4.1.5 Username in Authenticator

FIDO UAF supports Authenticators acting as first authentication factor (i.e., replacing username and password). In this case the Authenticator stores the username (uniquely identifying an account at the specific relying party) internally. See [UAFAuthnrCommands], section "Sign Command" for details.

1083 4.1.6 TLS Protected Communication

Normative

[C-General-010] In order to protect the data communication between FIDO Client and FIDO Server a protected TLS channel MUST be used by FIDO Client (or User Agent) and the [S-General-010] Relying Party for all protocol elements.

- The server endpoint of TLS connection MUST be at the Relying Party
- The client endpoint of TLS connection MUST be either FIDO Client or User Agent
- [C-General-010.1] TLS Client and Server [S-General-010.1] SHOULD use TLS v1.2 or newer. The use of TLS v1.1 is recommended if TLS v1.2 or higher are not available. The

"anon" and "null" TLS crypto suites are not allowed and MUST be rejected; insecure crypto-algorithms in TLS (e.g. MD5, RC4, SHA1) SHOULD be avoided [SP 800-131A].

- [C-General-10.3] TLS Client MUST verify and validate the server certificate chain according to [RFC5280], section 6 "Certificate Path Validation". Certificate revocation status MUST be checked (e.g. using OCSP [RFC2560] or CRL based validation [RFC5280]), as well as via TLS server identity checking [<u>RFC6125</u>].
- [C-General-10.3] TLS Client's trusted certificate root store MUST be properly maintained and at least require the CAs included in the root store to annually pass Web Trust or ETSI audits for SSL CAs.
- 1084 See [TR-03116-4] and [SHEFFER-TLS] for more recommendations on how to use TLS.

1085 4.2 Implementation Considerations

1086 4.2.1 Server Challenge and Random Numbers

Normative

Server Challenges (see section Type of ServerChallenge) need appropriate random sources in order to be effective (see [RFC4086] for more details). The (pseudo-)random numbers used for generating the Server Challenge SHOULD successfully pass the randomness test specified in [Coron99].

1087 4.3 Security Considerations

There is no "one size fits all" authentication method. The FIDO goal is to decouple the user verification method from the authentication protocol and the authentication server, and to support a broad range of user verification methods and a broad range of assurance levels. FIDO authenticators should be able to leverage capabilities of existing computing hardware, e.g. mobile devices or smart cards.

- 1093 The overall assurance level of electronic user authentications highly depends (a) on the 1094 security and integrity of the user's equipment involved and (b) on the authentication 1095 method being used to authenticate the user.
- 1096 When using FIDO, users should have the freedom to use any available equipment and
- a variety of authentication methods. The relying party needs reliable information about
 the security relevant parts of the equipment and the authentication method itself in order

- 1099 to determine whether the overall risk of an electronic authentication is acceptable in a 1100 particular business context.
- 1101 It is important for the UAF protocol to provide this kind of reliable information about the
- security relevant parts of the equipment and the authentication method itself to theFIDO server.
- 1104 The overall security is determined by the weakest link. In order to support scalable se-
- 1105 curity in FIDO, the underlying UAF protocol needs to provide a very high conceptual se-
- 1106 curity level, so that the protocol isn't the weakest link.

Relying Parties define Acceptable Assurance Levels FIDO Alliance envisions a 1107 broad range of FIDO Clients, FIDO Authenticators and FIDO Servers to be offered by 1108 various vendors. Relying parties should be able to select a FIDO Server providing the 1109 appropriate level of security. They should also be in a position to accept FIDO Authenti-1110 cators meeting the security needs of the given business context, to compensate assur-1111 ance level deficits by adding appropriate implicit authentication measures, and to reject 1112 authenticators not meeting their requirements. FIDO does not mandate a very high as-1113 surance level for FIDO Authenticators, instead it provides the basis for authenticator 1114 and user verification method competition. 1115

- Authentication vs. Transaction Confirmation Existing Cloud services are typically 1116 based on authentication. The user starts an application (i.e. User Agent) assumed to be 1117 trusted and authenticates to the Cloud service in order to establish an authenticated 1118 communication channel between the application and the Cloud service. After this au-1119 thentication, the application can perform any actions to the Cloud service. The service 1120 provider will attribute all those actions to the user. Essentially the user authenticates all 1121 actions performed by the application in advance until the service connection or authenti-1122 cation times out. This is a very convenient way as the user doesn't get distracted by 1123 manual actions required for the authentication. It is suitable for actions with low risk con-1124 1125 sequences.
- However, in some situations it is important for the relying party to know that a user really has seen and accepted a particular content *before* he authenticates it. This method is typically being used when non-repudiation is required. The resulting requirement for this scenario is called What You See Is What You Sign (WYSIWYS).
- 1129 Scenario is called what fou See is what fou Sign (WFSIWFS). 1130 UAF supports both methods; they are called "Authentication" and "Transaction Confir-
- mation". The technical difference is, that with Authentication the user confirms a random challenge, where in the case of Transaction Confirmation the user also confirms a hu-
- 1132 man readable content, i.e. the contract. From a security point, in the case of authentica-
- tion the application needs to be trusted as it performs any action once the authenticated
- 1135 communication channel has been established. In the case of Transaction Confirmation
- only the secure display component implementing WYSIWYS needs to be trusted, not
- 1137 the entire application.

1138 **Distinct Attestable Security Components** For the relying party in order to determine

- the risk associated with an authentication, it is important to know details about some
- 1140 components of the user's environment. Web Browsers typically send a "User Agent"
- string to the web server. Unfortunately any application could send any string as "User
- Agent" to the relying party. So this method doesn't provide strong security. UAF is based on a concept of cryptographic attestation. With this concept, the component to be
- 1144 attested owns a cryptographic secret and authenticates its identity with this crypto-
- 1145 graphic secret. In UAF the cryptographic secret is called "Authenticator Attestation Key".
- 1146 The relying party gets access to reference data required for verifying the attestation.
- 1147 In order to enable the relying party to appropriately determine the risk associated with 1148 an authentication, all components performing significant security functions need to be 1149 attestable.
- 1150 In UAF significant security functions are implemented in the "FIDO Authenticators". Se-1151 curity functions are:
- 1152 1. Protecting the attestation key.
- Generating and protecting the Authentication key(s), typically one per relying party and user account on relying party.
- 1155 3. Providing the WYSIWYS capability ("Secure Display" component).
- Some FIDO Authenticators might implement these functions in software running on the FIDO User Device, others might implement these functions in hardware. Some FIDO Authenticators might even be formally evaluated and accredited to some national scheme. Each FIDO Authenticator model has an attestation ID (AAID), uniquely identifying the related security properties. Relying parties get access to these security properties of the FIDO Authenticators and the reference data required for verifying the attestation.
- **Resilience to leaks from other verifiers** One of the important issues with existing authentication solutions is a weak server side implementation, affecting the security of authentication of typical users to *other* relying parties. It is the goal of the UAF protocol to decouple the security of different relying parties.
- 4467 Descuption Heat Variation Mathed from Authentication Protocol Lands
- **Decoupling User Verification Method from Authentication Protocol** In order to decouple the user verification method from the authentication protocol, UAF is based on
- an extensible set of cryptographic authentication algorithms. The cryptographic secret
- 1170 will be unlocked after user verification by the Authenticator. This secret is then used for
- 1171 the authenticator-to-relying party authentication. The set of cryptographic algorithms is
- 1172 chosen according to the capabilities of existing cryptographic hardware and computing
- 1173 devices. It can be extended in order to support new cryptographic hardware.

1174 **Privacy Protection** Different regions in the world have different privacy regulations.

1175 The UAF protocol should be acceptable in all regions and hence MUST support the

1176 highest level of data protection. As a consequence, UAF doesn't require transmission of

biometric data to the relying party nor does it require the storage of biometric reference

data [BioVocab] at the relying party. Additionally, cryptographic secrets used for differ-

ent relying parties shall not allow the parties to link actions to the same user entity. UAF supports this concept, known as non-linkability. Consequently, the UAF protocol doesn't

1181 require a trusted third party to be involved in every transaction.

1182 Relying parties can interactively discover the AAIDs of all enabled FIDO Authenticators

on the FIDO User Device using the Discovery interface [FIDO-UAF-Client-API]. The

1184 combination of AAIDs adds to the entropy provided by the client to relying parties.

Based on such information, relying parties can fingerprint clients on the internet (see

1186 Browser Uniqueness at eff.org and <u>https://wiki.mozilla.org/Fingerprinting</u>). In order to

1187 minimize the entropy added by FIDO, the user can enable/disable individual Authentica-

tors – even when they are embedded in the device (see [Error: Reference source not

- 1189 found], section "privacy considerations").
- 1190 4.3.1 FIDO Authenticator Security
- 1191 See [UAFAuthnrCommands].

1192 **4.3.2 Cryptographic Algorithms**

In order to keep key sizes small and to make private key operations fast enough for
small devices, it is suggested that implementers prefer ECDSA in combination with
SHA-256 / SHA-512 hash algorithms. However, the RSA algorithm is also supported.
See [FIDORegistry] "Authentication Algorithms and Key Formats" for a list of generally
supported cryptographic algorithms.

1198 One characteristic of ECDSA is that it needs to produce, for each signature generation, 1199 a fresh random value. For effective security, this value must be chosen randomly and 1200 uniformly from a set of modular integers, using a cryptographically secure process. 1201 Even slight biases in that process may be turned into attacks on the signature schemes. 1202 If such random values cannot be provided under all possible environmental conditions, 1203 then a deterministic version of ECDSA should be used (see [RFC6979]).

1204 4.3.3 Application Isolation

1205 There are two concepts implemented in UAF to prevent malicious applications from mis-1206 using AppID specific keys registered with FIDO Authenticators. First concept is called 1207 "FacetID Assertion" and second is based on the "KHAccessToken".

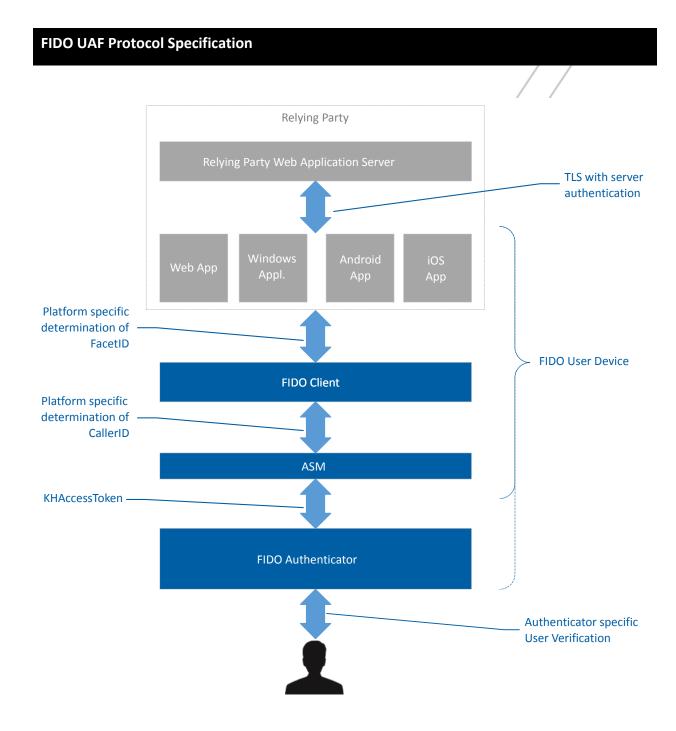
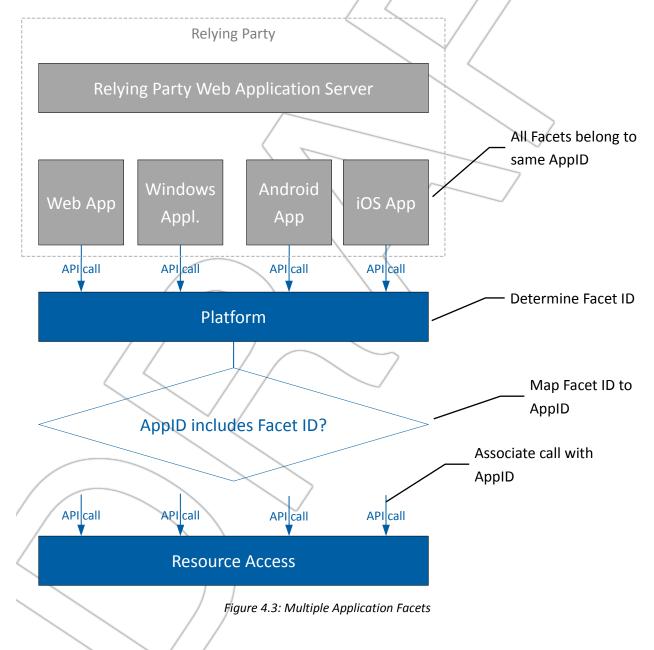


Figure 4.2: FIDO Entity Verification Overview

1208 4.3.3.1 AppID and FacetID Assertion

- 1209 Applications often run on multiple platforms. For example, MyCorp runs a web service
- 1210 (on mycorp.com), has an Android app, and an iOS app. For many use cases, it is desir-
- able to reliably establish that those three aspects of such a service all belong to the
- 1212 same application.
- 1213 The main idea here is that instead of binding user authentication keys to web origins 1214 only, we bind them to a more generic application identity (AppID). So instead of saying

- 1215 "this keypair can only be used with mycorp.com", we say "this keypair can only be used 1216 by the MyCorp applications".
- 1217 An "application", for the purpose of this section, can have multiple facets. For example, 1218 the various facets of the "MyCorp application" could be:
- The web site mycorp.com
- An Android app signed with a certain public key
- The iOS app with the iOS Bundle ID com.mycorp
- 1222 ...
- 1223 The following diagram depicts the facet architecture.



- 1226 The calling app passes its AppID (e.g. "https://mycorp.com/app-identity") to the API. On
- each platform, the FIDO Client will identify the calling app, and thus determine its
- 1228 FacetID (see section Type of TrustedApps). It then resolves the AppID and checks
- 1229 whether the FacetID is indeed included in the TrustedApps list returned by accessing
- the AppID (URL) using HTTP GET. For example, the browser is be able to see the web
- origin of the calling app. Similarly, an Android system component like the Account Man-
- ager could identify the APK signing key of the Android app making an API call into the
- 1233 Account Manager There is a similar mechanism in iOS.
- 1234 Note, that the FIDO Client must correctly determine the FacetID (see section Type of
- 1235 TrustedApps for more details). FIDO Client and ASM vendors should implement vendor
- specific component verification methods in order to determine whether FIDO Client /
 ASM are legitimate.
- 1238 The UAF protocol supports passing FacetID to the FIDO Server and including the
- 1239 FacetID in the computation of the authentication response.
- 1240 A weakness in the facet identification mechanism results in a security vulnerability, i.e.,
- identity assertions that are issued to facets other than those legitimately belonging to an
- application. In contrast, a weakness in the application identity matching mechanism re-
- sults in a privacy (but not the above-mentioned security) vulnerability, causing the au-
- 1244 thenticator to use an authentication key (in other words, a user identifier) that should
- 1245 have been reserved for a different application.
- 1246 4.3.3.2 Isolation using KHAccessToken
- Authenticators might be implemented in dedicated hardware and hence might not be able to verify the calling software entity (i.e. the ASM).
- 1249 The KHAccessToken allows restricting access to the keys generated by the FIDO Au-1250 thenticator to the intended ASM. It is based on a Trust On First Use (TOFU) concept.
- 1251 FIDO Authenticators are capable of binding UAuth keys with a key provided by the 1252 caller (i.e. the ASM). This key is called KHAccessToken.
- 1253 This technique allows making sure that registered keys are only accessible by the caller
- that originally registered them. A malicious App on a mobile platform won't be able to
- access keys by bypassing the related ASM (assuming that this ASM originally registered these keys).
- 1257 The KHAccessToken is typically specific to the AppID, PersonalD, ASMToken and the 1258 CallerID. See [UAFASM] for more details.
- 1259 Note: On some platforms, the ASM additionally might need special permissions in order
- to communicate with the FIDO Authenticator or reliably identify the calling application.
- 1261 Some platforms to not provide a means to reliably enforce access control among appli-1262 cations.

1263 4.3.4 TLS Binding

- Various channel binding methods have been proposed (e.g. [RFC5929] and [ChannellD]).
- 1266 UAF relies on TLS server authentication for binding authentication keys to AppIDs.
- 1267 There are threats:
- 1268 1. Attackers might fraudulently get a TLS server certificate for the same AppID as 1269 the relying party and they might be able to manipulate the DNS system.
- 1270 2. Attackers might be able to steal the relying party's TLS server private key and 1271 certificate and they might be able to manipulate the DNS system.
- 1272 And there are functionality requirements:
- 1273 1. UAF transactions might span across multiple TLS sessions. As a consequence, 1274 "tls-unique" defined in [RFC5929] might be difficult to implement.
- 1275 2. Data centers might use SSL concentrators.
- Data centers might implement load-balancing for TLS endpoints using *different* TLS certificates. As a consequence, "tls-server-end-point" defined in [RFC5929],
 i.e. the hash of the TLS server certificate might be inappropriate.
- Unfortunately, hashing of the TLS server certificate (as in "tls-server-end-point") 1279 also limits the usefulness of the channel binding in a particular, but guite common 1280 circumstance. If the client is operated behind a trusted (to that client) proxy that 1281 acts as a TLS man-in-the-middle, your client will see a different certificate than 1282 the one the server is using. This is actually quite common on corporate or mili-1283 1284 tary networks with a high security posture that want to inspect all incoming and outgoing traffic. If the FIDO Server just gets a hash value, there's no way to dis-1285 tinguish this from an attack. If sending the entire certificate is acceptable from a 1286 performance perspective, the server can examine it and determine if it is a certifi-1287 cate for a valid name from a non-standard issuer (likely administratively trusted) 1288 or a certificate for a different name (which almost certainly indicates a forwarding 1289 1290 attack).
- 1291 See section Type of TLSData for more details.

1292 **4.3.5** Personas

- 1293 FIDO supports unlinkability [AnonTerminology] of accounts at different relying parties by 1294 using relying party specific keys.
- 1295 Sometimes users have multiple accounts at a particular relying party and even want to
- 1296 maintain unlinkability between these accounts.
- 1297 Today, this is difficult and requires certain measures to be strictly applied.

- FIDO does not want to add more complexity to maintaining unlinkability between ac-1298 counts at a relying party. 1299
- In the case of Roaming Authenticators, it is recommended to use different Authentica-1300
- tors for the various personas (e.g. "business", "personal"). This is possible as Roaming 1301 Authenticators typically are small and not excessively expensive. 1302
- In the case of Bound Authenticators, this is different. FIDO recommends the concept of 1303 Personas for this situation. 1304
- All relevant data in an Authenticator are related to one Persona (e.g. "business" or "per-1305 sonal"). Some administrative interface (not standardized by FIDO) of the Authenticator 1306 may allow maintaining and switching Personas. 1307
- The Authenticator will only "know" / "recognize" data (e.g. authentication keys, User-1308
- names, KeyIDs, ...) related to the Persona being active at that time. 1309
- With this concept, the User can switch to the "Personal" Persona and register new ac-1310
- counts. After switching back to "Business" Persona, these accounts will not be recog-1311
- nized by the Authenticator (until the User switches back to "Personal" Persona again). 1312
- In order to support the persona feature, the FIDO Authenticator Commands specifica-1313
- tion [UAFAuthnrCommands] supports the use of a 'PersonalD' to identify the persona in 1314
- use by the authenticator. How Personas are managed or communicated with the user is 1315
- 1316 out of scope for FIDO.

4.3.6 ServerData and KeyHandle 1317

- Data contained in the field serverData (see section Operation Header) of UAF requests 1318 is sent to the FIDO Client and will be echoed back to the FIDO Server as part of the re-1319 1320 lated UAF response message.
- The FIDO Server should not assume any kind of implicit integrity protection of such data 1321
- nor any implicit session binding. FIDO Server must explicitly bind the serverData to an 1322 active session. 1323
- In some situations, it is desirable to protect sensitive data such that it can be stored in 1324 arbitrary places (e.g. in serverData or in the KeyHandle). In such situations, the confi-1325 dentiality and integrity of such sensitive data must be protected. This can be achieved 1326 by using a suitable encryption algorithm, e.g. AES with a suitable cipher mode, e.g. 1327 CBC or CTR [CTRMode]. This cipher mode needs to be used correctly. For CBC, for 1328 example, a fresh random IV for each encryption is required. The data might have to be 1329 padded first in order to obtain an integral number of blocks in length. The integrity pro-1330 tection can be achieved by adding a MAC on the ciphertext, using a different key for the 1331 MAC, e.g. using HMAC [FIPS198-1]. Alternatively, an authenticated encryption scheme 1332 1333 such as AES-GCM [SP 800-38D] or AES-CCM [SP 800-38C] could be used. Such a scheme provides both integrity and confidentiality in a single algorithm and using a sin-1334
- gle key. 1335

1336 If protecting *serverData*, the MAC should also be over some data that binds the data to 1337 its associate message, for example by re-including the *challenge* value in the authenti-

1338 cated serverData.

1339 4.3.7 Authenticator Information retrieved through UAF Client API vs. Metadata

- 1340 Several Authenticator properties (e.g. UserVerificationMethods, KeyProtection, Secure-
- 1341 Display, ...) are available in the Metadata [UAFAuthnrMetadata] and through the UAF
- 1342 Client API. The properties included in the Metadata are authoritative and are provided
- 1343 by a trusted source. When in doubt, decisions should be based on the properties re-
- trieved from the Metadata as opposed to the data retrieved through the UAF Client API.
- 1345 However, the properties retrieved through UAF Client API provide a good "hint" what to
- expect from the Authenticator. Such "hints" are well suited to drive and optimize theuser experience.

1348 **4.3.8 Policy Verification**

- 1349 FIDO UAF Response messages do not include *all* parameters received in the related
- 1350 UAF request message into the to-be-signed object. As a consequence, any MITM could 1351 modify such entries.
- 1352 FIDO Server will detect such changes if the modified value is unacceptable.

For example, a MITM could replace a generic policy by a policy specifying only the
weakest possible FIDO Authenticator. Such a change will be detected by FIDO Server if
the weakest possible FIDO Authenticator does not match the initial policy (see section

1356 **Processing Rules**).

1357 4.3.9 Replay Attack Protection

- 1358 FIDO UAF protocol specifies two different methods for replay-attack protection:
- 1359 1. secure transport protocol (TLS)
- 1360 2. Server Challenge.
- The TLS protocol by itself protects against replay-attacks when implemented correctly [TLS].
- 1363 Additionally, each protocol message contains some random bytes called "Server Chal-
- 1364 lenge". The FIDO Server only accepts incoming FIDO UAF messages, if the Server
- 1365 Challenge can be verified. This verification is done by recomputing the FinalChallenge
- included in the signed response object KRD and SignedData. See section Type of Fi-
- 1367 nalChallengeParams.

1368 **4.3.10** Protection against Cloned Authenticators

FIDO UAF relies on the Uauth key to be protected and managed by an Authenticator with the security characteristics specified for the model (identified by the AAID). The security is better when only a single Authenticator with that specific Uauth key instance exists. Consequently FIDO UAF specifies some protection measures against cloning of Authenticators.

Firstly, if the Uauth privates keys are protected by appropriate measures then cloning should be hard as such keys cannot be extracted easily.

1376 Secondly, UAF specifies a Signature Counter (see section Authentication Response

1377 Processing Rules for FIDO Server). This counter is increased by every signature oper-

1378 ation. If a cloned Authenticator is used, then the subsequent use of the original Authen-

ticator would include a signature counter lower to or equal to the previous (malicious)

1380 operation. Such an incident can be detected by the FIDO Server.

1381 **4.4 Interoperability Considerations**

1382 FIDO supports Web Applications, Mobile Applications and Native PC Applications.

1383 These environments require different bindings in order to achieve interoperability.

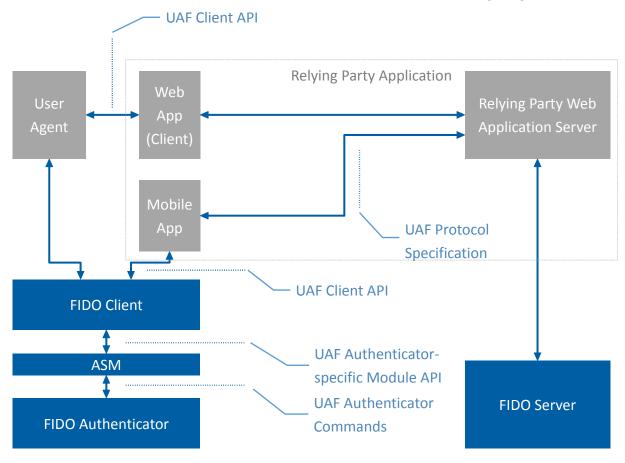


Figure 4.4: FIDO Interoperability Overview

Web applications typically consist of the web application server and the related Web App. The Web App code (e.g. HTML and JavaScript) is rendered and executed on the client side by the User Agent. The Web App code talks to the User Agent via a set of

1387 JavaScript APIs, e.g. HTML DOM. The FIDO ECMAScript binding [ECMAScript] is de-1388 fined in [UAFAppAPI&Binding]. The protocol between the Web App and the Relying

fined in [UAFAppAPI&Binding]. The protocol between the Web App and the Relying
 Party Web Application Server is typically proprietary. Web Apps SHALL use the UAF

Party Web Application Server is typically proprietary. Web Apps SHALL use
 message format defined in this document (see section Protocol Details).

Mobile Apps play the role of the User Agent and the Web App (Client). The protocol between the Mobile App and the Relying Party Web Application Server is typically proprietary. In order to ensure interoperability, such Apps SHALL use the UAF message

1394 format defined in this document (see section Protocol Details).

1395 **Native PC Applications** play the role of the User Agent, the Web App (Client) and po-1396 tentially also the FIDO Client. Those applications are typically expected to be indepen-

1396 dent from any particular Relying Party Web Application Server. These applications

1398 should use the UAF HTTP Binding defined in [UAFAppAPI&Binding].

1399	NOTES
1400	The objects KeyRegistrationData and SignedData [UAFAuthnrCommands] are gen-
1401	erated and signed by the FIDO Authenticators and have to be verified by the FIDO
1402	Server. Verification will fail if the values are modified during transport.
1403	The ASM API [UAFASM] specifies the standardized API to access Authenticator Spe-
1404	cific Modules (ASMs) on Desktop PCs and Mobile Devices.
1405	The document [UAFAuthnrCommands] does not specify a particular protocol or API.
1406	Instead it lists the minimum data set and a specific message format which needs to
1407	be transferred to and from the FIDO Authenticator.

1408 **5 UAF Supported Assertion Schemes**

1409 5.1 Assertion Scheme "UAFV1TLV"

- 1410 This Assertion Scheme allows the Authenticator and the FIDO Server to exchange an 1411 asymmetric authentication key generated by the Authenticator.
- 1412 The Authenticator MUST generate a key pair (UAuth.pub/UAuth.priv) to be used with 1413 authentication algorithms listed in "FIDO Registry of Predefined Values".
- 1414 This scheme is using Tag Length Value (TLV) compact encoding to encode KRD and
- 1415 SignedData messages generated by Authenticators. This is the default scheme for UAF
- 1416 protocol.
- 1417 TAGs and Algorithms are defined in [FIDORegistry].

Normative

[S-Auth-005] Conforming FIDO Servers MUST support all authentication algorithms and key formats listed in document "FIDO Registry of Predefined Values", section "Authentication Algorithms and Key Formats".

[A-Auth-002] Conforming Authenticators MUST support at least one Authentication Algorithm and one Key Format listed in in [FIDORegistry], section "Authentication Algorithms and Key Formats".

1418 5.1.1 KeyRegistrationData

- 1419 See [UAFAuthnrCommands], section "Register Command".
- 1420 5.1.2 SignedData
- 1421 See [UAFAuthnrCommands], section "Sign Command".

1422 6 Definitions

1423 See [FIDOGlossary].

1424 Bibliography

- 1425 FIDO Alliance Documents:
- [FIDOGIossary] Rolf Lindemann, Davit Baghdasaryan, Brad Hill, John Kemp. FIDO
 Technical Glossary. Version v1.0-rd-20140209, FIDO Alliance, February 2014.
 See http://fidoalliance.org/specs/fido-glossary-v1.0-rd-20140209.pdf
- [UAFAppAPI&Binding] Brad Hill. FIDO Universal Authentication Framework Application API and Transport Binding. Version v1.0-rd-2014020, FIDO Alliance, February 2014. See <u>http://fidoalliance.org/specs/fido-uaf-client-api-transport-v1.0-rd-</u> 20140209.pdf
- [UAFAuthnrCommands] Davit Baghdasaryan, John Kemp. FIDO Universal Authenti cation Framework Authenticator Commands. Version v1.0-rd-20140209, FIDO
 Alliance, February 2014. See http://fidoalliance.org/specs/fido-uaf-authnr-cmds-v1.0-rd-20140209.pdf
- [UAFASM] Davit Baghdasaryan, John Kemp. FIDO Universal Authentication Frame work Authenticator-specific Modules. Version v1.0-rd-20140209, FIDO Alliance,
 February 2014. See <u>http://fidoalliance.org/specs/fido-uaf-asm-api-v1.0-rd-</u>
 20140209.pdf
- 1441[UAFAuthnrMetadata]Davit Baghdasaryan, Brad Hill. FIDO Universal Authentica-1442tion Framework Authenticator Metadata. Version v1.0-rd-20140209, FIDO Al-1443liance, February 2014. See http://fidoalliance.org/specs/fido-uaf-authnr-1444metadata-v1.0-rd-20140209.pdf
- [FIDORegistry] Rolf Lindemann, Davit Baghdasaryan, Brad Hill. FIDO Universal
 Authentication Framework Registry of Predefined Values. Version v1.0-rd 20140209, FIDO Alliance. February 2014. See <u>http://fidoalliance.org/specs/fido-</u>
 uaf-reg-v1.0-rd-20140209.pdf
- 1449 Other References:
- 1450 **[AnonTerminology]** Pfitzmann, A. and M. Hansen, "Anonymity, Unlinkability, Unob-1451 servability, Pseudonymity, and Identity Management - A Consolidated Proposal
- for Terminology", Version 0.34, <u>http://dud.inf.tu-</u>
 <u>dresden.de/literatur/Anon_Terminology_v0.34.pdf</u>
- 1454 **[ABNF]** Augmented BNF for Syntax Specifications: ABNF (<u>RFC 5234</u>)
- [APK-Signing] Signing Your Applications, accessed January 17th 2014, see
 <u>http://developer.android.com/tools/publishing/app-signing.html</u>

[BioVocab] Harmonized Biometric Vocabulary. Text of Standing Document 2 (SD 2)
 Version 8, WD 2.8, work-in-progress, ISO/IEC JTC 1/SC 37: Biometrics, 2007-

- 1459 08-22. Download:
- 1460 http://isotc.iso.org/livelink/livelink/fetch/2000/2122/327993/327973/654118/66877
- 1461 <u>52/N_3004_JTC_1_SC_37_- Harmonized_Biometric_Vocabulary_- for_informa-</u> 1462 tion.pdf2podeid=67196838vernum=0
- 1462 <u>tion.pdf?nodeid=6719683&vernum=0</u>
- [BundleID] "Configuring Your Xcode Project for Distribution", section "About Bundle
 IDs", accessed January 17th 2014, see
 https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistri
- 1466 butionGuide/ConfiguringYourApp/ConfiguringYourApp.html
- [ChannelID] Transport Layer Security (TLS) Channel IDs, D. Balfanz, Work in progress.
 (draft-balfanz-tls-channelid)
- 1469 [Clickj] Clickjacking: Attacks and Defenses, Lin-Shung Huang and Collin Jackson
- 1470 Carnegie Mellon University; Alex Moshchuk, Helen J. Wang, and Stuart1471 Schlechter Microsoft Research. Download
- 1472 <u>https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-</u>
- 1473 <u>final39.pdf</u>
- [Coron99] Coron, J. and D. Naccache, "An accurate evaluation of Maurer's universal test", LNCS 1556, February 1999. Download
 http://www.isecrep.fr/wublications/wwisecrep.adf
- 1476 <u>http://www.jscoron.fr/publications/universal.pdf</u>.
- 1477 **[CTRMode]** Comments to NIST concerning AES Modes of Operations: CTR-Mode En-1478 cryption (see
- 1479 <u>http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ctr/ctr-</u>
 1480 <u>spec.pdf</u>)
- 1481 **[ECMAScript]** ECMA Script Language Specification, see http://www.ecma-internation-1482 al.org/publications/files/ECMA-ST/Ecma-262.pdf
- 1483 [FIPS180-4] Secure Hash Standard (SHS) (FIPS 180-4)
- 1484 **[FIPS198-1]** The Keyed-Hash Message Authentication Code (HMAC)
- [JWA] Jones, M., "JSON Web Algorithms (JWA)", draft-ietf-jose-json-web-algorithms
 (work in progress), January 2014. See http://tools.ietf.org/html/draft-ietf-jose-json web-algorithms
- 1488[JWK] M. Jones, "JSON Web Key (JWK)". Draft-ietf-jose-json-web-key (work in1489progress), January 20, 2014. See http://tools.ietf.org/html/draft-ietf-jose-json-1490web-key.
- 1491 **[RFC1321]** The MD5 Message-Digest Algorithm (<u>RFC 1321</u>), R. Rivest, April 1992
- 1492 [RFC1341] MIME (Multipurpose Internet Mail Extensions) (RFC1341)

- [RFC2049] Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Cri teria and Examples (<u>RFC2049</u>)
- 1495 [RFC2083] PNG (Portable Network Graphics) Specification (RFC2083).
- 1496 **[RFC2119]** Key words for use in RFCs to Indicate Requirement Levels (<u>RFC2119</u>)
- 1497 [RFC2560] X.509 Internet Public Key Infrastructure Online Certificate Status Protocol –
 1498 OCSP (<u>RFC 2560</u>), M. Myers et al, June 1999
- 1499 **[RFC4086]** Randomness Requirements for Security (<u>RFC 4086</u>)
- **[RFC4627]** The application/json Media Type for JavaScript Object Notation (JSON),
 (<u>RFC4627</u>)
- 1502 **[RFC4648]** The Base16, Base32, and Base64 Data Encodings (<u>RFC 4648</u>)
- 1503 **[RFC5056]** On the Use of Channel Bindings to Secure Channels (<u>RFC 5056</u>).
- 1504 **[RFC5246]** See [TLS]
- **[RFC5280]** Internet X.509 Public Key Infrastructure Certificate and Certificate Revoca tion List (CRL) Profile (<u>RFC 5280</u>)
- **[RFC5639]** Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve
 Generation (<u>RFC 5639</u>), M Lochter et al, March 2010
- 1509 [RFC5929] Channel Bindings for TLS (<u>RFC 5929</u>)
- 1510 [RFC6125] Representation and Verification of Domain-Based Application Service Iden-
- 1511 tity within Internet Public Key Infrastructure using X.509 (PKIX) Certificates in the 1512 Context of Transport Layer Security (TLS) (<u>RFC6125</u>), P. Saint-Andre, J.
- 1512 Context of Transport Layer Security (TLS) (<u>RFC612</u> 1513 Hodges, March 2011
- 1514 **[RFC6234]** US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF), D. 1515 Eastlake, T. Hansen, May 2011 (RFC6234)
- 1516 [**RFC6287**] OCRA: OATH Challenge-Response Algorithm (<u>RFC 6287</u>), D. M'Raihi et al, 1517 June 2011
- 1518 **[RFC6454]** The Web Origin Concept (<u>RFC 6454</u>)
- 1519 **[RFC6979]** Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic
- 1520 Curve Digital Signature Algorithm (ECDSA) (<u>RFC6979</u>), T. Pornin, August 2013
- 1521 [SHEFFER-TLS] <u>Recommendations for Secure Use of TLS and DTLS, draft-sheffer-tls-</u>
 1522 <u>bcp-00</u>
- 1523 **[SP 800-131A]** NIST Transitions: Recommendation for Transitioning the Use of Crypto-1524 graphic Algorithms and Key Lengths (<u>NIST SP 800-131A</u>)

- **[SP 800-38C]** NIST Recommendation for Block Cipher Modes of Operation: The CCM
 Mode for Authentication and Confidentiality (<u>NIST SP 800-38C</u>).
- ISP 800-38D] NIST Recommendation for Block Cipher Modes of Operation:
 Galois/Counter Mode (GCM) and GMAC (NIST SP 800-38D).
- 1529 **[SP 800-63-1]** NIST Electronic Authentication Guideline SP 800-63-1 (<u>NIST SP 800-63-</u> 1530 <u>1</u>).
- IS31
 [TLS] The TLS Protocol Version 1.0 (RFC 2246), Version 1.1 (RFC 4346), Version 1.2

 IS32
 (RFC 5246)
- 1533 [TR-03116-4] eCard-Projekte der Bundesregierung, BSI TR-03116-4
- 1534 **[WebIDL]** <u>Web IDL</u>, World Wide Web Consortium, work in progress.