# FIDO UAF Android Protected Confirmation Assertion Format

## FIDO Alliance Proposed Standard 20 October 2020

**This version:**
https://fidoalliance.org/specs/fido-uaf-v1.2-ps-20201020/fido-uaf-apccbor-v1.2-ps-20201020.html
**Editor:**
Dr. Rolf Lindemann, Nok Nok Labs, Inc.

The English version of this specification is the only normative version. Non-normative translations may also be available.



---

## Abstract

This document defines the assertion format "APCV1CBOR" in order to use Android Protected Confirmation for FIDO UAF Transaction Confirmation.

## Status of This Document

*This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current FIDO Alliance publications and the latest revision of this technical report can be found in the FIDO Alliance specifications index at https://fidoalliance.org/specifications/.*

This document was published by the FIDO Alliance as a Proposed Standard. If you wish to make comments regarding this document, please Contact Us. All comments are welcome.

Implementation of certain elements of this Specification may require licenses under third party intellectual property rights, including without limitation, patent rights. The FIDO Alliance, Inc. and its Members and any other contributors to the Specification are not, and shall not be held, responsible in any manner for

This document has been reviewed by FIDO Aliance Members and is endorsed as a Proposed Standard. It is a stable document and may be used as reference material or cited from another document. FIDO Alliance's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment.

## Table of Contents



## 1. Notation

Type names, attribute names and element names are written as `code`.

String literals are enclosed in “”, e.g. “UAF-TLV”.

In formulas we use “|” to denote byte wise concatenation operations.

UAF specific terminology used in this document is defined in [FIDOGlossary].

All diagrams, examples, notes in this specification are non-normative.

### 1.1 Key Words

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

## 2. Overview

*This section is non-normative.*

This document defines the assertion format "APCV1CBOR" in order to use Android Protected Confirmation for FIDO Transaction Confirmation.

# 3. Data Structures for APCV1CBOR

*This section is normative.*

## 3.1 Registration Assertion

The registration assertion for the assertion format "APCV1CBOR" contains an object as specified in section 5.2.1 in [UAFAuthnrCommands], with the following specifics:

1. Only Surrogate Basic Attestation is supported. The extension "fido.uaf.android.key_attestation" [UAFRegistry] MUST be present.
2. The signature field (TAG_SIGNATURE) SHALL have zero bytes length, since the key cannot be used to create a self-signature.

## 3.2 Authentication Assertion

The authentication assertion is a TLV structure containing a CBOR encoded to-be-signed object:

| TLV Structure | | Description |
|---|---|---|
| 1 | UINT16 Tag | TAG_APCV1CBOR_AUTH_ASSERTION |
| 1.1 | UINT16 Length | Length of the structure. |
| 1.2 | UINT16 Tag | TAG_APCV1CBOR_SIGNED_DATA |
| 1.2.1 | UINT16 Length | Length of the structure. |
| 1.2.2 | UINT8 tbsData | The serialized Android Protected Confirmation CBOR object. |
| 1.3 | UINT16 Tag | TAG_AAID |
| 1.3.1 | UINT16 Length | Length of AAID |
| 1.3.2 | UINT8[] AAID | Authenticator Attestation ID |
| 1.4 | UINT16 Tag | TAG_KEYID |
| 1.4.1 | UINT16 Length | Length of KeyID |
| 1.4.2 | UINT8[] | (binary value of) KeyID |

| | | |
|---|---|---|
| | KeyID | |
| 1.5 | UINT16 Tag | TAG_SIGNATURE |
| 1.5.1 | UINT16 Length | Length of Signature |
| 1.5.2 | UINT8[] Signature | Signature calculated using UAuth.priv over tbsData - *not* including any TAGs nor the KeyID and AAID. |

NOTE

Only the data in `tbsData` is included in the signature computation. All other fields are essentially unauthenticated and are treated as 'hints' only.

# 4. Processing Rules

*This section is normative.*

## 4.1 Registration Response Processing Rules for ASM

Refer to [UAFAuthnrCommands] document for more information about the TAGs and structure mentioned in this paragraph.

1. Locate authenticator using `authenticatorIndex`. If the authenticator cannot be located, then fail with `UAF_ASM_STATUS_AUTHENTICATOR_DISCONNECTED`.
2. If a user is already enrolled with this authenticator (such as biometric enrollment, PIN setup, etc. for example) then the ASM MUST request that the authenticator verifies the user.

   NOTE

   If the authenticator supports `UserVerificationToken` (see [UAFAuthnrCommands]), then the ASM must obtain this token in order to later include it with the `Register` command.

   If the user is locked out (e.g. too many failed attempts to get verified) and the authenticator cannot automatically trigger unblocking, return `UAF_ASM_STATUS_USER_LOCKOUT`.

   - If verification fails, return `UAF_ASM_STATUS_ACCESS_DENIED`
3. If the user is not enrolled with the authenticator then take the user through the enrollment process.
   - If neither the ASM nor the Authenticator can trigger the enrollment process, return `UAF_ASM_STATUS_USER_NOT_ENROLLED`.
   - If enrollment fails, return `UAF_ASM_STATUS_ACCESS_DENIED`
4. Hash the provided `RegisterIn.finalChallenge` using the authenticator-specific hash function (`FinalChallengeHash`)

   An authenticator's preferred hash function information MUST meet the algorithm defined in the

AuthenticatorInfo.authenticationAlgorithm field.

5. Generate a key pair with apropriate protection settings and mark it for use with Android Protected Confirmation, see [https://developer.android.com/training/articles/security-android-protected-confirmation](https://developer.android.com/training/articles/security-android-protected-confirmation).
6. Create a TAG_AUTHENTICATOR_ASSERTION structure containing a TAG_UAFV1_REG_ASSERTION object with the following specifics:
    1. set signature of Surrogate Basic Attestation to 0 bytes length
    2. add the Android Hardware Key Attestation extension
7. If the authenticator is a bound authenticator
    1. Store CallerID (see [UAFASM]), AppID, TAG_KEYHANDLE, TAG_KEYID and CurrentTimestamp in the ASM's database.

> NOTE
>
> What data an ASM will store at this stage depends on underlying authenticator's architecture. For example some authenticators might store AppID, KeyHandle, KeyID inside their own secure storage. In this case ASM doesn't have to store these data in its database.

8. Create a RegisterOut object
    1. Set RegisterOut.assertionScheme according to "APCV1CBOR"
    2. Encode the content of TAG_AUTHENTICATOR_ASSERTION (i.e. TAG_UAFV1_REG_ASSERTION) in base64url format and set as RegisterOut.assertion as described in section "Data Structures for APCV1CBOR".
    3. Return RegisterOut object

## 4.2 Registration Response Processing Rules for FIDO Server

Instead of skipping the assertion as described in step 6.9, follow these rules:

1. if a.assertionScheme == "APCV1CBOR" AND a.assertion.TAG_UAFV1_REG_ASSERTION contains TAG_UAFV1_KRD as first element:
    1. Obtain Metadata(AAID).AttestationType for the AAID and make sure that a.assertion.TAG_UAFV1_REG_ASSERTION contains the most preferred attestation tag specified in field MatchCriteria.attestationTypes in RegistrationRequest.policy (if this field is present).
        - If a.assertion.TAG_UAFV1_REG_ASSERTION doesn't contain the preferred attestation - it is RECOMMENDED to skip this assertion and continue with next one
    2. Make sure that a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.FinalChallengeHash == FCHash
        - If comparison fails - continue with next assertion
    3. Obtain Metadata(AAID).AuthenticatorVersion for the AAID and make sure that it is lower or equal to a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.AuthenticatorVersion.
        - If Metadata(AAID).AuthenticatorVersion is higher (i.e. the authenticator firmware is outdated), it is RECOMMENDED to assume increased risk. See sections "StatusReport

dictionary" and "Metadata TOC object Processing Rules" in [FIDOMetadataService] for more details on this.

4. Check whether `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.RegCounter` is 0 since it is not supported in this assertion scheme.
   - If `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.RegCounter` is non-zero, this assertion might be skipped and processing will continue with next one
5. Make sure `a.assertion.TAG_UAFV1_REG_ASSERTION` contains an object of type `ATTESTATION_BASIC_SURROGATE`
   1. There is no real attestation for the AAID, so we just assume the AAID is the real one.
   2. If entry `AttestationRootCertificates` for the AAID in the metadata is not empty - continue with next assertion (as the AAID obviously is expecting a different attestation method).
   3. Verify that extension "fido.uaf.android.key_attestation" is present and check whether it is positively verified according to its server processing rules as specified [UAFRegistry].
      - If verification fails – continue with next assertion
   4. Verify that the attestation statement included in that extension includes the flag `TRUSTED_CONFIRMATION_REQUIRED` indicating that the key will be restricted to sign valid transaction confirmation assertions (see https://developer.android.com/training/articles/security-key-attestation and https://developer.android.com/training/articles/security-android-protected-confirmation).
      - If verification fails – continue with next assertion
   5. Mark assertion as positively verified
6. Extract `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.PublicKey` into PublicKey, `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.KeyID` into KeyID, `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.SignCounter` into SignCounter, `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.TAG_ASSERTION_INFO.authenticatorVersion` into AuthenticatorVersion, `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.TAG_AAID` into AAID.

## 4.3 Authentication Response Generation Rules for ASM

See [UAFASM] for details of the ASM API.

1. if this is a bound authenticator, verify `callerid` against the one stored at registration time and return `UAF_ASM_STATUS_ACCESS_DENIED` if it doesn't match.
2. The ASM MUST request the authenticator to verify the user.
3. Hash the provided `AuthenticateIn.finalChallenge` using the preferred authenticator-specific hash function (`FinalChallengeHash`).

   The authenticator's preferred hash function information MUST meet the algorithm defined in the `AuthenticatorInfo.authenticationAlgorithm` field.

4. If AuthenticateIn.keyIDs is not empty,
   1. If this is a bound authenticator, then look up ASM's database with `AuthenticateIn.appID` and `AuthenticateIn.keyIDs` and obtain the KeyHandles associated with it.
      - Return `UAF_ASM_STATUS_KEY_DISAPPEARED_PERMANENTLY` if the related key disappeared permanently from the authenticator.

        `UAF_ASM_STATUS_ACCESS_DENIED`

Return if no entry has been found.

2. If this is a roaming authenticator, then treat `AuthenticateIn.keyIDs` as KeyHandles

5. If AuthenticateIn.keyIDs is empty, lookup all KeyHandles matching this request.
6. If multiple KeyHandles exist that match this request, show the related distinct usernames and ask the user to choose a single username. Remember the KeyHandle related to this key.
7. Call `ConfirmationPrompt.Builder` and pass the transactionText as parameter to method `setPromptText` see also [https://developer.android.com/training/articles/security-android-protected-confirmation](https://developer.android.com/training/articles/security-android-protected-confirmation).
8. Pass the `FinalChallengeHash` as parameter to method `setExtraData`, see also [https://developer.android.com/training/articles/security-android-protected-confirmation](https://developer.android.com/training/articles/security-android-protected-confirmation)
9. Call `build` method of the ConfirmationPrompt and then call method `presentPrompt` providing an appropriate callback that will sign the `dataThatWasConfirmed` with the key identified by the KeyHandle remembered earlier.
10. Create `TAG_APCV1CBOR_AUTH_ASSERTION` structure.
    1. Copy the serialized `dataThatWasConfirmed` CBOR object into field `tbsData`.
    2. Copy `AAID` and `KeyID` into the respective TLV fields.
    3. Copy `signature` into the `TAG_SIGNATURE` field.
11. Create the `AuthenticateOut` object
    1. Set `AuthenticateOut.assertionScheme` to "APCV1CBOR"
    2. Encode the content of `TAG_APCV1CBOR_AUTH_ASSERTION` in base64url format and set as `AuthenticateOut.assertion`
    3. Return the `AuthenticateOut` object

The authenticator metadata statement MUST truly indicate the type of transaction confirmation display implementation. Typically the "Transaction Confirmation Display" flag will be set to `TRANSACTION_CONFIRMATION_DISPLAY_ANY` (bitwise) or `TRANSACTION_CONFIRMATION_DISPLAY_PRIVILEGED_SOFTWARE`.

## 4.4 Authentication Response Processing Rules for FIDO Server

Instead of skipping the assertion according to step 6.6. in section 3.5.7.5 [UAFProtocol], follow these rules:

> NOTE
>
> The extraData in tbsData.dataThatWasConfirmed is the finalChallengeHash as computed by the ASM. The promptText in tbsData.dataThatWasConfirmed is the AuthenticateIn.Transaction.content value. AuthenticateIn.Transaction.contentType is "text/plain".

1. if `a.assertionScheme` == "APCV1CBOR" AND `a.assertion` startes with a valid CBOR structure as defined in section [3.2 Authentication Assertion](#), then
    1. set `tbsData` to the CBOR object contained in `a.assertion.tbsData`.
    2. Verify the AAID against the AAID stored in the user's record at time of Registration.
        - If comparison fails – continue with next assertion
    3. Locate `UAuth.pub` associated with (`a.assertion.AAID`, `a.assertion.KeyID`) in the user's record.

- If such record doesn't exist - continue with next assertion

4. Locate authenticator specific authentication algorithms from authenticator metadata (field `AuthenticationAlgs`)
5. If `fcp` is of type FinalChallengeParams, then hash `AuthenticationResponse.FinalChallengeParams` using the 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 ALG_SIGN.
   - `FCHash = hash(AuthenticationResponse.FinalChallengeParams)`
6. If `fcp` is of type ClientData, then hash `AuthenticationResponse.fcParams` using hashing algorithm specified in `fcp.hashAlg`.
   - `FCHash = hash(AuthenticationResponse.fcParams)`
7. Make sure that `tbsData.dataThatWasConfirmed.extraData` == `FCHash`
   - If comparison fails – continue with next assertion
8. Make sure there is a transaction cached on Relying Party side in the list `cachedTransactions`.
   - If not – continue with next assertion

> NOTE
>
> The `promtpText` included in this `AuthenticationResponse` must match the transaction content specified in the related `AuthenticationRequest`. As FIDO doesn't mandate any specific FIDO Server API, the transaction content could be cached by any relying party software component, e.g. the FIDO Server or the relying party Web Application.

9. Make sure that `tbsData.dataThatWasConfirmed.promptText` is included in the list `cachedTransactions`
   - If it's not in the list – continue with next assertion
10. Use the `UAuth.pub` key found in step 1.2 and the appropriate authentication algorithm to verify the signature `a.assertion.Signature` of the to-be-signed object `tbsData`.
    1. If signature verification fails – continue with next assertion

# 5. Example for FIDO Metadata Statement

*This section is non-normative.*

This example Authenticator has the following characteristics:

- Authenticator implementing transaction confirmation display using TrustedUI (i.e. in TEE)
- Leveraging TEE backed key store and user verification
- Only fingerprint based user verification is implemented - no alternative password

EXAMPLE 1: MetadataStatement for UAF Authenticator

```
{
  "description": "FIDO Alliance Sample UAF Authenticator supporting Android Protected
Confirmation",
```

```
    "aaid": "1234#5679",
    "authenticatorVersion": 2,
    "upv": [
      { "major": 1, "minor": 2 }
    ],
    "assertionScheme": "APCV1CBOR",
    "authenticationAlgorithm": 1,
    "publicKeyAlgAndEncoding": 256,
    "attestationTypes": [15880],
    "userVerificationDetails": [
      [{
        "userVerification": 2,
        "baDesc": {
          "selfAttestedFAR": 0.00002,
          "maxRetries": 5,
          "blockSlowdown": 30,
          "maxTemplates": 5
        }
      }]
    ],
    "keyProtection": 6,
    "isKeyRestricted": true,
    "matcherProtection": 2,
    "cryptoStrength": 128,
    "operatingEnv": "TEEs based on ARM TrustZone HW",
    "attachmentHint": 1,
    "isSecondFactorOnly": false,
    "tcDisplay": 5,
    "tcDisplayContentType": "text/plain",
    "attestationRootCertificates": [ ],
    "supportedExtensions": [{
        "id": "fido.uaf.android.key_attestation",
        "data": "{ \"attestationRootCertificates\": [
          \"MIICPTCCAeOgAwIBAgIJAOuexvU3Oy2wMAoGCCqGSM49BAMCMHsxIDAeBgNVBAMM
            F1NhbXBsZSBBdHRlc3RhdGlvbiBSb290MRYwFAYDVQQKDA1GSURPIEFsbGlhbmNl
            MREwDwYDVQQLDAhVQUYgVFdHLDESMBAGA1UEBwwJUGFsbyBBbHRvMQswCQYDVQQI
            DAJDQTELMAkGA1UEBhMCVVMwHhcNMTQwNjE4MTMzMzMyWhcNNDExMTAzMTMzMzMy
            WjB7MSAwHgYDVQQDDBdTYW1wbGUgQXR0ZXN0YXRpb24gUm9vdDEWMBQGA1UECgwN
            RklETyBBbGxpYW5jZTERMA8GA1UECwwIVUFGIFRXRywxEjAQBgNVBAcMCVBhbG8g
            QWx0bzELMAkGA1UECAwCQ0ExCzAJBgNVBAYTAlVTMFkwEwYHKoZIzj0CAQYIKoZI
            zj0DAQcDQgAEH8hv2D0HXa59/BmpQ7RZehL/FMGzFd1QBg9vAUpOZ3ajnuQ94PR7
            aMzH33nUSBr8fHYDrqOBb58pxGqHJRyX/6NQME4wHQYDVR0OBBYEFPoHA3CLhxFb
            C0It7zE4w8hk5EJ/MB8GA1UdIwQYMBaAFPoHA3CLhxFbC0It7zE4w8hk5EJ/MAwG
            A1UdEwQFMAMBAf8wCgYIKoZIzj0EAwIDSAAwRQIhAJ06QSXt9ihIbEKYKIjsPkri
            VdLIgtfsbDSu7ErJfzr4AiBqoYCZf0+zI55aQeAHjIzA9Xm63rruAxBZ9ps9z2XN
            lQ==\"] }",
        "fail_if_unknown": false
      }],
    "icon": "data:image/png;base64,
      iVBORw0KGgoAAAANSUhEUgAAAE8AAAAvCAYAAACiwJfcAAAAAXNSR0IArs4c6QAAAARnQU1BAACx
      jwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAAahSURBVGhD7Zr5bxRlGMf9KzTB8AM/YEhE2W7p
      QZcWKKBclSpHATlELARE7kNECCA3FkWK0CKKSCFIsKBcgVCDWGNESdAYidwgggJBiRiMhFc/4wy8
      884zu9NdlnGTfZJP2n3nO++88933fveBBx+PqCzJkTUvBbLmpUDWvBTImpcCSZvXLCdX9R05Sk19
      bb5atf599fG+/erA541q47aP1LLVa9SIyVNUi8Ii8d5kGTsi30NFv7ai9n7QZPMwbdys2erU2XMq
      Udy8+ZcaNmGimE8yXN3RUd3a18nF0fUlovZ+0CTzWpd2Vj+eOm1bEyy6Dx4i5pUMGWveo506q227
      dtuWBIuffr6oWpV0FPNLhow1751Nm21LvPH3rVtWjfz66Lfql8tX7FRl9YFSXsmSseb9ceOGbYk7
      MNUcGPg8ZsbMe9rfQUaaV/JMX9sqdzDCSvp0kZHmTZg9x7bLHcMnThb16eJ+mVfQq8yaUZQNG64i
      XZ+0/kq6uOZFO0QtatdWKfXnRQ99Bj91R5OIFnk54jN0mkUiqlO3XDW+Ml+98mKB6tW7rWpZcPc+
      0zg4tLrYlUc86E6eGDjIMubVpcusearfgIYGRk6brhZVr/JcHzooL7550jedLExopWcApi2ZUqhu
      7JLvrVsQU81zkzOPeemMRYvVuQsX7PbiDQY5JvZonftK+1VY8H9utx530h0ob+jmRYqj6ouaYvEe
      nW/WlYjp8cwbMm682tPwqW1R4tj/2SH13IRJYl4moZvXpiSqDr7dXtQHxa/PK3/+BWsK1dTgHu6V
      8tQJ3bwFkwpFrUOQ50s1r3levm8zZcq17+BBaw7K8lEK5qzkYeark9A8p7P3GzDK+nd3DQow+6UC
      8SVN82iuv38im7NtaXtV1CVq6Rgw4pksmbdi3bu2De7YfaBBxcqfvqPrUjFQNTQ22lfdUVVT68rT
      JKF5DnSmUjgdqg4mSS9pmsfDJR3G6ToH0iW9aV7LWLHYXKllTDt0LTAtkYIaamp1QjVv++uyGUxV
      dJ0DNVXSm+b1qRxpl84ddfX1Lp1O/d69tsod0vs5hGre9xu8o+fpLR1cGhNTD6Z57C9KMWXefJdO
      Z94bb9oqd1ROnS7qITTzHimMqivbO3g0DdVyk3WQBhBztK35YKNdOnc8O3acS6fDZFgKaXLsEJp5
      rdrliBqp89cJcs/m7Tvs0rkjGfN4b0kPoZn3UJuIOrnZ22yP1fmvUx+O5gSqebV1m+zSuYNVhq7T
      WbDiLVvljplLlop6CLXP+2qtvGLIL/1vimISdMBgzSoFZyu6Tqd+jzxgsPaV9BCqee/NjYk6v6lK
      9cwiUc/STtf1HDpM3b592y7h3Thx5ozK69HLpYWuAwaqS5cv26q7ceb8efVYaReP3iFU8zj1knSw
      ZXHMmnCjY0Ogalo7UQfSCM3qQQr2H/XFP7ssXx45Yl91ByeCep4moZoH+1fG3xD4tT7x8kwyj8nw
      b9ev26V0B6d+7H4zKvudAH537FjqyzOHdJnHEuzmXq/WjxObvNMbv7nhywsX2aVsWtC8+48aLeap
      E7p5wKZi0A2AQRV5nvR4E+uJc+b61kApqInxBgmd/4V5QP/mt18HDC7sRHftmeu5lmhV0rn/ALX2
      32bqd4BFnDx7Vi1cWS2uff0IbB47qexxmUj9QutYjupd3tYD6abWBBMrh+apNbOKrNF1+ugCa4ri
      XGfwMPPtViavhU3YMOAAnuUb/R07L0yOSeOadE88ApsXFGff30ynhlJgM51CU6vN9EzgnpvHBFUy
      iVraePiwJ53DF5ZTZnomENg85kNUd2oJi2Wpr4OmmkfN4x4zHfiVFc8Dv8NzuhNqOidilGvA6DGu
      eZwO78AAQn6ciEk6+rw5VcvjvqNDYPOoIUwaKShrxAuXLlkH4aYuGfMYDc10WF5Ta31hPJOfcUhr
      U/JlINi6c6elRYdBpo6++Yfjx61lGNfRm4MD5rJ1j3FoGHnjDSBNarYUgMLyMszKpb7tXpoHfPs8
      h3Wp1LzNfNk54XxC1wDGUmYzXYefh6z/cKtVm4EBxa9VQGDzYr3LrUMRjHEKkk7zaFKYQA2hGQU1
      z+85NFWpXDrkz3vx10GqxQ6BzeNboBk5n8k4nebRh+k1hWfxTF0D1EyWUs5nv+dgQqKaxzuCdE0i
      sHl02NQ8ah0mXr12La3m0f9wik9+wLNTMY/86MPo8yi31OfxmT6PWoqG9+DZukYna56mSZt5WWSy
      5qVA1rwUyJqXAlnzkiai/gHSD7RkTyihogAAAABJRU5ErkJggg=="
}
```

# A. References

## A.1 Normative references

**[FIDOGlossary]**
R. Lindemann; D. Baghdasaryan; B. Hill; J. Hodges. *FIDO Technical Glossary*. Review Draft. URL: https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-glossary-v2.0-id-20180227.html

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

**[UAFASM]**
D. Baghdasaryan; J. Kemp; R. Lindemann; B. Hill; R. Sasson. *FIDO UAF Authenticator-Specific Module API*. Review Draft. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-ps-20201020/fido-uaf-asm-api-v1.2-ps-20201020.html

**[UAFAuthnrCommands]**
D. Baghdasaryan; J. Kemp; R. Lindemann; R. Sasson; B. Hill; J. Hodges; K. Yang. *FIDO UAF Authenticator Commands*. Review Draft. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-ps-20201020/fido-uaf-authnr-cmds-v1.2-ps-20201020.html

**[UAFProtocol]**
R. Lindemann; D. Baghdasaryan; E. Tiffany; D. Balfanz; B. Hill; J. Hodges; K. Yang. *FIDO UAF Protocol Specification v1.2*. Review Draft. URL: https://fidoalliance.org/specs/fido-uaf-v1.2-ps-20201020/fido-uaf-protocol-v1.2-ps-20201020.html

**[UAFRegistry]**
R. Lindemann; D. Baghdasaryan; B. Hill. *FIDO UAF Registry of Predefined Values*. Review Draft. URL: https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-registry-v2.0-id-20180227.html

## A.2 Informative references

**[FIDOMetadataService]**
R. Lindemann; B. Hill; D. Baghdasaryan. *FIDO Metadata Service*. Review Draft. URL: https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-metadata-service-v2.0-id-20180227.html