FIDO UAF 认证器特定模块 API V1.0

FIDO 联盟推荐标准　2014-12-08

当前版本：

https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-asm-api-v1.0-ps-20141208.html

之前版本：

https://fidoalliance.org/specs/fido-uaf-asm-api-v1.0-rd-20141008.pdf

编写者：

达维特•巴格达萨利安（Davit Baghdasaryan）, Nok Nok Labs, Inc.

约翰•肯普（John Kemp）， FIDO 联盟

贡献者：

罗尔夫•林德曼博士（Dr. Rolf Lindemann）, Nok Nok Labs, Inc.

布拉德•希尔（Brad Hill）, 贝宝（PayPal, Inc.）

罗尼•萨森（Roni Sasson）, Discretix, Inc.

摘要

UAF 认证器通过各种各样的物理接口（SPI、USB、蓝牙……）与用户设备相连。UAF 认证器特定模块（ASM）是 UAF 认证器上层的软件接口，它为 FIDO UAF 客户端探测和获取 UAF 认证器功能以及隐藏 FIDO UAF 客户端的内部通信复杂性提供了一种标准方法。

本文档描述了 AMSs 的内部功能，定义了 UAF ASM API，并解释了 FIDO UAF 客户端应如何使用该 API。

本文档的目标受众是 FIDO 认证器和 FIDO UAF 客户端的供应商。

文档状态

本章节描述了文档发布时的状态。本文档有可能会被其它文档所取代。当前 FIDO 联盟出版物的列表以及此技术报告的最新修订可在 FIDO 联盟规范索引 上找到。网址：https://www.fidoalliance.org/specifications/.

本文档由 FIDO 联盟 作为推荐标准发布。如果您希望就此文档发表评论，请 联系我们 。欢迎所有评论。

本规范中某些元素的实现可能需要获得第三方知识产权的许可，包括（但不限于）专利权。FIDO 联盟及其成员，以及此规范的其他贡献者们不能，也不应该为任何识别或未能识别所有这些第三方知识产权的行为负责。

本 FIDO 联盟规范是“按原样”提供，没有任何类型的担保，包括但不限于，任何明确的或暗示的不侵权、适销性或者适合某一特定用途的担保。

本文档已经由 FIDO 联盟成员评审并签署成为推荐标准。这是一篇稳定的文档，可能会作为参考材料被其它文档引用。FIDO 联盟的作用是引起对规范的注意并促进其广泛的分发。

目录

1、注释 3
1.1 关键字 4
2、概览 4
2.1 编码和示例格式 5
3、ASM 请求和响应 5
3.1 请求枚举 6
3.2 状态码接口 7
3.2.1 常量 7
3.3 ASMRequest 结构 7
3.3.1 ASMRequest 结构成员 8
3.4 ASMResponse 结构 8
3.4.1 ASMResponse 结构成员 9
3.5 GetInfo 请求 9
3.5.1 GetInfoOut 结构 10
3.5.1.1 GetInfoOut 结构成员 10
3.5.2 AuthenticatorInfo 结构 11
3.5.2.1 AuthenticatorInfo 结构成员 11
3.6 注册请求 14

3.6.1 RegisterIn 对象 15
3.6.1.1 RegisterIn 结构成员 15
3.6.2 RegisterOut 对象 16
3.6.2.1 RegisterOut 结构成员 16
3.6.3 处理注册请求过程的详细描述 16
3.7 鉴别请求 18
3.7.1 AuthenticateIn 对象 18
3.7.1.1 AuthenticateIn 结构成员 19
3.7.2 Transaction 对象 19
3.7.2.1 Transaction 结构成员 19
3.7.3 AuthenticateOut 对象 20
3.7.3.1 AuthenticateOut 结构成员 20
3.7.4 鉴别请求过程的详细描述 20
3.8 注销请求 22
3.8.1 DeregisterIn 对象 23
3.8.1.1 DeregisterIn 结构成员 23
3.8.2 注销请求过程的详细描述 23
3.9 GetRegistrations 请求 24
3.9.1 GetRegistrationsOut 对象 24
3.9.1.1 GetRegistrationsOut 结构成员 24
3.9.2 AppRegistration 对象 25
3.9.2.1 AppRegistration 结构成员 25
3.9.3GetRegistrations 请求过程的详细描述 25
3.10 OpenSettings 请求 25
4、使用 ASM API.....26
5、在不同平台上使用 ASM API.....26
5.1 Android ASM Intent API 26
5.1.1 发现 ASMs 27
5.2 Windows ASM API.....27
6、安全性和私密性 30
6.1 KHAccessToken.....32
6.2 ASM APIs 的访问控制 34
A.参考文献 35
A.1 参考规范 35
A.2 参考资料 36

1、注释

类型名称、属性名称和元素名称用代码形式书写。

字符串文本包含在双引号“”内，比如“UAF-TLV”。

公式中用 “|” 来表示按字节串联操作。

DOM APIs 被描述为使用 ECMAScript [ECMA-262]绑定 WebIDL [WebIDL-ED]。

注释 base64url 指的是无填充（padding）版本的“对于 URL 和文件名安全的 Base64 编码表” [RFC4648] 。

根据[WebIDL-ED]，结构成员是可选的，除非它们被明确标注为必需（required）。WebIDL 的结构成员不得为空值。

除非特别声明，如果 WebIDL 的结构成员是 DOMString，则不得为空。

除非特别声明，如果 WebIDL 的结构成员是一个列表（List），则不得为一个空列表（List）。

本文档中用到的 UAF 专用术语在 FIDO 术语表[FIDOGlossary]中均有定义。

此规范中的所有的图表、示例、注释都是非规范的。

> **注释**
>
> 特定的结构成员需要遵从 FIDO 协议的要求。本文档中以 required 为标示，标注了这些词汇在 WebIDL 的定义。关键词 required 是在开发中版本[WebIDL-ED]提出，如果使用执行 WebIDL 开发程序的解析器，则可删除在 WebIDL 中的关键词 required 并通过其他方式将这些字段填满。

1.1 关键字

本文档中的关键字：“必须”，“不得”，“要求”，“将”，“将不”，“应该”，“不应该”，“建议”，“可能”，“可选”都会按照[RFC2119]的描述来解释。

2、概览

本节是非规范性的。

UAF 认证器通过各种各样的物理接口（SPI、USB、蓝牙……）与用户设备相连。

UAF 认证器特定模块（ASM）是 UAF 认证器上层的软件接口，它为 FIDO UAF 客户端探测和获取 UAF 认证器功能以及隐藏 FIDO UAF 客户端的内部通信复杂性提供了一种标准方法。

ASM 是一个提供 API 给 FIDO UAF 客户端的特定平台的软件组件，以保证客户端能发现一个或更多可用认证器并与其进行通讯。

单个认证器特定模块（ASM）可以代表多个认证器进行报告。

本文档的目标阅读者是 FIDO 认证器和 FIDO UAF 客户端的供应商。

> **注释**
>
> 平台供应商可能选择不向应用暴露本文档中定义的 ASM API，而是通过其他的 API（例如：Android KeyStore API、IOS KeyChain API）来暴露 ASM 的功能。在这种情况下，以本文档定义的方式确认底层 ASM 与 FIDO UAF 认证器的通讯是非常重要的。

FIDO UAF 协议和其各项操作在 FIDO UAF 协议标准[UAFProtocol]中均有定义。

下述简易的架构图展示了此文档中主要关注的交互行为和主体：

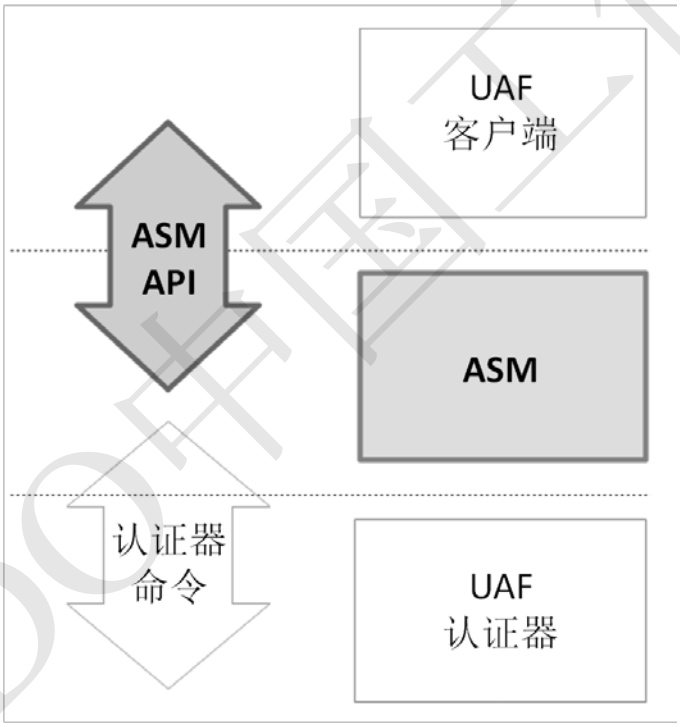

图 1 UAF ASM API 结构

2.1 编码和示例格式

ASM 请求和响应是以 WebIDL 格式显示。

3、ASM 请求和响应

本节是规范性的。

ASM API 是根据 JSON 格式[ECMA-404]的请求和响应消息来定义的。为了将请求传递到 ASM，FIDO UAF 客户端会生成一个合适的对象（如，ECMAscript），将其"字符串化（stringifies）"（也被称为“序列化”）为 JSON 格式的字符串，最后传输到 ASM。ASM 反序列化 JSON 格式的字符串，处理该请求，生成响应消息，将响应消息序列化，然后以 JSON 格式的字符串返回。

> **注释**
>
> 此文档中的 ASM 请求处理规则是假设底层认证器实施了在[UAFProtocol]中定义的“UAFV1TLV”断言方案（如，参考 TLVs 和 tags）。如果认证器支持不同的断言方案，那么相应的处理规则应该被适当的特定的断言方案规则所替代。

认证器实施者可能会生成自定义的认证器命令接口，而不是在[UAFAuthnrCommands]中定义的控制命令接口。这种实施过程并不需要严格遵循在本节描述的具体的信息传输过程。然而，

1. 控制命令接口必须要显示具有与下面表述相一致的外部特征的 ASM，这是为了让 ASM 能够对客户端的请求信息给出恰当的响应 （比如，返回合适特定情况的 UAF 状态码）。
2. 所有的认证器实施必须支持[UAFRegistry]中定义的断言方案并且必须返回相应的对象，比如 TAG_UAFV1_REG_ASSERTION 和 TAG_UAFV1_AUTH_ASSERTION。

3.1 请求枚举

WebIDL

```
enum Request {
    "GetInfo",
    "Register",
    "Authenticate",
    "Deregister",
    "GetRegistrations",
    "OpenSettings"};
```

枚举描述	
GetInfo	获取信息
Register	注册
Authenticate	鉴别
Deregister	注销
GetRegistrations	获取注册信息
OpenSettings	打开设置

3.2 状态码接口

WebIDL

```
Interface StatusCode {
    const short UAF_ASM_STATUS_OK = 0x00;
    const short UAF_ASM_STATUS_ERROR = 0x01;
    const short UAF_ASM_STATUS_DENIED = 0x02;
    const short UAF_ASM_STATUS_CANCELLED = 0x03;}
```

3.2.1 常量

UAF_ASM_ATSTUS_OK类型为 short

未遇到错误状态。

UAF_ASM_ATSTUS_ERROR类型为 short

处理过程中有未知差错。

UAF_ASM_ATSTUS_DENIED类型为 short

访问请求被拒绝。

UAF_ASM_ATSTUS_CANCELLED类型为 short

表示用户明确地取消请求。

3.3 ASMRequest 结构

所有 ASM 请求均被描绘成 ASMRequest 对象。

```
WebIDL
    dictionary ASMRequest {
        required Request        requestType;
        Version                 asmVersion;
        unsigned short          authenticatorIndex;
        object                  args;
        Extension[]             exts;
    };
```

3.3.1 ASMRequest 结构成员

requestType类型为 required Request

请求类型。

asmVersion类型为 Version

此请求使用的ASM版本信息。Version 结构的定义可查看[UAFProtocol]。ASM 的版本必须是 1.0（比如，主要版本为 1，次要版本为 0）。

authenticatorIndex类型为 unsigned short

参阅 GetInfo 请求获取更多细节。不得为 GetInfo 请求设置 authenticatorIndex 字段。

args类型为 object

特定的请求参数。如果设定该参数则这个属性可能会是下列类型中的一种：

- RegisterIn
- AuthenticateIn
- DeregisterIn

exts类型为 Extension 数组

UAF 扩展列表。Extension 结构的定义可以查看 [UAFProtocol]。

3.4 ASMResponse 结构

所有 ASM 响应均表示成 ASMResponse 对象。

```
WebIDL
    dictionary ASMResponse {
        required short      statusCode;
        object              responseData;
        Extension[]         exts;
    };
```

3.4.1 ASMResponse 结构成员

statusCode类型为 required short

必须包含一个 StatusCode 接口中定义的值。

responseData类型为 object

特定的请求响应数据。这个属性必须是下面类型的一种：

- GetInfoOut
- RegisterOut
- AuthenticateOut
- GetRegistrationOut

exts类型为 Extension 数组

UAF 扩展列表。Extension 结构的定义可以查看 [UAFProtocol]。

3.5 GetInfo 请求

返回可用认证器相关的信息。

1、 枚举出此 ASM 支持的所有认证器。

2、 收集关于它们的所有信息。

3、 给它们分配索引(authenticatorIndex)。

4、 向访问者返回信息。

> **注释**
>
> 在可能的情况下，authenticatorIndex 应该是一个可以唯一地标识一个认证器的持久标识符，即使它反复地断开和重新连接。这样可以避免一组可用认证器集合的 GetInfo 请求和随后的 ASM 请求之间变化时产生的混淆。并且

> 为了获得更好的用户体验，允许 FIDO 客户端对可移动的认证器的信息进行缓存。

针对 GetInfo 请求，下列 **ASMRequest** 成员必须具有如下的值。其它的 **ASMRequest** 成员应该被省略：

- ASMRequest.requestType 必须被设定到 GetInfo

针对 GetInfo 响应，下列 **ASMResponse**t 成员必须具有如下的值。其它的 **ASMResponse** 成员应该被省略:

- ASMResponse.statusCode 必须具有下列值中的一个：
  - UAF_ASM_STATUS_OK
  - UAF_ASM_STATUS_ERROR
- ASMResponse.responseData 必须是 GetInfoOut 类型的对象。

3.5.1 GetInfoOut 结构

WebIDL

```
dictionary GetInfoOut {
    required AuthenticatorInfo[]      Authenticators;
    };
```

3.5.1.1 GetInfoOut 结构成员

Authenticators类型为 required AuthenticatorInfo 数组

当前 ASM 报告的认证器列表。可能为一个空列表。

3.5.2 AuthenticatorInfo 结构

WebIDL

```
dictionary AuthenticatorInfo {
    required unsigned short                 authenticatorIndex;
    required Version[]                      asmVersions;
    required Boolean                        isUserEnrolled;
    required Boolean                        hasSettings;
    required AAID                           aaid;
    required DOMString                      assertionScheme;
    required unsigned short                 authenticationAlgorithm;
    required unsigned short[]               attestationTypes;
    required unsigned long                  userVerification;
    required unsigned short                 keyProtection;
    required unsigned short                 matcherProtection;
    required unsigned long                  attachmentHint;
    required Boolean                        isSecondFactorOnly;
    required Boolean                        isRoamingAuthenticator;
    required DOMString[]                    supportedExtensionIDs;
    required unsigned short                 tcDisplay;
    DOMString                               tcDisplayContentType;
    DisplayPNGCharacteristicsDescriptor[]   tcDisplayPNGCharacteristics;
    DOMString                               title;
    DOMString                               description;
    DOMString                               icon;
};
```

3.5.2.1 AuthenticatorInfo 结构成员

authenticatorIndex类型为 required unsigned short

认证器索引。该值在一个 ASM 报告的所有认证器的范围内是唯一的，索引指代其中某一个认证器。UAF 客户端使用这个索引将后续的请求关联到合适的认证器。

asmVersions类型为 required Version 数组

可以使用该认证器的 ASM 版本列表。关于 Version 结构的定义可以查看文档[UAFProtocol]。

isUserEnrolled类型为 required boolean

表示用户是否使用该认证器注册。没有用户校验技术的认证器必须返回 true。支持按操作系统用户配置不同的所有绑定的认证器必须报告当前操作系统用户的注册状态。

hasSettings类型为 required boolean

布尔值表明认证器是否有自己的设置。如果有自己的设置，FIDO UAF 客户端即可以发送一个 OpenSettings 请求来启动这些设置。

aaid类型为 required AAID

“认证器鉴证标识符”（AAID），表示认证器的类型和批次。关于 AAID 结构的定义信息可以查阅文档[UAFProtocol] 。

assetionScheme类型为 required DOMString

认证器用来验证数据和签名的断言方案。

断言方案标识符在 UAF 协议标准[UAFProtocol]中定义。

authenticationAlgorithm类型为 required unsigned short

表示认证器所采用的鉴别算法。鉴别算法标识符在[UAFRegistry] 中有定义，以 UAF_ALG 为前缀。

attestationTypes类型为 required unsigned short 数组

表示认证器支持的鉴证类型。鉴证类型标签定义在[UAFRegistry]中，以 TAG_ATTESTATION 为前缀。

userVerification类型为 required unsigned long

一组位标志，用来表示认证器支持的用户校验方法。这些值由[UAFRegistry]中的 USER_VERIFY 常量定义。

keyProtection类型为 required unsigned short

一组位标志，用来表示认证器所使用的密钥保护方案。这些值由[UAFRegistry]中的 KEY_PROTECTION 常量定义。

matcherProtection类型为 required unsigned short

一组位标志，用来表示认证器所采用的匹配器保护方案。这些值由[UAFRegistry]中的 MATCHER_PROTECTION 常量定义。

attachmentHint类型为 required unsigned long

一组位标志，用来表示认证器目前是如何连接到托管 FIDO UAF 客户端软件的系统的。这些值由[UAFRegistry]中的 ATTACHMENT_HINT 常量定义。

> 注释
>
> 由于认证器的连接状态和拓扑结构可能是短暂的，这些值仅是那些被服务器提供的策略用于指导用户体验的提示，例如：选择已连接并且已经准备好用来做鉴别、或者可确认低价值交易的设备，而不是一个更安全但需要耗费大量用户精力的设备。这些值不反映在认证器的元数据中，也不能被依赖方依赖，尽管一些类型的认证器可能会提供具有相似语义的经鉴证的度量作为 UAF 协议消息的一部分。

isSecondFactorOnly类型为 required boolean

表示该认证器是否仅可以被作为第二因子使用。

isRoamingAuthticator类型为 required boolean

表示该认证器是否是漫游认证器。

supportedExtensionIDs类型为 required DOMString 数组

支持 UAF 扩展 ID 列表。该列表可能为空。

tcDisplay类型为 required unsigned short

一组位标志表示认证器交易确认显示的可用性和类型。这些值由[UAFRegistry]中的 TRANSACTION_CONFIRMATION_DISPLAY 常量定义。

如果认证器不支持交易确认类型，那么该值必须是 0。

tcDisplayContentType类型为 DOMString

[UAFAuthnrMetadata]支持的交易内容类型。

如果支持交易确认，例如 tcDisplay 是非零的，则该值必须存在。

tcDisplayPNGCharacteristics类型为 DisplayPNGCharacteristicsDescriptor 数组

支持的交易可移植网络图形（PNG）类型[UAFAuthnrMetadata]。

DisplayPNGCharacteristicsDescriptor 结构的定义，请见[UAFAuthnrMetadata]。

如果支持交易确认，例如 tcDisplay 是非零的，则该列表必须存在。

title类型为 DOMString

为认证器提供的一个用户可读的标题。它应该本地化为当前区域设置。

> **注释**
>
> 如果 ASM 不返回一个标题，FIDO UAF 客户端必须给当前调用的 App 提供一个标题。可参阅 [UAFAppAPIAndTransport]中的“认证器接口”。

description类型为 DOMString

表示认证器显示的用户可读的较长的说明。

> **注释**
>
> 这段文本应该本地化为当前区域设置。
>
> 该文本意在呈现给用户。它可能会偏离认证器的元数据声明中对于认证器的特定描述[UAFAuthnrMetadata]。
>
> 如果 ASM 没有返回一个说明消息，FIDO UAF 客户端会给调用程序一个说明。详细信息可见[UAFAppAPIAndTransport]中的“认证器接口（Authenticator interface）”部分。

icon类型为 DOMString

便携网络图像（PNG）格式图像文件表示 data:url[RFC2397]编码的图标。

> **注释**
>
> 如果 ASM 没有返回一个图标，FIDO UAF 客户端将给调用程序一个默认的图标。
>
> 详细信息可见[UAFAppAPIAndTransport]中的“认证器接口（Authenticator interface）”部分。

3.6 注册请求

验证用户并返回认证器生成的注册断言。

对于注册请求，下列 **ASMRequest** 成员必须具有如下值。其它的 **ASMRequest** 成员应该被省略。

- ASMRequest.requestType 必须被设置为 Register
- ASMRequest.asmVersion 必须被设置为所需的版本
- ASMRequest.authenticatorIndex 必须设置为目标认证器索引

- ASMRequest.args 必须设置为 RegisterIn 类型的对象

对于注册响应，下列 **ASMResponse** 成员必须具有如下值。其它的 **ASMResponse** 成员应该被省略。

- ASMResponse.statusCode 必须有下列值中的一个：
  - UAF_ASM_STATUS_OK
  - UAF_ASM_STATUS_ERROR
  - UAF_ASM_STATUS_ACCESS_DENIED
  - UAF_ASM_STATUS_USER_CANCELLED
- ASMResponse.responseData 必须是 RegisterOut 类型的对象

3.6.1 RegisterIn 对象

WebIDL

```
dictionary RegisterIn {
    required DOMString          appID;
    required DOMString          username;
    required DOMString          finalChallenge;
    required unsigned short     attestationType;
};
```

3.6.1.1 RegisterIn 结构成员

appID类型为 required DOMString

FIDO 服务器应用程序标识。

username类型为 required DOMString

用户可读的用户帐户名。

finalChallenge类型为 required DOMString

base64url 编码的挑战数据 [RFC4648]。

attestationType类型为 required unsigned short

单一请求鉴证类型。

3.6.2 RegisterOut 对象

WebIDL

```
dictionary RegisterOut {
    required DOMString          Assertion;
    required DOMString          assertionScheme;
};
```

3.6.2.1 RegisterOut 结构成员

assertion类型为 required DOMString

FIDO UAF 认证器注册断言，base64url 编码。

assertionScheme类型为 required DOMString

断言方案。

断言方案标识符定义在 UAF 标准协议[UAFProtocol]中。

3.6.3 处理注册请求过程的详细描述

查阅[UAFAuthnrCommands] 文档可获得更多关于 TAGs 和本章节提及的架构的相关信息。

1、 使用 authenticatorIndex 来定位认证器。如果认证器不能被定位，那么返回 UAF_ASM_STATUS_ERROR。

2、 如果用户已经在认证器注册过（比如，已采集生物特征、设置了 PIN），那么 ASM 必须请求认证器验证该用户。

> 注释
>
> 如果认证器支持 UserVerificationToken（请见文档[UAFAuthnrCommands]），那么 ASM 必须获取到该令牌以便在接下来的 Register 命令中包含它。

- 如果校验失败，返回 UAF_ASM_STATUS_ACCESS_DENIED。

3、 如果用户没有在认证器注册，那么用户指引到注册流程。

- 如果注册失败，返回 UAF_ASM_STATUS_ACCESS_DENIED。

4、 生成 KHAccessToken（详情请查阅 KHAccessToken）。

5、 使用特定认证器的哈希函数（FinalChallengeHash）来哈希提供的 RegisterIn.finalChallenge。

认证器优选的哈希函数必须满足 AuthenticatorInfo.authenticationAlgorithm 字段中定义的算法。

6、 创建一个 TAG_UAFV1_REGISTER_CMD 结构并传给认证器。

- 复制 FinalChallengeHash, KHAccessToken, RegisterIn.Username, UserVerificationToken, RegisterIn.AppID, RegisterIn.AttestationType 。
  - 根据 AuthenticatorType，一些参数可能是可选的。参阅 [UAFAuthnrCommands]获取更多的关于认证器类型和要求的参数的信息。

7、 调用命令，接收响应。

8、 解析 TAG_UAFV1_REGISTER_CMD_RESP。

- 解析 TAG_AUTHENTICATOR_ASSERTION 的内容（如：TAG_UAFV1_REG_ASSERTION）并提取 TAG_KEYID。

9、 如果认证器是绑定认证器。

- 将 CallerID, AppID, TAG_KEYHANDLE, TAG_KEYID 和 CurrentTimestamp 存储在 ASM 的数据库中。

> **注释**
>
> 在该阶段认证器特定模块存储的什么样的数据是由认证器的底层结构决定。比如，一些认证器可能会存储 AppID，KeyHandle，KeyID 在他们自己的安全存储中。在此种情况下，ASM 不需要在数据库中存储这些数据了。

10、 创建一个 RegisterOut 对象。

1) 根据 AuthenticatorInfo.assertionScheme 设定 RegisterOut.assertionScheme。

2) 以 base64url 格式编码 TAG_AUTHENTICATOR_ASSERTION(比如 TAG_UAFV1_REG_ASSERTION)的内容 ，并设定为 RegisterOut.assertion 。

3) 返回 RegisterOut 对象。

3.7 鉴别请求

核实用户并返回认证器生成的鉴别断言。

对于鉴别请求，下列 **ASMRequest** 成员必须具有如下值。其它的 **ASMRequest** 成员应该被省略。

- ASMRequest.requestType 必须被设置为 Authenticate
- ASMRequest.asmVersion 必须被设置为所需的版本
- ASMRequest.authenticatorIndex 必须设置被为目标认证器索引
- ASMRequest.args 必须被设置为 AuthenticateIn 类型的对象

对于鉴别响应，下列 **ASMResponse** 成员必须具有如下值。其它的 **ASMResponse** 成员应该被省略。

- ASMResponse.statusCode 必须有下列值中的一个：
  - UAF_ASM_STATUS_OK
  - UAF_ASM_STATUS_ERROR
  - UAF_ASM_STATUS_ACCESS_DENIED
  - UAF_ASM_STATUS_USER_CANCELLED
- ASMResponse.responseData 必须是 AuthenticateOut 类型的对象

3.7.1 AuthenticateIn 对象

WebIDL

```
dictionary AuthenticateIn {
    required DOMString      appID;
    DOMString[]             keyIDs;
    required DOMString      finalChallenge;
    Transaction[]           transaction;
};
```

3.7.1.1 AuthenticateIn 结构成员

appID类型为 required DOMString

appID 字符串。

keyIDs类型为 DOMString 数组

base64url[RFC4648]编码的 keyIDs。

finalChallenge类型为 required DOMString

base64url[RFC4648]编码的挑战数据。

transaction类型为 Transaction 数组

用户确认的交易数据的数组。如果提供了多个交易结构，ASM 必须选择一个最符合当前显示特性的。

> **注释**
>
> 举例来说，这可能取决于用户的设备在交易进行的那一刻是水平还是垂直放置的。

3.7.2 Transaction 对象

WebIDL

```
dictionary Transaction {
    required DOMString                      contentType;
    required DOMString                      content;
    DisplayPNGCharacteristicsDescriptor     tcDisplayPNGCharacteristics;
};
```

3.7.2.1 Transaction 结构成员

contentType类型为 required DOMString

包含认证器元数据中描述的其支持的 MIME 类型（见[UAFAuthnrMetadata]）。

content类型为 required DOMString

包含 base64url 编码的[RFC4648]交易内容，通过 contentType 展示给用户。

tcDisplayPNGCharacteristics类型为 DisplayPNGCharacteristicsDescriptor

交易内容满足 PNG 特点。DisplayPNGCharacteristicsDescriptor 结构的定义见[UAFAuthnrMetadata]。

3.7.3 AuthenticateOut 对象

WebIDL

```
dictionary AuthenticateOut {
    required DOMString          Assertion;
    required DOMString          assertionScheme;
};
```

3.7.3.1 AuthenticateOut 结构成员

assertion类型为 required DOMString

认证器 UAF 鉴别断言。

assertionScheme类型为 required DOMString

断言方案。

3.7.4 鉴别请求过程的详细描述

查阅[UAFAuthnrCommands] 文档可获得更多的关于 TAGs 和本章节涉及的结构的信息。

1. 使用 authenticatorIndex 来定位认证器。
2. 如果用户已经在认证器注册过（比如：生物登记法，PIN 设置），返回 UAF_ASM_STATUS_ACCESS_DENIED。
3. ASM 必须请求认证器验证该用户。
   - 如果验证失败，返回 UAF_ASM_STATUS_ACCESS_DENIED。

   > 注释
   > 如果认证器支持 UserVerificationToken（文档 [UAFAuthnrCommands] ），ASM 必须包含该特征以便在接下来转到 Sign 命令。
4. 生成 KHAccessToken（查阅 KHAccessToken）。
5. 使用特定认证器的哈希函数（FinalChallengeHash）来哈希提供的 RegisterIn.finalChallenge。

   认证器优选的哈希函数必须满足

AuthenticatorInfo.authenticationAlgorithm 字段中定义的算法。

6. 如果这是第二因子认证器并且 AuthenticateIn.keyIDs 为空，那么返回 UAF_ASM_STATUS_ACCESS_DENIED。
7. 如果 AuthenticateIn.keyIDs 不为空
   - 如果认证器为绑定认证器，使用 AuthenticateIn.appID 和 AuthenticateIn.keyIDs 来查阅 ASM 数据库，获得相应的 KeyHandles。
     - 如果没有查到相关条目，返回 UAF_ASM_STATUS_ACCESS_DENIED。
   - 如果认证器是漫游认证器，那么就把 AuthenticateIn.keyIDs 看作 KeyHandles。
8. 创建 TAG_UAFV1_SIGN_CMD 结构并把它传递给认证器。
   - 复制 AuthenticateIn.AppID, AuthenticateIn.Transaction.content（如果不为空），FinalChallengeHash, KHAccessToken, UserVerificationToken, KeyHandles。
     - 根据认证器类型，一些参数是可选的。查阅 [UAFAuthnrCommands]获取关于认证器类型和相应参数的更多信息。
     - 如果提供了多个交易结构，ASM 必须选择一个最符合当前显示特性的。

       注释

       举例来说，这可能取决于用户的设备在交易进行的那一刻是水平还是垂直放置的。

     - 在传递到认证器之前将 base64URL 编码的 AuthenticateIn.Transaction.content 解码。
9. 调用命令，接收响应。
10. 解析 TAG_UAFV1_REGISTER_CMD_RESP。
    - 如果是第一因子的认证器并且响应含有 TAG_USERNAME_AND_KEYHANDLE，然后:

1) 从 TAG_USERNAME_AND_KEYHANDLE 字段中提取出用户名。
2) 如果有两个相同的用户名，选取最近注册的那个。
3) 显示剩下的不同的用户名，让用户从中选择一个。
4) 将 TAG_UAFV1_SIGN_CMD.KeyHandles 设置成和选取的用户名相关联的单独 KeyHandle。
5) 转到步骤 8，发送新的 TAG_UAFV1_SIGN_CMD 命令。

11. 创建对象 AuthenticateOut。
    1) 将 AuthenticateOut.assertionScheme 设置为 AuthenticatorInfo.assertionScheme。
    2) 将 TAG_AUTHENTICATOR_ASSERTION 的内容（如 TAG_UAFV1_AUTH_ASSERTION）以 base64url 形式编码，并设置为 AuthenticateOut.assertion。
    3) 返回对象 AuthenticateOut。

> **注释**
> 一些认证器可能不在它内部而在 ASM 中支持“交易信息确认显示”。这些通常是基于交易信息确认显示的软件。当用一个已知的交易来执行 Sign 命令时，ASM 应该在它自己的 UI 上显示交易信息，当用户确认后将内容发送给认证器，这样认证器就可以将其添加到最终断言中。
> 描述交易信息确认显示类型的标志可查阅[UAFRegistry]。

认证器的元数据必须能够正确指示实现的交易信息确认显示的类型。“交易信息确认显示”标志将被设置为 TRANSACTION_CONFIRMATION_DISPLAY_ANY 或 TRANSACTION_CONFIRMATION_DISPLAY_PRIVILEGED_SOFTWARE 。

3.8 注销请求

从认证器中删除注册的 UAF 记录。

对于注销请求，下列 **ASMRequest** 成员必须具有如下值。其它的 **ASMRequest** 成员应该被省略。

- ASMRequest.requestType 必须设置为 Deregister。
- ASMRequest.asmVersion 必须设置为所需的版本。
- ASMRequest.authenticatorIndex 必须设置为目标认证器索引。
- ASMRequest.args 必须设置为 DeregisterIn 类型的对象。

对于注销响应，下列 **ASMResponse** 成员必须具有如下值。其它的 **ASMResponse** 成员应该被省略。

- ASMResponse.statusCode 必须具有下列值中的一个：
  - UAF_ASM_STATUS_OK
  - UAF_ASM_STATUS_ERROR
  - UAF_ASM_STATUS_ACCESS_DENIED

3.8.1 DeregisterIn 对象

WebIDL

```
dictionary DeregisterIn {
    required DOMString          appID;
    required DOMString          keyID;
};
```

3.8.1.1 DeregisterIn 结构成员

appID类型为 required DOMString

FIDO 服务器应用程序标识。

keyID类型为 required DOMString

base64 编码 [RFC4648]的注销认证器的密钥标识符。

3.8.2 注销请求过程的详细描述

查阅[UAFAuthnrCommands] 文档可获得更多的关于 TAGs 和本章节涉及的结构的信息。

1、使用 authenticatorIndex 来定位认证器。

2、生成 KHAccessToke（更多细节见 KHAccessToken）。

3、如果认证器为绑定认证器，那么：

- 在 ASM 数据库中查阅认证器的相关数据，删除与 DeregisterIn.appID 和 DeregisterIn.appID 相关的记录。

4、创建 TAG_UAFV1_DEREGISTER_CMD 结构，复制 KHAccessToken, DeregisterIn.keyID，并传递给认证器。

5、调用命令，接收响应。

3.9 GetRegistrations 请求

返回为当前的 FIDO UAF Client 建立的所有注册。

对于 GetRegistrations 请求，下列 **ASMRequest** 成员必须具有如下值。其它的 **ASMRequest** 成员应该被省略。

- ASMRequest.requestType 必须设置为 GetRegistrations。
- ASMRequest.asmVersion 必须设置为所需的版本。
- ASMRequest.authenticatorIndex 必须根据相应的 ID 设置。

对于 GetRegistrations 响应，下列 **ASMResponse** 成员必须具有如下值。其它的 **ASMResponse** 成员应该被省略。

- ASMResponse.statusCode 必须具有下列值中的一个：
  - UAF_ASM_STATUS_OK
  - UAF_ASM_STATUS_ERROR
- ASMResponse.responseData 必须是 GetRegistrationsOut 类型的对象。

3.9.1 GetRegistrationsOut 对象

WebIDL

```
dictionary GetRegistrationsOut {
    required AppRegistration[]          appRegs;
};
```

3.9.1.1 GetRegistrationsOut 结构成员

appRegs类型为 required AppRegistration 数组

与 appID 相关的注册列表（见 AppRegistration），可能是一个空表。

3.9.2 AppRegistration 对象

WebIDL

```
dictionary AppRegistration {
    required DOMString          appID;
    required DOMString          keyIDs;
};
```

3.9.2.1 AppRegistration 结构成员

appID类型为 required DOMString

FIDO 服务器应用程序标识。

keyIDs类型为 required DOMString 数组

和 appID 相关的密钥标识列表。

3.9.3GetRegistrations 请求过程的详细描述

1. 使用 authenticatorIndex 定位认证器。
2. 如果认证器为绑定认证器，那么：
   - 在 ASM 数据库中查询与 CallerID 和 AppID 相关的注册记录并且建立一个 AppRegistration 对象的列表。

   > **注释**
   > 一些 ASM 可能并没有将这些信息存储在自身数据库中，而是将其存储在认证器的安全存储空间。在这种情况下，ASM 必须发送一个适当的命令来获取相应的数据。
3. 创建对象 GetRegistrationsOut 并返回。

3.10 OpenSettings 请求

显示认证器具体设置接口。如果认证器有内置的用户界面，ASM 必须调用 TAG_UAFV1_OPEN_SETTINGS_CMD 来显示它。

对于 OpenSettings 请求，下列 **ASMRequest** 成员必须具有如下值。其它的 **ASMRequest** 成员应该被省略。

- ASMRequest.requestType 必须设置为 OpenSettings。
- ASMRequest.asmVersion 必须设置为所需的版本。
- ASMRequest.authenticatorIndex 必须设置为目标认证器索引。

对于 OpenSettings 响应，下列 **ASMRequest** 成员必须具有如下值。其它的 **ASMRequest** 成员应该被省略。

- ASMResponse.statusCode 必须具有下列值中的一个：
    - UAF_ASM_STATUS_OK

4、使用 ASM API

本节不是规范性的。

在一个典型的实现过程中，FIDO UAF 客户端会在初始化和回去认证器信息时调用 GetInfo。一旦得到信息，通常将被用于在 FIDO UAF 消息处理过程中找到与其匹配的 FIDO UAF 策略。一旦策略匹配，FIDO UAF 客户端将向 ASM 发送合适的请求（注册、鉴别、注销等）。

FIDO UAF Client 可能会利用从 GetInfo 响应中获取的信息来向用户显示认证器的相关信息。

5、在不同平台上使用 ASM API

本节是规范的。

5.1 Android ASM Intent API

在安卓系统中，FIDO UAF ASMs 可能实现为一个单独的 APK 打包的应用程序。FIDO UAF 客户端通过 Android Intents 调用 ASM 操作。FIDO UAF 客户端和 Android 系统中 ASM 之间的所有信息交互都通过下面的 intert 标识发生：org.fidoalliance.intent.FIDO_OPERATION。

为了获取此文档中描述的消息，intent 必须有 type 属性设置为 application/fido.uaf_asm+json。

ASM 必须在其清单文件中注册该 intent 并为其实现一个处理程序。

在调用 intent 之前，FIDO UAF 客户端必须附带一个 message，其中含有 **ASMRequest** 的 String 表示。

FIDO UAF 客户端必须通过调用 startActivityForResult()来调用 ASMs。

FIDO UAF 客户端为了处理该 intent 应该假定 ASM 可以向用户展示一个界面。例如：提示用户完成验证过程。但是，ASM 在处理 GetInfo 请求时不应该显示任何用户界面。

该进程完成后，ASM 会把响应 intent 作为一个参数返回给 onActivityResult()。响应 intent 附带一个 message，其中包含 **ASMRequest** 的 String 表示。

5.1.1 发现 ASMs

FIDO UAF 客户端可以通过使用 PackageManager.queryIntentActivities(Intent intent, int flags)和上述介绍过的 FIDO Intent 来发现系统上可用 ASM，从而判断是否有任何活动可用。

典型的 FIDO UAF 客户端会运用此功能来枚举所有的 ASM 应用程序，并且也会对发现的 ASM 调用 GetInfo 操作。

5.2 Windows ASM API

在 Windows 中，AMS 以动态链接库（DLL）的形式来实现。下面是一个定义了 Windows ASM API 的 asmplugin.h 报头文件。

例 1

```
/*! @file asm.h
*/

#ifndef __ASMH_
#define __ASMH_
#ifdef _WIN32
#define ASM_API __declspec(dllexport)
#endif
```

```
#ifdef _WIN32
#pragma warning ( disable : 4251 )
#endif

#define ASM_FUNC extern "C" ASM_API
#define ASM_NULL 0

/*! \brief Error codes returned by ASM Plugin API.
*    Authenticator specific error codes are returned in JSON form.
*    See JSON schemas for more details.
*/

enum asmResult_t
{
   Success = 0, /**< Success */
   Failure /**< Generic failure */
};

/*! \brief Generic structure containing JSON string in UTF-8
*    format.
*    This structure is used throughout functions to pass and receives
*    JSON data.
*/

struct asmJSONData_t
{
   int length; /**< JSON data length */
   char *pData; /*< JSON data */
};

/*! \brief Enumeration event types for authenticators.
These events will be fired when an authenticator becomes
   available (plugged) or unavailable (unplugged).
*/

enum asmEnumerationType_t
{
   Plugged = 0, /**< Indicates that authenticator Plugged to system */
   Unplugged /**< Indicates that authenticator Unplugged from system */
};

namespace ASM
{
```

```
    /*! \brief Callback listener.
    FIDO UAF Client must pass an object implementating this interface to
    Authenticator::Process function. This interface is used to provide
    ASM JSON based response data.*/
    class ICallback
    {
       public
          virtual ~ICallback() {}
          /**
          This function is called when ASM's response is ready.
          *
          @param response JSON based event data
          @param exchangeData must be provided by ASM if it needs some
          data back right after calling the callback function.
          The lifecycle of this parameter must be managed by ASM. ASM must
          allocate enough memory for getting the data back.
          */

          virtual void Callback(const asmJSONData_t &response,
          asmJSONData_t &exchangeData) = 0;
    };

    /*! \brief Authenticator Enumerator.
    FIDO UAF Client must provide an object implementing this
    interface. It will be invoked when a new authenticator is plugged or
    when an authenticator has been unplugged. */

    class IEnumerator
    {
       public
          virtual ~IEnumerator() {}
          /**
             This function is called when an authenticator is plugged or
             unplugged.
          * @param eventType event type (plugged/unplugged)
             @param AuthenticatorInfo JSON based GetInfoResponse object
          */

          virtual void Notify(const asmEnumerationType_t eventType, const
          asmJSONData_t &AuthenticatorInfo) = 0;
    };
}

/**
```

```
    Initializes ASM plugin. This is the first function to be
                called.
    *
    @param pEnumerationListener caller provided Enumerator
    */

    ASM_FUNC asmResult_t asmInit(ASM::IEnumerator
            *pEnumerationListener);
    /**
    Process given JSON request and returns JSON response.
    *
    If the caller wants to execute a function defined in ASM JSON
                schema then this is the function that must be called.
    *
    @param pInData input JSON data
    @param pListener event listener for receiving events from ASM
    */
    ASM_FUNC asmResult_t asmProcess(const asmJSONData_t *pInData,
            ASM::ICallback *pListener);
    /**
    Unitializes ASM plugin.
    *
    */
    ASM_FUNC asmResult_t asmUninit();
    #endif // __ASMPLUGINH_
```

基于 Windows 的 FIDO UAF 客户端必须按照下列注册表路径来寻找 ASM DLLs：

HKCU\Software\FIDO\UAF\ASM

HKLM\Software\FIDO\UAF\ASM

FIDO UAF 客户端遍历这个路径下的所有关键字并寻找“path”字段：

[HK**\Software\FIDO\UAF\ASM\<exampleASMName>]

"path"="<ABSOLUTE_PATH_TO_ASM>.dll"

path 必须指向 ASM DLL 的绝对位置。

6、安全性和私密性

本节是规范的。

ASM 的开发者必须保护好他们正在使用的 FIDO UAF 数据。ASM 必须满足下列

安全性：

- ASM 必须实现可以把由两个不同的 FIDO UAF 客户端注册的 UAF 证书相互隔离的机制。一个 FIDO UAF 客户端不得访问通过另一个不同的 FIDO UAF 客户端注册的 FIDO UAF 凭证。这防止了恶意软件运用与合法 FIDO 客户端相关的凭证。

  注释

  为了满足[FIDOSecRef]中的假定，ASMs 必须通过平台提供的隔离机制来妥善保护自身的敏感数据，以此来对抗恶意软件。有 root 系统访问权限的恶意软件或针对设备的直接物理攻击不在此要求的范围内。

  注释

  以下是满足此要求的例子：

  - 如果 ASM 与 FIDO UAF 客户端是绑定的，那么隔离机制已经内建。
  - 如果 ASM 和 FIDO UAF 客户端由相同的供应商实现，供应商可以通过提供专用机制来绑定 ASM 与 FIDO UAF 客户端。
  - 在有些平台上，ASMs 和 FIDO UAF 客户端可能被赋予常规应用不具备的特殊权限或许可。这类平台上的 ASM 可能不会支持隔离每个 FIDO UAF 客户端的 UAF 证书，因为所有的 FIDO UAF 客户端都被认为是同样值得信赖的。

- 为绑定认证器专门设计的 ASM 必须确保：在一个 ASM 注册的 FIDO UAF 凭证不能被其它的 ASM 访问。这是为了防止某个应用伪装成 ASM 来利用合法的 UAF 凭证。
  - 使用 KHAccessToken提供这样的机制。
- 为了保护 UAF 相关的存储数据，ASM 必须实现平台提供的最佳的安全实践。
- ASMs 不得允许在它的本地存储任何敏感的 FIDO UAF 数据，下列除外：
  - CallerID, ASMToken, PersonaID, KeyID, KeyHandle, AppID

> 注释
> 例如，ASM 不能以任何形式在本地存储 FIDO 服务器提供的用户名，除非仅能够被认证器解密。

- ASMs 应该保证应用程序不能使用静默认证器进行追踪。支持静默认证器的 ASM 在每次注册时必须展示用户界面，用来解释什么是静默认证器并询问用户是否进行注册。并且建议 ASM 设计支持漫游静默认证器：
  - 在系统中具有特别许可或特权，或
  - 与保证其他的应用程序不能绕过此 ASM 直接与认证器取得联系的认证器有一个内置绑定。

6.1 KHAccessToken

KHAccessToken 是保护认证器的 FIDO UAF 证书不被非法使用的访问控制机制。它是 ASM 通过混合不同来源的信息创建的。通常情况下一个 KHAccessToken 包含了下列四个数据项 AppID, PersonaID, ASMToken 和 CallerID。

AppID 由 FIDO Server 提供，它包含在所有的 FIDO UAF 信息中。

PersonaID 由 ASM 从操作环境中获得。通常情况下不同的 **PersonaID** 分配到每个操作系统的用户帐号。

ASMToken 是随机生成的秘密信息，它受到 ASM 的维护和保护。

> 注释
> 一般实现过程是：ASM 在第一次启动时会随机生成一个 ASMToken，并且将一直维护这个秘密直到 ASM 被删除。

CallerID 是平台分配给请求的 FIDO UAF 客户端的 ID（如：ios 的“bundle ID”）。在不同的平台上 CallerID 可以以不同的方式获得。

> 注释
> 例如在安卓平台上，ASM 可以使用请求者的 apk-signing-cert 哈希值。

ASM 使用 KHAccessToken 来建立一个 ASM 和密钥句柄之间的连接，密钥句柄是由认证器为了代表此 ASM 而创建。

ASM 将 KHAccessToken 和所有需要密钥句柄的命令语句一起提供给认证器。

> 注释

下列示例说明了 ASM 怎样构建和使用 KHAccessToken。

- 在 Register 请求中
  - 附加上 AppID
    - KHAccessToken = AppID
  - 如果是绑定认证器，附加上 ASMToken, PersonaID 和 CallerID。
    - KHAccessToken |= ASMToken | PersonaID | CallerID
  - 对 KHAccessToken 进行哈希。
    - 使用认证器的哈希函数对 KHAccessToken 进行哈希。之所以使用认证器的特定哈希函数是为了确认和 AMSs 之间的互操作性。如果不要求互操作性，ASM 可以使用它想用的任何安全的哈希函数。
    - KHAccessToken=hash(KHAccessToken)
  - 提供 KHAccessToken 给认证器
  - 认证器把 KHAccessToken 放在 RawKeyHandle 中（更多细节查阅 [UAFAuthnrCommands]）。
- 在其它要求 KHAccessToken 作为输入参数的命令语句中：
  - ASM 用 Register 请求过程中的相同方法计算 KHAccessToken，并把它和其它参数一起给认证器。
  - 认证器打开密钥句柄，仅在 RawKeyHandle.KHAccessToken 与提供的 KHAccessToken 相同的情况下，将密钥句柄和控制命令一起运行。

绑定认证器必须支持一个绑定密钥句柄和 ASMs 的机制。这个绑定机制的安全特性必须至少和前述的保护 KHAccessToken 的机制相同。所以，建议由 AppID, ASMToken, PersonaID 和 CallerID 安全的获得 KHAccessToken。

注释

对漫游认证器来说，建议 KHAccessToken 仅包含 AppID，否则用户不能在不同的机器上（PersonaID, ASMToken 和 CallerID 是特定平台的）使用。允许认证器供应商为了满足特定的用途这样做。

对所有类型的认证器来说，在 KHAccessToken 中包含 PersonaID 是可选的。然而

为多用户系统设计的认证器往往应该包含它。

6.2 ASM APIs 的访问控制

下表总结了调用每个 API 的访问控制要求。

ASMs 必须实现下述定义的访问控制要求。ASM 供应商可能会实现额外的安全机制。

下表中使用的术语：

- NoAuth--无访问控制
- CallerID--分配给 FIDO UAF 客户端平台的 ID 是经过验证的
- UserVerify--用户必须被明确验证
- KeyIDList--调用方必须知晓

命令	第一因子绑定认证器	第二因子绑定认证器	第一因子漫游认证器	第二因子漫游认证器
GetInfo	NoAuth	NoAuth	NoAuth	NoAuth
OpenSettings	NoAuth	NoAuth	NoAuth	NoAuth
Register	UserVerify	UserVerify	UserVerify	UserVerify
Authenticate	UserVerify AppID CallerID PersonaID	UserVerify AppID KeyIDList CallerID PersonaID	UserVerify AppID	UserVerify AppiD KeyIDList
GetRegistrations*	CallerID PersonaID	CallerID PersonaID	X	X
Deregister	AppID KeyID PersonaID CallerID	AppID KeyID PersonaID CallerID	AppID KeyID	AppID KeyID

A.参考文献

A.1 参考规范

[ECMA-262]

ECMAScript Language Specification, Edition 5.1. June 2011.

URL: http://www.ecma-international.org/publications/standards/Ecma-262.htm

[FIDOGlossary]

R. Lindemann, D. Baghdasaryan, B. Hill, J. Hodges, *FIDO Technical Glossary*. FIDO Alliance Proposed Standard. URLs:

HTML: fido-glossary-v1.0-ps-20141208.html

PDF: fido-glossary-v1.0-ps-20141208.pdf

[RFC2119]

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

[RFC4648]

S. Josefsson, The Base16, Base32, and Base64 Data Encodings (RFC 4648), IETF, October 2006, URL: http://www.ietf.org/rfc/rfc4648.txt

[UAFAuthnrCommands]

D. Baghdasaryan, J. Kemp, R. Lindemann, R. Sasson, B. Hill, *FIDO UAF Authenticator Commands v1.0*. FIDO Alliance Proposed Standard. URLs:

HTML: fido-uaf-authnr-cmds-v1.0-ps-20141208.html

PDF: fido-uaf-authnr-cmds-v1.0-ps-20141208.pdf

[UAFAuthnrMetadata]

B. Hill, D. Baghdasaryan, J. Kemp, *FIDO UAF Authenticator Metadata Statements v1.0*. FIDO Alliance Proposed Standard. URLs:

HTML: fido-uaf-authnr-metadata-v1.0-ps-20141208.html

PDF: fido-uaf-authnr-metadata-v1.0-ps-20141208.pdf

[UAFProtocol]

R. Lindemann, D. Baghdasaryan, E. Tiffany, D. Balfanz, B. Hill, J. Hodges, *FIDO UAF Protocol Specification v1.0*. FIDO Alliance Proposed Standard. URLs:

HTML: fido-uaf-protocol-v1.0-ps-20141208.html

PDF: fido-uaf-protocol-v1.0-ps-20141208.pdf

[UAFRegistry]

R. Lindemann, D. Baghdasaryan, B. Hill, *FIDO UAF Registry of Predefined Values*. FIDO Alliance Proposed Standard. URLs:

HTML: fido-uaf-reg-v1.0-ps-20141208.html

PDF: fido-uaf-reg-v1.0-ps-20141208.pdf

[WebIDL-ED]

Cameron McCormack, Web IDL, W3C. Editor's Draft 13 November 2014. URL: http://heycam.github.io/webidl/

A.2 参考资料

[ECMA-404]

. The JSON Data Interchange Format. 1 October 2013. Standard. URL: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[FIDOSecRef]

R. Lindemann, D. Baghdasaryan, B. Hill, *FIDO Security Reference*. FIDO Alliance Proposed Standard. URLs:

HTML: fido-security-ref-v1.0-ps-20141208.html

PDF: fido-security-ref-v1.0-ps-20141208.pdf

[RFC2397]

L. Masinter. The "data" URL scheme. August 1998. Proposed Standard. URL: https://tools.ietf.org/html/rfc2397

[UAFAppAPIAndTransport]

B. Hill, D. Baghdasaryan, B. Blanke, *FIDO UAF Application API and Transport Binding Specification*. FIDO Alliance Proposed Standard. URLs:

HTML: fido-uaf-client-api-transport-v1.0-ps-20141208.html

PDF: fido-uaf-client-api-transport-v1.0-ps-20141208.pdf

[WebIDL]

Cameron McCormack. Web IDL. 19 April 2012. W3C Candidate Recommendation. URL: http://www.w3.org/TR/WebIDL/