FIDO UAF 认证器控制命令 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-authnr-cmds-v1.0-ps-20141208.html

之前版本:

https://fidoalliance.org/specs/fido-uaf-authnr-cmds-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.

罗尼·萨森（Roni Sasson）, Discretix, Inc.

布拉德•希尔 (Brad Hill)， 贝宝（PayPal, Inc.）

摘要

UAF 认证器可能会采用不同的形式。实现的范围可能从防篡改硬件上的一个安全应用到客户端设备的纯软件解决方案

本文档定义 UAF 认证器各方面的规范性，并为认证器实施者提供了安全性和实施的指导。

文档状态

本章节描述了本文在发布之时的状态。本文档有可能会被其他文档所取代。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 联盟规范索引中找到，网址为：*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*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

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。

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。

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. 综述

本节是非规范性的。

本文档阐述了 UAF 认证器为了满足 UAF 协议应该实现的底层功能。它需要达到以下目标：

- 定义 UAF 认证器实现方面的规范。
- 定义各种不同认证器实现 UAF 功能的一组命令。
- 定义 FIDO 服务器将解析的特定 UAFV1TLV 断言方案结构。

> **注释**
>
> UAF 协议支持多种断言方案。本文档中定义的命令和结构假定认证器支持 UAFV1TLV 断言方案。认证器实现不同的断言方案不必须遵循本文档的特定要求。

UAF 协议的总体架构及各种操作在[UAFProtocol]中均有描述。下面的结构简图说明了本文档主要关注的交互和参与者。

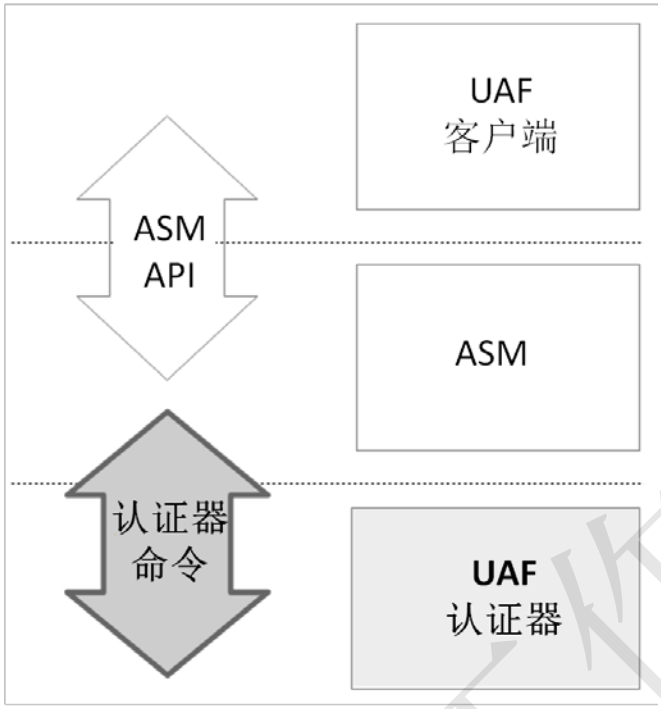

图 1 UAF 认证器命令

3. 附加说明

本节是规范性的。

除非特别说明，本文档中描述的所有数据都必须是以**小字节序**格式编码的。

所有的 TLV 结构都使用“递归下降”解析方法，标签顺序并不重要。在某些情况下，在一个结构中可能允许同一标签多次出现，在这种情况下，所有的值必须保留。

除非特别说明，否则所有 TLV 结构的字段都是*强制性的*。

4. UAF 认证器

本节是非规范性的。

UAF 认证器是满足[UAFProtocol]中描述的 UAF 协议要求的认证单元。UAF 认证器提供的主要功能有：

1. [强制的] 在认证器内部实现用户校验机制。校验机制是不同的，从生物识别校验到简单验证物理存在，或者无用户校验（又叫“*静默认证器*”）

2. [强制的] 实现在[UAFProtocol]中定义的加密操作。
3. [强制的] 生成 FIDO 服务器可以解析的数据结构。
4. [强制的] 如果认证器内置鉴证功能，需要向 FIDO 服务器进行鉴证。
5. [可选的] 使用交易确认显示功能将交易内容呈现给用户。

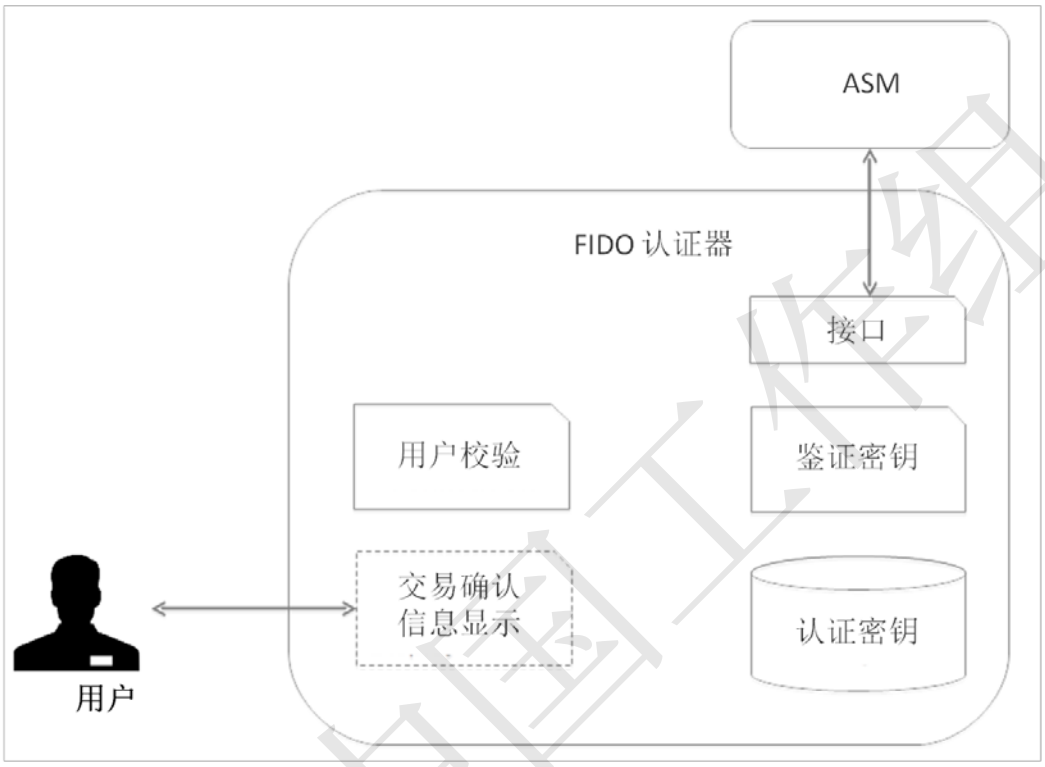

图 2 FIDO 认证器逻辑子组件

UAF 认证器举例：

- 手机中内置的指纹传感器。
- *安全元件*内的 PIN 码认证器。
- 一台作为其他设备认证器的手机。
- 有内置用户在场验证的 USB 令牌。
- 设备中内置的声音或人脸识别验证技术。

4.1 认证器的类型

本文档中定义了四种类型的认证器。这些定义不是规范性的（特别声明的除外），仅仅为了简化一些定义的描述。

> 注释

以下是仅考虑的这四种认证器的理论依据：

- 绑定认证器需要嵌入用户的计算设备，这样就可以利用设备的存储满足其需求。从节约成本考虑，利用设备的存储空间比内置存储更有意义。可信执行环境 (TEE)，安全元件和可信平台模块(TPM)是根据此规则定义的。
- 第一因子漫游认证器必须具有内置存储空间来存储密钥句柄。
- 为了避免内置存储，第二因子漫游认证器可以把它的密钥句柄存储在与其相关联的服务器上。
- 定义这样的约束使得定义主要用例更加简单明了。

然而，供应商不受这些条件限制。例如，有存储密钥句柄的内置存储的绑定认证器也是可以的。在设计方案满足本文档描述的标准要求的前提下，供应商可以自主的设计和实现认证器。

- 第一因子绑定认证器
    - 此类认证器有内置匹配器。匹配器能够校验已经注册的用户。如果不止一个用户注册了，匹配器可以识别不同用户。
    - 此类认证器和其隶属的设备间存在逻辑绑定关系（这种关系通过 KeyHandleAccessToken 的概念表述）。此类认证器不能和多个设备绑定在一起。
    - 此类认证器不会将密钥句柄存储在它们的内置存储中。他们通常会向 ASM 返回密钥句柄然后将其存储在本地数据库中。
    - 此类认证器也可以作为第二因子。
    - 例如：
        - 笔记本、手机和平板电脑中内建的指纹传感器。
        - 嵌入移动设备的安全元件。
        - 集成在设备上的声音识别方案。
- 第二因子绑定认证器
    - 此类认证器与第一因子绑定认证器很相似，除了它在多因子认证中可以仅作为第二因子。
    - 例如：

        - 具有内置触摸装置用来验证用户在场的 USB 适配器（dongle）。
        - 运行在手机可信执行环境中的“可信应用（Trustlet）”程序，它能利用安全键盘来验证用户在场。
- 第一因子漫游认证器
    - 此类认证器不与任何设备绑定。用户可以在任意数量的的设备上使用它。
    - 假定此类认证器有内置匹配器。匹配器能够校验已经注册的用户。如果不止一个用户注册了，匹配器可以识别不同用户。
    - 假定此类认证器设计为将密钥句柄存储在其内置的安全存储中并且不对外暴露。
    - 此类认证器也可以作为第二因子。
    - 例如：
        - 内置指纹传感器的蓝牙令牌。
        - 支持 PIN 码保护的 USB 令牌。
        - 因为用户习惯，将第一因子绑定认证器作为多设备的第一因子漫游认证器使用。
- 第二因子漫游认证器
    - 此类认证器不与任何设备绑定。用户可以在任意数量的的设备上使用它。
    - 此类认证器可能有内置匹配器。匹配器能够校验已经注册的用户。如果不止一个用户注册了，匹配器可以识别不同用户。
    - 假定此类认证器设定不会将密钥句柄保存在其内部存储中。作为替代他们会将密钥句柄发送给 FIDO 服务器，然后在鉴别过程中再接收回密钥句柄。
    - 此类认证器只能作为第二因子使用。
    - 例如：
        - 具有内置触摸装置用来验证用户在场的 USB 适配器。
        - 运行在手机可信执行环境中的“可信”应用程序，它能利用安

全键盘来验证用户在场。

纵观本文档，在应用这些类型的认证器的过程中会需要特殊的条件。

> **规范**
>
> 在某些部署中，ASM 和绑定认证器结合可以作为漫游认证器（例如，移动设备上的 ASM 和内嵌认证器可以作为另一设备的漫游认证器使用）。在这种情况下，认证器必须要符合绑定认证器在绑定的系统的边界；还得满足连接的外部系统自身对漫游认证器的要求。

> **注释**
>
> 如上所述，绑定认证器不会存储密钥句柄，而漫游认证器会存储。上述例子中，ASM 会存储绑定认证器的密钥句柄，因此满足此假定。

5. 标签

本节是规范性的。

在本文档中，UAF 认证器使用“标签-长度-值”(TLV)格式与外界联系。所有的请求和响应数据必须用 TLV 编码。

命令和预定义 TLV 标签可以通过附加其他的 TLV 标签（自定义或预定义）来扩展。

预定义 TLV 标签可以参考[UAFRegistry]。

TLV 格式的数据具有下述简单结构：

2 bytes	2 bytes	Length bytes
标签（Tag）	长度（Length in bytes）	数据（Data）

所有的长度以字节（bytes）计算，例如 UINT32[4]的长度为 16.

尽管分配了 2 字节给此标签，但是只有前 14bits（最高值 0-0x3FFF）是用来适应硬件平台的限制。

数组是隐式的。一些结构的描述会显示在允许多个数据的地方，在这种情况下，如果相同的标签不只出现一次，所有的值都是有意义的并且应该作为一个

数组看待。

复合标签是必须使用递归降序解析其数据的标签，为了方便解析 TLV 格式消息，复合标签必须在 TLV 第 13bit 位置位（0x1000）。

在第 14bit 位置位（0x2000）的标签表明 TLV 消息是极为重要的，接收者如果无法处理这个标签，则必须放弃处理整条消息。

由于 UAF 认证器可能仅有极为有限的处理环境，ASM 在发送命令时必须遵循结构的规范顺序。

假定 ASM 和服务器有足够的资源来按任意顺序处理解析的标签，那么认证器发出的结构可能使用任意顺序的标签。

5.1 命令标签

标签名	值	描述
TAG_UAFV1_GETINFO_CMD	0x3401	GetInfo 命令标签
TAG_UAFV1_GETINFO_CMD_RESPONSE	0x3601	GetInfo 命令响应标签
TAG_UAFV1_REGISTER_CMD	0x3402	Register 命令标签
TAG_UAFV1_REGISTER_CMD_RESPONSE	0x3602	Register 命令响应标签
TAG_UAFV1_SIGN_CMD	0x3403	Sign 命令标签
TAG_UAFV1_SIGN_CMD_RESPONSE	0x3603	Sign 命令响应标签
TAG_UAFV1_DEREGISTER_CMD	0x3404	Dergister 命令标签
TAG_UAFV1_DEREGISTER_CMD_RESPONSE	0x3604	Dergister 命令响应标签
TAG_UAFV1_OPEN_SETTINGS_CMD	0x3406	OpenSetting 命令标签
TAG_UAFV1_OPEN_SETTINGS_CMD_RESPONSE	0x3606	OpenSetting 命令响应标签

表 5.1.1: UAF 认证器命令 TLV 标签(0x3400 - 0x34FF, 0x3600-0x36FF)

5.2 仅在认证器中使用的标签

标签名	值	描述
TAG_KEYHANDLE	0x2801	代表密钥句柄。 参考[FIDOGlossary]获取更多关于密钥句柄信息。
TAG_USERNAME_AND_KEYHANDLE	0x3802	代表一组相关的用户名和密钥句柄。 这是一个复合标签包含了 TAG_USERNAME 和 TAG_KEYHANDLE，识别认证器内一个有效注册。 参考[FIDOGlossary]获取更多关于用户名的信息。
TAG_USERVERIFY_TOKEN	0x2803	代表用户校验令牌。 参考[FIDOGlossary]获取更多关于用户校验令牌信息。
TAG_APPID	0x2804	完整的应用标识符是一个 UINT8[]编码后的 UTF-8 字符串。 参考[FIDOGlossary]获取更多关于应用标识符信息。
TAG_KEYHANDLE_ACCESS_TOKEN	0x2805	代表密钥句柄访问令牌。
TAG_USERNAME	0x2806	用户名是一个 UINT8[]编码后的 UTF-8 字符串。
TAG_ATTESTATION_TYPE	0x2807	代表鉴证类型。
TAG_STATUS_CODE	0x2808	代表状态编码。

TAG_AUTHENTICATOR_METADATA	0x2809	代表更详细的一组认证器数据。
TAG_ASSERTION_SCHEME	0x280A	如[UAFRegistry]中定义的是一个 UINT8[]编码后的 UTF-8 编码断言方案，。（“UAFV1TLV”）
TAG_TC_DISPLAY_PNG_CHARACTERISTICS	0x280B	如果认证器不是通过上层软件来实现 PNG 图的交易确认显示，此标签表示这种显示。参考[UAFAuthnrMetadata]获取更多信息。
TAG_TC_DISPLAY_CONTENT_TYPE	0x280C	[UAFAuthnrMetadata]中定义的 UINT8[]编码的 UTF-8 编码后的交易确认显示类型。（“image/png”）
TAG_AUTHENTICATOR_INDEX	0x280D	认证器序号。
TAG_API_VERSION	0x280E	API 版本。
TAG_AUTHENTICATOR_ASSERTION	0x280F	该 TLV 标签的内容为认证器生成的断言。由于认证器可以生成不同格式的断言，所以不同认证器生成内容格式也可能不同。
TAG_TRANSACTION_CONTENT	0x2810	代表传送到认证器的交易内容。
TAG_AUTHENTICATOR_INFO	0x3811	包含了认证器属性的详细信息。
TAG_SUPPORTED_EXTENSION_ID	0x2812	代表认证器支持的扩展标识。

表 5.2.1: 非命令标签 (0x2800 - 0x28FF, 0x3800 - 0x38FF)

5.3 UAF 协议中使用的标签

标签名	值	描述
TAG_UAFV1_REG_ASSERTION	0x3E01	认证器对注册命令的响应。
TAG_UAFV1_AUTH_ASSERTION	0x3E02	认证器对签名命令的响应。
TAG_UAFV1_KRD	0x3E03	密钥注册数据。
TAG_UAFV1_SIGNED_DATA	0x3E04	认证器用用户私钥密钥签名的数据。
TAG_ATTESTATION_CERT	0x2E05	每个条目包含一个单独 X.509 DER 编码[ITU-X690-2008]的证书。允许出现多个并形成鉴证证书链。出现多个时必须是有序的，鉴证证书必须最先出现，后续的每一个（如果出现）必须由前面一个证书签发。
TAG_SIGNATURE	0x2E06	加密签名。
TAG_ATTESTATION_BASIC_FULL	0x3E07	[UAFProtocol]中定义的完整的基本鉴证。
TAG_ATTESTATION_BASIC_SURROGATE	0x3E08	[UAFProtocol]中定义的代理的基本鉴证。
TAG_KEYID	0x2E09	代表密钥标识符。
TAG_FINAL_CHALLENGE	0x2E0A	代表最终挑战。 参考[UAFProtocol]获取更多关于最后挑战值的信息。
TAG_AAID	0x2E0B	代表认证器标识符。 参考[UAFProtocol]获取更多关于认证器标识符的信息。
TAG_PUB_KEY	0x2E0C	代表公钥。
TAG_COUNTERS	0x2E0	代表认证器计数器。

	D	
TAG_ASSERTION_INFO	0x2E0E	代表消息处理过程中必要的断言信息。
TAG_AUTHENTICATOR_NONCE	0x2E0F	代表认证器生成的随机数值。
TAG_TRANSACTION_CONTENT_HASH	0x2E10	代表交易内容的哈希值。
TAG_EXTENSION	0x3E11，0x3E12	代表此内容是扩展的复合标签。 如果此标签是 0x3E11-表明这是一个临界的扩展，如果接受者不能理解此标签的含义，它必须中止整个消息处理过程。 此标签含有两个内嵌标签-TAG_EXTENSION_ID 和 TAG_EXTENSION_DATA。更多的关于 UAF 扩展的信息可参考 [UAFProtocol]。 **注释** 此标签可以附在任何命令和响应上。 使用标签 0x3E11（与 0x3E12 相反）和 [UAFProtocol] 中描述的标签 fail_if_unknown 具有相同的意思。
TAG_EXTENSION_ID	0x2E13	代表扩展标识符。 此标签的内容是 UINT8[]编码后的 UTF-8 字符串。
TAG_EXTENSION_DATA	0x2E14	代表扩展数据。 此标签的内容是一个 UINT8[]字节数组。

表 5.3.1: UAF 协议中采用的标签 (0x2E00 - 0x2EFF, 0x3E00 - 0x3EFF). 在 [UAFRegistry] 中定义。

5.4 状态编码

标签名	值	描述
UAF_CMD_STATUS_OK	0x00	成功。
UAF_CMD_STATUS_ERR_UNKNOWN	0x01	未知错误。
UAF_CMD_STATUS_ACCESS_DENIED	0x02	拒绝访问此操作。
UAF_CMD_STATUS_USER_NOT_ENROLLED	0x03	用户没有在认证器注册。
UAF_CMD_STATUS_CANNOT_RENDER_TRANSACTION_CONTENT	0x04	不能获取交易内容。
UAF_CMD_STATUS_USER_CANCELLED	0x05	用户已取消此操作。
UAF_CMD_STATUS_CMD_NOT_SUPPORTED	0x06	不支持的命令。
UAF_CMD_STATUS_ATTESTATION_NOT_SUPPORTED	0x07	不支持的断言。

表 5.4.1: UAF 认证器状态编码 (0x00 - 0xFF)

6. 结构

本节是规范性的。

6.1 RawKeyHandle

RawKeyHandle 是由认证器生成和解析的结构。认证器可能采用不同的方式来定义 RawKeyHandle，内部结构仅与特定的认证器实现相关。

标准的**第一因子绑定认证器**的 RawKeyHandle 具有下述结构：

依据哈希算法(例如	依据密钥类型 (例	用户名长度	最大 128 bytes

32 bytes)	如 32 bytes)	(1 byte)	
KHAccessToken	UAuth.priv	Size	Username

表 6.1: RawKeyHandle 结构

第一因子认证器必须在 RawKeyHandle 内存储用户名，而第二因子认证器不得存储。支持用户名的能力是第一因子认证器和第二因子认证器的关键区别。

由于 RawKeyHandle 含有用户私钥，因此在离开认证器的边界之前，RawKeyHandle 必须是加密包裹过的。

6.1.1 FIDO 服务器解析的结构

本部分定义的是由 UAF 认证器创建，被 FIDO 服务器解析的结构。
如果认证器使用“UAFV1TLV”断言方案，那么它们必须生成这些结构。

> 注释
> "UAFV1TLV"断言方案假定认证器对所有数据有高级控制权，包括 TAG_UAFV1_KRD 和 TAG_UAFV1_SIGNED_DATA。

嵌套结构必须被保存，不过包含符合标签在内的标签的顺序是不规范的。FIDO 服务器必须准备好处理以任意顺序出现的标签。

6.1.1.1 TAG_UAFV1_REG_ASSERTION

下面的 TLV 结构是认证器在注册命令的过程中生成，然后被完整的传送给 FIDO 服务器，FIDO 服务器将对其进行解析。此结构嵌入一个在其他数据中包含新生成的用户公钥的 TAG_UAFV1_KRD 标签。

如果认证器想在 TAG_UAFV1_KRD 结构（用鉴证密钥签名）中附加上自定义数据，那么这些数据必须在 TAG_UAFV1_KRD 内部作为一个附加标签。

如果认证器想发送没有签名的附加数据到 FIDO 服务器，那么这些数据必须在 TAG_UAFV1_REG_ASSERTION 内作为一个附加标签，并且不在 TAG_UAFV1_KRD 内。

目前，此文档仅定义了 TAG_ATTESTATION_BASIC_FULL 和 TAG_ATTESTATION_BASIC_SURROGATE。如果认证器被要求在 TAG_UAFV1_KRD 上进行“Some_Other_Attestation”，那么其必须使用为“Some_Other_Attestation”定义的 TLV 标签和内容（定义在[UAFRegistry]中）。

TLV 结构		描述
1	UINT16 Tag	TAG_UAFV1_REG_ASSERTION。
1.1	UINT16 Length	结构长度。
1.2	UINT16 Tag	TAG_UAFV1_KRD。
1.2.1	UINT16 Length	结构长度。
1.2.2	UINT16 Tag	TAG_AAID。
1.2.2.1	UINT16 Length	认证器验证标识符长度。
1.2.2.2	UINT8[] AAID	认证器验证标识符。
1.2.3	UINT16 Tag	TAG_ASSERTION_INFO。
1.2.3.1	UINT16 Length	断言信息长度。
1.2.3.2	UINT16 AuthenticatorVersion	供应商指定的认证器版本。
1.2.3.3	UINT8 AuthenticationMode	注册时必须等于 0x01，代表用户明确确认该操作。
1.2.3.4	UINT16 SignatureAlgAndEncoding	签名算法和鉴别签名的编码。 参考[UAFRegistry]获取更多关于支持算法和其值的相关信息。
1.2.3.5	UINT16 PublicKeyAlgAndEncoding	公钥算法和新生成的 PAuth.pub公钥编码。 参考[UAFRegistry]获取更多关于支持算法和其值的相关信息。
1.2.4	UINT16 Tag	TAG_FINAL_CHALLENGE。
1.2.4.1	UINT16 Length	最终挑战值长度。
1.2.4.2	UINT8[] FinalChallenge	命令中提供的最终挑战（二进制

		值）。
1.2.5	UINT16 Tag	TAG_KEYID。
1.2.5.1	UINT16 Length	密钥标识符长度。
1.2.5.2	UINT8[] KeyID	认证器生成的密钥标识符（二进制值）。
1.2.6	UINT16 Tag	TAG_COUNTERS。
1.2.6.1	UINT16 Length	计数器长度。
1.2.6.2	UINT32 SignCounter	签名计数器值。 标示此认证器过去已经进行的签名次数。
1.2.6.3	UINT32 RegCounter	注册计数器值。 标示此认证器过去已经进行的注册次数。
1.2.7	UINT16 Tag	TAG_PUB_KEY。
1.2.7.1	UINT16 Length	用户公钥长度。
1.2.7.2	UINT8[] PublicKey	认证器新生成的用户鉴别公钥（UAuth.pub）。
1.3(choice 1)	UINT16 Tag	TAG_ATTESTATION_BASIC_FULL。
1.3.1	UINT16 Length	结构长度。
1.3.2	UINT16 Tag	TAG_SIGNATURE。
1.3.2.1	UINT16 Length	签名长度。
1.3.2.2	UINT8[] Signature	使用基本鉴证私钥对 TAG_UAFV1_KRD 的内容进行签名。 完整的 TAG_UAFV1_KRD 内容，包含了标签和其长度字段，必须包含在签名运算中。
1.3.3	UINT16 Tag	TAG_ATTESTATION_CERT (可能多

		次出现)。 出现多个时必须是有序的，出现多个时必须是有序的，鉴证证书必须最先出现，后续的每一个（如果出现）必须由前面一个证书签发。最后一个必须链接到相关源数据声明[UAFAuthnrMetadata]中 attestationRootCertificate字段中的证书。
1.3.3.1	UINT16 Length	鉴证证书长度。
1.3.3.2	UINT8[] Certificate	单个 X.509 DER 编码的[ITU-X690-2008]鉴证证书或者鉴证证书链中的单个证书（参考之前的描述）。
1.3(choice 2)	UINT16 Tag	TAG_ATTESTATION_BASIC_SURROGATE。
1.3.1	UINT16 Length	结构长度。
1.3.2	UINT16 Tag	TAG_SIGNATURE。
1.3.2.1	UINT16 Length	签名长度。
1.3.2.2	UINT8[] Signature	使用新产生的用户私钥对 TAG_UAFV1_KRD 的内容进行签名。 完整的 TAG_UAFV1_KRD 内容，包含了标签和其长度字段，必须包含在签名运算中。

6.1.1.2 TAG_UAFV1_AUTH_ASSERTION

下面的 TLV 结构是认证器在签名命令的过程中生成的，然后被完整的传送给 FIDO 服务器，FIDO 服务器将对它进行解析。此结构嵌入一个 TAG_UAFV1_SIGNED_DATA 标签 。

如果认证器要在 TAG_UAFV1_SIGNED_DATA 结构（用鉴证密钥签名）中附加自定义数据，那么这些数据必须在 TAG_UAFV1_SIGNED_DATA 内作为一个附加标签。

如果认证器要发送没有签名的附加数据到 FIDO 服务器，那么这些数据必须在 TAG_UAFV1_AUTH_ASSERTION 内作为一个附加标签，并且不在 TAG_UAFV1_SIGNED_DATA 内。

TLV 结构		描述
1	UINT16 Tag	TAG_UAFV1_AUTH_ASSERTION。
1.1	UINT16 Length	结构长度。
1.2	UINT16 Tag	TAG_UAFV1_SIGNED_DATA。
1.2.1	UINT16 Length	结构长度。
1.2.2	UINT16 Tag	TAG_AAID。
1.2.2.1	UINT16 Length	认证器验证标识符长度。
1.2.2.2	UINT8[] AAID	认证器验证标识符。
1.2.3	UINT16 Tag	TAG_ASSERTION_INFO。
1.2.3.1	UINT16 Length	断言信息长度。
1.2.3.2	UINT16 AuthenticatorVersion	供应商指定的认证器版本。
1.2.3.3	UINT8 AuthenticationMode	鉴别模式，表明用户是否通过明确验证，以及交易内容是否存在。 • 0x01 表示用户通过明确验证。 • 0x02 表示交易内容已被展示在屏幕上并且用户通过认证器明确验证并确认了交易。
1.2.3.4	UINT16 SignatureAlgAndEncoding	签名算法和编码格式。 参考[UAFRegistry]获取更多关于支持算法和其值的相关信息。
1.2.4	UINT16 Tag	TAG_AUTHENTICATOR_NONCE。

1.2.4.1	UINT16 Length	认证器随机数长度，必须至少 8 字节。
1.2.4.2	UINT8[] AuthnrNonce	认证器生成的随机数（二进制值）。
1.2.5	UINT16 Tag	TAG_FINAL_CHALLENGE。
1.2.5.1	UINT16 Length	最终挑战长度。
1.2.5.2	UINT8[] FinalChallenge	命令中提供的最终挑战（二进制值）。
1.2.6	UINT16 Tag	TAG_TRANSACTION_CONTENT_HASH。
1.2.6.1	UINT16 Length	交易内容哈希值的长度。 如果鉴别模式为 0x01，即仅鉴别，没有交易确认，此长度为 0。
1.2.6.2	UINT8[] TCHash	交易内容哈希值（二进制值）。
1.2.7	UINT16 Tag	TAG_KEYID。
1.2.7.1	UINT16 Length	密钥标识符的长度。
1.2.7.2	UINT8[] KeyID	密钥标识符（二进制值）。
1.2.8	UINT16 Tag	TAG_COUNTERS。
1.2.8.1	UINT16 Length	计数器长度。
1.2.8.2	UINT32 SignCounter	签名计数器。 标示此认证器过去已经进行的签名次数。
1.3	UINT16 Tag	TAG_SIGNATURE。
1.3.1	UINT16 Length	签名长度。
1.3.2	UINT8[] Signature	使用用户私钥对 TAG_UAFV1_SINGNED_DATA 结构的内容进行签名。 完整的 TAG_UAFV1_SIGNED_DATA 内容，包括标签和其长度字段，必须包含在签名运算中。

6.1.2 用户验证令牌

本标准没有详细阐述在认证器中怎样执行用户校验。用户校验是认证器和供应商

的具体操作。

本文档提供了一个实例来说明 “Vendor_specific_UserVerify”命令（一个检验用户是否使用认证器内置技术的命令）怎样才能安全地与 UAF 注册和签名命令绑定。该绑定是通过 UserVerificationToken 概念建立的。该绑定允许“Vendor_specific_UserVerify”和“UAF Register/Sign”命令解耦合。

定义如下所示：

- ASM 调用“Vendor_specific_UserVerify”命令。认证器核实用户并返回 UserVerificationToken。
- ASM 调用 UAF.Register/Sign 命令并传送 UserVerificationToken。认证器检验 UserVerificationToken 的有效性，如果有效，认证器执行 FIDO 操作。

UserVerificationToken 概念是非规范性的。认证器可以决定用不同的方法来实现该绑定。例如，认证器供应商可能会直接在“Vendor_specific_UserVerify” 命令上附加一个 UAF 注册，并把它们都作为独立的命令处理。

如果使用 UserVerificationToken 绑定，需要满足下列标准中的一个或者提供一个相同或更好的安全性机制：

- UserVerificationToken 必须只允许进行一个 UAF 注册操作或一个 UAF 签名操作。
- UserVerificationToken 必须受时间限制，并且允许在规定的时间内做多重 UAF 操作。

6.2 控制命令

本节是非规范性的。

规范

能够与其他供应商提供的 ASM 互联互通的认证器必须实现本节定义的命令接口。此类认证器的实例如下所示：

- 绑定认证器的核心认证器功能由一个供应商开发，ASM 由另一个供应商开发。
- 漫游认证器。

规范

与定制 ASM 紧密结合的 UAF 认证器（一般是绑定认证器）可能会实现不同的命令接口。

所有的 UAF 认证器命令和响应都是语义相近的，它们都是 TLV 编码的。每个命令的前 2 字节是命令编码。接收到命令后，认证器必须解析第一个 TLV 标识并明确发出了哪个命令。

6.2.1 GetInfo 命令

6.2.1.1 命令描述

该命令返回内部认证器信息。它可能会返回 0 个或多个认证器。每个认证器有一个分配的 authenticatorIndex，在其它控制命令中作为认证器参考使用。

6.2.1.2 命令结构

TLV 结构		描述
1	UINT16 Tag	TAG_UAFV1_GETINFO_CMD。
1.1	UINT16 Length	完整命令长度，在这个命令中必须为 0。

6.2.1.3 命令响应

TLV 结构		描述
1	UINT16 Tag	TAG_UAFV1_GETINFO_CMD_RESPONSE。
1.1	UINT16 Length	响应长度。
1.2	UINT16 Tag	TAG_STATUS_CODE。
1.2.1	UINT16 Length	状态编码长度。
1.2.2	UINT16 Value	认证器返回的状态编码。
1.3	UINT16 Tag	TAG_API_VERSION。
1.3.1	UINT16 Length	API 版本长度（必须为 0x0001）。
1.3.2	UINT8 Version	认证器 API 版本（必须为 0x01）。该版本显示了

		此认证器支持的命令类型和相关的格式。
1.4	UINT16 Tag	TAG_AUTHENTICATOR_INFO （可能多次出现）。
1.4.1	UINT16 Length	认证器信息长度。
1.4.2	UINT16 Tag	TAG_AUTHENTICATOR_INDEX。
1.4.2.1	UINT16 Length	认证器索引长度（必须是 0x0001）。
1.4.2.2	UINT8 AuthenticatorIndex	认证器索引。
1.4.3	UINT16 Tag	TAG_AAID。
1.4.3.1	UINT16 Length	认证器验证标识符长度。
1.4.3.2	UINT8[] AAID	供应商指定的认证器验证标识符。
1.4.4	UINT16 Tag	TAG_AUTHENTICATOR_METADATA。
1.4.4.1	UINT16 Length	认证器元数据的长度。
1.4.4.2	UINT16 AuthenticatorType	标识认证器是绑定认证器还是漫游认证器、是第一因子的还是第二因子的。ASM 必须利用这个信息来确定怎样和认证器协同工作。 预定义值： • 0x0001——表示第二因子认证器（当不设置此标识时是第一因子认证器）。 • 0x0002——表示漫游的认证器（当不设置此标志时表示是绑定的认证器）。 • 0x0004——表示密钥句柄将存储在认证器内并且不会返回给 ASM。 • 0x0008——认证器具有注册和验证的内置用户界面。ASM 不能显示自定义用户界面。 • 0x0010——认证器具可设置的内置用户界面，支持 OpenSettings 命令。

		• 0x0020——认证器将 TAG_APPID 作为一个参数传递给命令，在命令中它是可选参数。 • 0x0040——至少一个用户在认证器中注册。不支持用户注册概念的认证器（例如，USER_VERIFY_NONE,USER_VERIFY_PRESENCE）必须一直设置此值。
1.4.4.3	UINT8 MaxKeyHandles	表明认证器在单个命令运行中可以接收和处理密钥句柄的最大数量。当 ASM 和多个密钥句柄一起调用 SIGN 命令时会使用这个信息。
1.4.4.4	UINT32 UserVerification	用户验证方式（[UAFRegistry]中定义）。
1.4.4.5	UINT16 KeyProtection	密钥保护方式（[UAFRegistry]中定义）。
1.4.4.6	UINT16 MatcherProtection	匹配器保护方式（[UAFRegistry]中定义）。
1.4.4.7	UINT16 TransactionConfirmationDisplay	交易确认类型（[UAFRegistry]中定义）。 **注释** 如果认证器不支持交易确认，将此值设为 0。
1.4.4.8	UINT16 AuthenticationAlg	鉴别算法（[UAFRegistry]中定义）。
1.4.5	UINT16 Tag	TAG_TC_DISPLAY_CONTENT_TYPE (可选) 。
1.4.5.1	UINT16 Length	交易确认显示内容内容类型长度。
1.4.5.2	UINT8[] ContentType	交易确认显示内容类型（更多关于此信息，参考[UAFAuthnrMetadata]）。
1.4.6	UINT16 Tag	TAG_TC_DISPLAY_PNG_CHARACTERISTICS（可选，允许出现多次）。

1.4.6.1	UINT16 Length	显示特征信息长度。
1.4.6.2	UINT32 Width	参考[UAFAuthnrMetadata]，获取更多信息。
1.4.6.3	UINT32 Height	参考[UAFAuthnrMetadata]，获取更多信息。
1.4.6.4	UINT8 BitDepth	参考[UAFAuthnrMetadata]，获取更多信息。
1.4.6.5	UINT8 ColorType	参考[UAFAuthnrMetadata]，获取更多信息。
1.4.6.6	UINT8 Compression	参考[UAFAuthnrMetadata]，获取更多信息。
1.4.6.7	UINT8 Filter	参考[UAFAuthnrMetadata]，获取更多信息。
1.4.6.8	UINT8 Interlace	参考[UAFAuthnrMetadata]，获取更多信息。
1.4.6.9	UINT8[] PLTE	参考[UAFAuthnrMetadata]，获取更多信息。
1.4.7	UINT16 Tag	TAG_ASSERTION_SCHEME。
1.4.7.1	UINT16 Length	断言方案长度。
1.4.7.2	UINT8[] AssertionScheme	断言方案（[UAFRegistry]中定义）。
1.4.8	UINT16 Tag	TAG_ATTESTATION_TYPE（可能多次出现）。
1.4.8.1	UINT16 Length	鉴证方式长度。
1.4.8.2	UINT16 AttestationType	鉴证方式（[UAFRegistry]中定义）。
1.4.9	UINT16 Tag	TAG_SUPPORTED_EXTENSION_ID （可选，可能多次出现）。
1.4.9.1	UINT16 Length	扩展标识符长度。
1.4.9.2	UINT8[] SupportedExtensionID	扩展标识符是经过 UINT8[]编码的的 UTF-8 字符串。

6.2.1.4 返回状态码

- UAF_CMD_STATUS_OK
- UAF_CMD_STATUS_ERR_UNKNOWN

6.2.2 注册命令

此命令生成 UAF 注册断言，该断言用来把认证器注册到 FIDO 服务器上。

6.2.2.1 命令结构

TLV 结构		描述
1	UINT16 Tag	TAG_UAFV1_REGISTER_CMD。
1.1	UINT16 Length	命令长度。
1.2	UINT16 Tag	TAG_AUTHENTICATOR_INDEX。
1.2.1	UINT16 Length	认证器索引长度（必须 0x0001）。
1.2.2	UINT8 AuthenticatorIndex	认证器索引。
1.3	UINT16 Tag	TAG_APPID （可选）。
1.3.1	UINT16 Length	应用标识符长度。
1.3.2	UINT8[] AppID	应用标识符（最大 512 字节）。
1.4	UINT16 Tag	TAG_FINAL_CHALLENGE。
1.4.1	UINT16 Length	最终挑战值长度。
1.4.2	UINT8[] FinalChallenge	ASM 提供的最终挑战值（最大 32 字节）。
1.5	UINT16 Tag	TAG_USERNAME。
1.5.1	UINT16 Length	用户名的长度。
1.5.2	UINT8[] Username	ASM 提供的用户名（最大 128 字节）。
1.6	UINT16 Tag	TAG_ATTESTATION_TYPE。
1.6.1	UINT16 Length	断言类型长度。
1.6.2	UINT16 AttestationType	使用的断言类型。
1.7	UINT16 Tag	TAG_KEYHANDLE_ACCESS_TOKEN。
1.7.1	UINT16 Length	密钥句柄访问令牌的长度。
1.7.2	UINT8[]	ASM 提供的密钥句柄访问令牌（最大 32 字节）。

	KHAccessToken	
1.8	UINT16 Tag	TAG_USERVERIFY_TOKEN（可选）。
1.8.1	UINT16 Length	用户验证令牌的长度。
1.8.2	UINT8[] VerificationToken	用户验证令牌。

6.2.2.2 命令响应

TLV 结构		描述
1	UINT16 Tag	TAG_UAFV1_REGISTER_CMD_RESPONSE。
1.1	UINT16 Length	命令长度。
1.2	UINT16 Tag	TAG_STATUS_CODE。
1.2.1	UINT16 Length	状态码长度。
1.2.2	UINT16 Value	认证器返回状态码。
1.3	UINT16 Tag	TAG_AUTHENTICATOR_ASSERTION。
1.3.1	UINT16 Length	断言长度。
1.3.2	UINT8[] Assertion	注册断言（参考 TAG_UAFV1_REG_ASSERTION 部分）。
1.4	UINT16 Tag	TAG_KEYHANDLE（可选）。
1.4.1	UINT16 Length	密钥句柄长度。
1.4.2	UINT8[] Value	密钥句柄（二进制值）。

6.2.2.3 返回状态码

- UAF_CMD_STATUS_OK
- UAF_CMD_STATUS_ACCESS_DENIED
- UAF_CMD_STATUS_USER_CANCELLED
- UAF_CMD_STATUS_ATTESTATION_NOT_SUPPORTED
- UAF_CMD_STATUS_ERR_UNKNOWN

6.2.2.4 命令描述

认证器必须按照如下步骤进行（命令结构表见后续表格）：

1. 如果认证器具有交易确认显示并且能够显示 AppID，确保提供了 Command.TAG_APPID，并把它的内容显示在显示器上。用 TAG_APPID 更新 Command.TAG_APPID：
    - 通过将 Command.KHAccessToken 和 Command.TAG_APPID 混合来更新它。加密哈希函数是此类混合方法的例子。

      > **注释**
      >
      > 这种方法将避免将 AppID 分开存储在 RawKeyHandle 中。

        - 例如：Command.KHAccessToken= hash(Command.KHAccessToken | Command.TAG_APPID)。
2. 如果用户已经在认证器注册（通过生物特征登记、PIN 码设置等方法），那么验证该用户。如果之前的命令已经进行了用户验证，那么需要确认 Command.TAG_USERVERIFY_TOKEN 是有效的令牌。
    1. 如果验证失败，则返回 UAF_CMD_STATUS_ACCESS_DENIED。
3. 如果用户没有在认证器注册，那么引导用户进行注册流程。
    1. 如果注册失败，则返回 UAF_CMD_STATUS_ACCESS_DENIED。
    2. 如果用户明确取消了此操作，则返回 UAF_CMD_STATUS_USER_CANCELLED。
4. 确保支持 Command.TAG_ATTESTATION_TYPE。如果不支持则返回 UAF_CMD_STATUS_ATTESTATION_NOT_SUPPORTED。
5. 生成新的密钥对(用户公钥/用户私钥) 。
6. 生成 RawKeyHandle：
    1. 将用户私钥加入 RawKeyHandle 中。
    2. 将 Command.KHAccessToken 加入 RawKeyHandle 中。
    3. 如果是第一因子认证器，将 Command.Username 加入 RawKeyHandle 中。
7. 用 Wrap.sym 密钥包裹 RawKeyHandle。
8. 创建 TAG_UAFV1_KRD 结构：
    1. 如果是第二因子漫游认证器，那么将密钥句柄放在 TAG_KEYID 中。否

则，生成新的随机 KeyID 并将它放在 TAG_KEYID 中。

2. 复制所有必填字段（参考 TAG_UAFV1_REG_ASSERTION）。

9. 基于提供的 Command.AttestationType 在 TAG_UAFV1_KRD 执行鉴证。
10. 创建 TAG_AUTHENTICATOR_ASSERTION：
    1. 创建 TAG_UAFV1_REG_ASSERTION：
        1. 复制所有必填字段（参考 TAG_UAFV1_REG_ASSERTION）。
        2. 如果为第一因子漫游认证器，那么在内部存储中添加 KeyID 和密钥句柄。
        3. 如果是绑定认证器，则返回 TAG_KEYHANDLE 内的密钥句柄。
    2. 将完整的 TLV 结构作为 TAG_AUTHENTICATOR_ASSERTION 的值，放在 TAG_UAFV1_REG_ASSERTION 中。
11. 返回 TAG_UAFV1_REGISTER_CMD_RESPONSE：
    1. UAF_CMD_STATUS_OK作为状态码。
    2. 添加 TAG_AUTHENTICATOR_ASSERTION。
    3. 如果密钥句柄必须存储在认证器外，添加 TAG_KEY_HANDLE。

规范

在没有校验用户的情况下（或者如果第一次使用该认证器时就对用户进行注册），认证器不得执行 Register命令。

认证器在每次调用注册命令时必须生成唯一的密钥对。

认证器应该将密钥句柄存储在它内部安全存储区中，或者将密钥句柄加密包裹后传递给 ASM。

对于静默认证器而言，密钥句柄必须不存储在 FIDO 服务器中。否则，由于用户没有从本地设备上清空密钥句柄的能力，所以能够对用户进行跟踪。

如果 KeyID 不是密钥句柄本身（例如第二因子漫游认证器中），那么它必须是唯一的、不可猜测的、最大长度 32 字节的字节数组。在 AAID 的范围内，它必须是唯一的。

注释

如果 KeyID 是随机生成的（举例来说，它不是从密钥句柄导出的），那么它应该存储在 RawKeyHandle 中，这样认证器就能在执行签名命令的

过程中识别它。

如果认证器不支持 SignCounter和 RegCounter等计数器功能，必须将 TAG_UAFV1_KRD 中这两个值设为 0。当认证器恢复出厂设置时，RegCounter和 SignCounter必须设置为 0。

6.2.3 签名命令

该命令生成 UAF 断言，该断言可以在已经注册过该认证器的 FIDO 服务器上进一步进行验证。

6.2.3.1 命令结构

TLV 结构		描述
1	UINT16 Tag	TAG_UAFV1_SIGN_CMD。
1.1	UINT16 Length	命令长度。
1.2	UINT16 Tag	TAG_AUTHENTICATOR_INDEX。
1.2.1	UINT16 Length	认证器索引长度（必须为 0x0001）。
1.2.2	UINT8 AuthenticatorIndex	认证器索引。
1.3	UINT16 Tag	TAG_APPID（可选）。
1.3.1	UINT16 Length	应用标识符长度。
1.3.2	UINT8[] AppID	应用标识符（最长为 512 字节）。
1.4	UINT16 Tag	TAG_FINAL_CHALLENGE。
1.4.1	UINT16 Length	最终挑战值长度。
1.4.2	UINT8[] FinalChallenge	（二进制）ASM 提供的最终挑战值（最长为 32 字节）。
1.5	UINT16 Tag	TAG_TRANSACTION_CONTENT （可选）。
1.5.1	UINT16 Length	交易内容长度。
1.5.2	UINT8[] TransactionContent	ASM 提供的交易内容（二进制值）。

1.6	UINT16 Tag	TAG_KEYHANDLE_ACCESS_TOKEN。
1.6.1	UINT16 Length	密钥句柄访问令牌长度。
1.6.2	UINT8[] KHAccessToken	（二进制）ASM 提供的密钥句柄访问令牌的值（最长为 32 字节）。
1.7	UINT16 Tag	TAG_USERVERIFY_TOKEN （可选）。
1.7.1	UINT16 Length	用户验证令牌长度。
1.7.2	UINT8[] VerificationToken	用户验证令牌。
1.8	UINT16 Tag	TAG_KEYHANDLE （可选，可能多次出现）。
1.8.1	UINT16 Length	密钥句柄长度。
1.8.2	UINT8[] KeyHandle	（二进制）密钥句柄。

6.2.3.2 命令响应

TLV 结构		描述
1	UINT16 Tag	TAG_UAFV1_SIGN_CMD_RESPONSE。
1.1	UINT16 Length	完整命令响应长度。
1.2	UINT16 Tag	TAG_STATUS_CODE。
1.2.1	UINT16 Length	状态码长度。
1.2.2	UINT16 Value	返回的状态码。
1.3 **(choice 1)**	UINT16 Tag	TAG_USERNAME_AND_KEYHANDLE （可选，多次出现）。 该 TLV 标签包含多个（>=1）数组{Username, Keyhandle}条目。 如果存在该标签，那么就不得存在 TAG_AUTHENTICATOR_ASSERTION 标签。

1.3.1	UINT16 Length	结构长度。
1.3.2	UINT16 Tag	TAG_USERNAME。
1.3.2.1	UINT16 Length	用户名长度。
1.3.2.2	UINT8[] Username	用户名。
1.3.3	UINT16 Tag	TAG_KEYHANDLE。
1.3.3.1	UINT16 Length	密钥句柄长度。
1.3.3.2	UINT8[] KeyHandle	（二进制）密钥句柄。
1.3 **(choice 2)**	UINT16 Tag	TAG_AUTHENTICATOR_ASSERTION（可选）。 如果存在该标识，那么就不得存在 TAG_USERNAME_AND_KEYHANDLE 标签。
1.3.1	UINT16 Length	断言长度。
1.3.2	UINT8[] Assertion	认证器生成的认证断言（参考 TAG_UAFV1_AUTH_ASSERTION）。

6.2.3.3 返回状态码

- UAF_CMD_STATUS_OK
- UAF_CMD_STATUS_ACCESS_DENIED
- UAF_CMD_STATUS_USER_NOT_ENROLLED
- UAF_CMD_STATUS_USER_CANCELLED
- UAF_CMD_STATUS_CANNOT_RENDER_TRANSACTION_CONTENT
- UAF_CMD_STATUS_ERR_UNKNOWN

6.2.3.4 命令描述

> 注释
>
> 第一因子认证器分两阶段执行此命令。
>
> 1. 只有当认证器在过滤 KHAccessToken 后发现多个密钥句柄才会执行第一阶段。在这个阶段，认证器必须返回与密钥句柄对应的用户名列表。

2. 在第二阶段，用户选择用户名后，该命令会被与一个密钥句柄调用并返回一个基于此密钥句柄的 UAF 断言。

如果提供不止一个有效的密钥句柄给第二因子认证器，第二因子认证器仅会采用第一个密钥句柄而忽略其它的。

此命令分两步实现，以此来保证每个命令调用时仅生成一个断言。

认证器必须按照如下步骤进行：

1. 如果认证器具有交易确认显示并且能够显示 AppID，确保提供了 Command.TAG_APPID，并把它的内容显示在显示器上。用 TAG_APPID 更新 Command.TAG_APPID：
    - 通过将 Command.KHAccessToken 和 Command.TAG_APPID 混合来更新它。加密哈希函数是此类混合方法的例子。
        - Command.KHAccessToken=hash(Command.KHAccessToken| Command.TAG_APPID)。
2. 如果用户已经在认证器注册（通过生物特征登记、PIN 码设置等方法），那么验证该用户。如果之前的命令已经进行了用户验证，那么需要确认 Command.TAG_USERVERIFY_TOKEN 是有效的令牌。
    1. 如果验证失败，则返回 UAF_CMD_STATUS_ACCESS_DENIED。
    2. 如果用户明确取消了此操作，则返回 UAF_CMD_STATUS_USER_CANCELLED。
3. 如果用户没有注册过，那么返回 UAF_CMD_STATUS_USER_NOT_ENROLLED。
4. 使用 Wrap.sym 来将所有的密钥句柄从 Command.TAG_KEYHANDL 中解包出来。
    1. 如果是第一因子漫游认证器：
        - 如果提供了 Command.TAG_KEYHANDL，那么列表中的条目为 KeyIDs。使用这些 KeyIDs 来定位存储在内部存储中的密钥句柄。
        - 如果没有提供 Command.TAG_KEYHANDL，则将存储在内部存储中的所有密钥句柄解包。

5. 用 Command.KHAccessToken 过滤 RawKeyHandles，(RawKeyHandle.KHAccessToken == Command.KHAccessToken)。
6. 如果剩下的 RawKeyHandles 的数量为 0，返回失败状态码 UAF_CMD_STATUS_ACCESS_DENIED 。
7. 如果剩下的 RawKeyHandles 的数量大于 1：
    1. 如果是第二因子认证器，选择第一个 RawKeyHandle 并跳转到步骤 8.
    2. 复制所有剩下的所有 RawKeyHandles 的{Command.KeyHandle, RawKeyHandle.username}到 TAG_USERNAME_AND_KEYHANDLE 标签。
        - 如果是第一因子漫游认证器，那么返回的 TAG_USERNAME_AND_KEYHANDLEs 必须由密钥句柄注册日期进行排序（最新注册的密钥句柄必须在最后出现）。
    3. 将 TAG_USERNAME_AND_KEYHANDLE 复制到 TAG_UAFV1_SIGN_CMD_RESPONSE，并返回。
8. 如果只有一个剩下的 RawKeyHandles：
    1. 创建 TAG_UAFV1_SIGNED_DATA 并将 TAG_UAFV1_SIGNED_DATA . AuthenticationMode 设为 0x01。
    2. 如果 TransactionContent非空：
        - 如果是静默认证器，返回 UAF_CMD_STATUS_ACCESS_DENIED。
        - 如果认证器不支持交易确认（在 GetInfo命令的响应中，将 TransactionConfirmationDisplay设为 0），返回 UAF_CMD_STATUS_ACCESS_DENIED。
        - 如果认证器有内置交易确认显示，那么在屏幕显示 Command.TransactionContent和 Command.TAG_APPID（可选），等待用户来确认它：
            - 如果用户取消了交易确认过程，返回 UAF_CMD_STATUS_USER_CANCELLED。
            - 如果提供的交易确认内容不能显示，返

回 UAF_CMD_STATUS_CANNOT_RENDER_TRANSACTION_CONTENT。

- 计算 TransactionContent 的哈希值：
  - TAG_UAFV1_SIGNED_DATA.TAG_TRANSACTION_CONTENT_HASH = hash(Command.TransactionContent)。
  - 将 TAG_UAFV1_SIGNED_DATA.AuthenticationMode 设为 0x02。

3. 创建 TAG_UAFV1_AUTH_ASSERTION。
   - 填充 TAG_UAFV1_SIGNED_DATA 字段的剩下部分。
     - 增加 SignCounter 并将其放入 TAG_UAFV1_SIGNED_DATA。
     - 复制所有的必填字段（参考 TAG_UAFV1_AUTH_ASSERTION）。
     - 如果 TAG_UAFV1_SIGNED_DATA.AuthenticationMode==0x01，那么设置 TAG_UAFV1_SIGNED_DATA.TAG_TRANSACTION_CONTENT_HASH.Length 为 0.
   - 用用户私钥签名 TAG_UAFV1_SIGNED_DATA。
4. 将整个 TLV 结构作为 TAG_AUTHENTICATOR_ASSERTION 的值放入 TAG_UAFV1_AUTH_ASSERTION 中。
5. 复制 TAG_AUTHENTICATOR_ASSERTION 到 TAG_UAFV1_SIGN_CMD_RESPONSE 并返回。

规范

如果没有校验用户，认证器不得处理签名命令。

如果没有校验用户，认证器不得暴露用户名。

如果没有校验密钥句柄访问令牌，绑定认证器不得处理签名命令。

明文形式的用户私钥必须不能离开认证器的安全边界。用户私钥的保护边界在元数据[UAFAuthnrMetadata]中的 Metadata.keyProtection字段说明。

如果认证器的元数据显示它支持交易确认显示，那么它必须显示交易确认内容，包括 TAG_UAFV1_SIGNED_DATA 结构内容的哈希值。

为了满足[FIDOSecRef]中的假定，静默认证器不得作为第一因子运行。

如果认证器不支持 SignCounter，那么必须在 TAG_UAFV1_SIGNED_DATA 中将其设为 0。为了满足[FIDOSecRef]中的假定，当认证器恢复出厂设置时，SignCounter 必须设置为 0。

一些认证器可能并不在认证器内部实现交易确认显示功能，而是在 ASM 中实现。这是典型的基于软件的交易确认显示。当对一个特定的交易执行签名命令时，这类认证器假定其含有内置的交易确认显示，将交易内容的哈希值包含在最终断言中但不给用户显示任何信息。此类认证器的元数据文件也必须明确显示交互确认显示的类型。交易确认标识应该为 RANSACTION_CONFIRMATION_DISPLAY_ANY 或 TRANSACTION_CONFIRMATION_DISPLAY_PRIVILEGED_SOFTWARE。参考 [UAFRegistry]获取更多描述交易确认显示类型标识的信息。

6.2.4 注销命令

此命令会从认证器删除注册的 UAF 凭证。

6.2.4.1 命令结构

TLV 结构		描述
1	UINT16 Tag	TAG_UAFV1_DEREGISTER_CMD。
1.1	UINT16 Length	完整命令长度。
1.2	UINT16 Tag	TAG_AUTHENTICATOR_INDEX。
1.2.1	UINT16 Length	认证器索引长度（必须为 0x0001）。
1.2.2	UINT8 AuthenticatorIndex	认证器索引。
1.3	UINT16 Tag	TAG_APPID（可选）。
1.3.1	UINT16 Length	应用标识符长度。

1.3.2	UINT8[] AppID	应用标识符（最长为 512 字节）。
1.4	UINT16 Tag	TAG_KEYID。
1.4.1	UINT16 Length	密钥标识符长度。
1.4.2	UINT8[] KeyID	ASM 提供的密钥标识符（二进制值）。
1.5	UINT16 Tag	TAG_KEYHANDLE_ACCESS_TOKEN。
1.5.1	UINT16 Length	密钥句柄访问令牌长度。
1.5.2	UINT8[] KHAccessToken	（二进制）ASM 提供的密钥句柄访问令牌（最长为 32 字节）。

6.2.4.2 命令响应

TLV 结构		描述
1	UINT16 Tag	TAG_UAFV1_DEREGISTER_CMD_RESPONSE。
1.1	UINT16 Length	响应长度。
1.2	UINT16 Tag	TAG_STATUS_CODE。
1.2.1	UINT16 Length	状态码长度。
1.2.2	UINT16 StatusCode	认证器返回状态码。

6.2.4.3 返回状态码

- UAF_CMD_STATUS_OK
- UAF_CMD_STATUS_ACCESS_DENIED
- UAF_CMD_STATUS_CMD_NOT_SUPPORTED
- UAF_CMD_STATUS_ERR_UNKNOWN

6.2.4.4 命令描述

认证器必须执行下列步骤：

1. 如果认证器具有交易确认显示并且能够显示 AppID，确保提供了 Command.TAG_APPID：
    - 通过将 Command.KHAccessToken 和 Command.TAG_APPID 混合来更新它。加密哈希函数是此类混合方法的例子。

- Command.KHAccessToken=hash(Command.KHAccessToken|Command.TAG_APPID)。

2. 如果认证器不在内部存储密钥句柄，返回 UAF_CMD_STATUS_CMD_NOT_SUPPORTED。
3. 找出与 Command.KeyID 匹配的密钥句柄。
4. 使用 Wrap.sym 来对密钥句柄解包。
5. 确认 RawKeyHandle.KHAccessToken == Command.KHAccessToken。
   - 否则，返回 UAF_CMD_STATUS_ACCESS_DENIED。
6. 从内部存储中删除此密钥句柄。
7. 返回 UAF_CMD_STATUS_OK。

规范

如果密钥句柄访问令牌验证失败，绑定认证器不得处理注销命令。

注销命令不应该明确显示提供的密钥标识符是否已注册。

6.2.5 OpenSettings 命令

此命令指导认证器打开其内置交互界面设置（例如，改变 PIN 码，注册新的指纹等）。

如果认证器不支持这个功能，必须返回 UAF_CMD_STATUS_CMD_NOT_SUPPORTED。

6.2.5.1 命令结构

TLV 结构		描述
1	UINT16 Tag	TAG_UAFV1_OPEN_SETTINGS_CMD。
1.1	UINT16 Length	完整命令长度。
1.2	UINT16 Tag	TAG_AUTHENTICATOR_INDEX。
1.2.1	UINT16 Length	认证器索引长度（必须为 0x0001）。
1.2.2	UINT8 AuthenticatorIndex	认证器索引。

6.2.5.2 命令响应

TLV 结构		描述
1	UINT16 Tag	TAG_UAFV1_OPEN_SETTINGS_CMD_RESPONSE。
1.1	UINT16 Length	完整命令响应长度。
1.2	UINT16 Tag	TAG_STATUS_CODE。
1.2.1	UINT16 Length	状态码长度。
1.2.2	UINT16 StatusCode	认证器返回的状态码。

6.2.5.3 返回的状态码

I. UAF_CMD_STATUS_OK

II. UAF_CMD_STATUS_CMD_NOT_SUPPORTED

III. UAF_CMD_STATUS_ERR_UNKNOWN

7. 密钥标识符和密钥句柄

本节是非规范性的。

文档中定义了 4 种类型的认证器，由于它们之间的特性差异，在处理命令时它们的行为是不同的。它们的一个主要差异就在于它们怎样存储和处理密钥句柄。本节将通过描述不同类型的认证器在处理相关控制命令时的行为，来阐述该差异。

7.1 第一因子绑定认证器

注册命令	认证器不会存储密钥句柄。密钥句柄返回给 ASM 并存在 ASM 的数据库中。 密钥标识符是随机生成的 32 字节的数（或者就是密钥句柄的哈希值）。
签名命令	当没有用户会话（没有 cookies，干净的机器），服务器不提供密

	钥标识符（因服务器不知道提供哪个密钥标识符）。这种情况 ASM 选择所有的密钥句柄，并传递给认证器。 在递进式鉴别中（具有用户会话），服务器提供相关密钥标识符。ASM 选择与提供的密钥标识符对应的密钥句柄然后将其传给认证器。
注销命令	由于认证器不会存储密钥句柄，认证内器没有要删除的内容。 ASM 找到与提供的密钥标识符相关的密钥句柄然后将其删除。

7.2 第二因子绑定认证器

注册命令	认证器不会存储密钥句柄。密钥句柄返回给 ASM 并存在 ASM 的数据库中。 密钥标识符是随机生成的 32 字节的数（或者就是密钥句柄的哈希值）。
签名命令	认证器在没有服务器提供的密钥标识符的情况下不能运行。因此认证器在没有用户会话（没有 cookies，干净的机器）时不能使用。 在递进式鉴别中（具有用户会话），服务器提供相关密钥标识符。ASM 挑选与提供的密钥标识符对应的密钥句柄并将其传给认证器。
注销命令	由于认证器不会存储密钥句柄，认证内器没有要删除的内容。 ASM 找到与提供的密钥标识符相关的密钥句柄然后将其删除。

7.3 第一因子漫游认证器

注册命令	认证器在内部存储密钥句柄。密钥句柄不会返回给 ASM。 密钥标识符是随机生成的 32 字节的数（或者就是密钥句柄的哈希值）。
签名命令	当没有用户会话（没有 cookies，干净的机器）时，服务器不提供密钥标识符（不知道提供哪个密钥标识符）。在这种情况下，认证器使用与提供的密钥标识符相关的所有密钥句柄。

	在递进式鉴别过程中（具有用户会话机制），服务器提供相关密钥标识符。认证器挑选与提供的密钥标识符相关的密钥句柄并使用它们。
注销命令	认证器找出正确的密钥句柄，在内部存储中将其删除。

7.4 第二因子漫游认证器

注册命令	认证器和 ASM 都不存储密钥句柄。发送密钥句柄给服务器（代替密钥标识符）并存储在用户的记录中。从服务器来看它是密钥标识符。事实上密钥标识符即是密钥句柄。
签名命令	认证器在没有服务器提供的密钥标识符情况下不能运行。因此在没有用户会话（没有 cookies，干净的机器）时认证器不能使用。在递进式鉴别过程中，服务器提供密钥标识符（即是密钥句柄）。认证器找到正确的密钥句柄并使用它。
注销命令	由于认证器和 ASM 不会存储密钥句柄，所以客户端没有要删除的内容。

8. 命令的访问控制

本节是规范性的。

FIDO 认证器可能会采用多种方式来进行特权命令的访问保护。

下表总结了每个命令的访问控制要求。

所有的 UAF 认证器必须满足下面定义的访问控制机制。

认证器供应商可能提供附加的安全机制。

表中使用的项目:

- NoAuth - 没有访问控制
- UserVerify - 明确的用户验证机制
- KHAccessToken - 访问者必须确认的访问令牌机制
- KeyHandleList - 访问者必须确认的访问句柄机制

- KeyID - 访问者必须确认的密钥标识符机制

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
Sign	UserVerify KHAccessToken KeyHandleList	UserVerify KHAccessToken KeyHandleList	UserVerify KHAccessToken	UserVerify KHAccessToken KeyHandleList
Deregister	KHAccessToken KeyID	KHAccessToken KeyID	KHAccessToken KeyID	KHAccessToken KeyID

表 1：命令的访问控制

9. 与其它规范的关系

本节是非规范性的。

和 UAF 认证器相关的技术规范是[TPM], [TEE]和[SecureElement]。

实现这些机制的硬件模块可以通过他们自有的扩展方式与 UAF 功能相结合，例如通过在模块内置安全应用（trustlets, applets 等）。不支持这些扩展机制的模块将无法在 UAF 框架中得到充分利用。

9.1 可信执行环境

为了在 TEE 中支持 UAF，需要设计特别的 Trustlet (TEE 中的可信应用) ，它可以实现本文档中规定的 UAF 认证器功能，也能实现一些类型的用户校验技术（生物识别校验，PIN 码校验或其它）。

必须额外提供一个和 TEE 认证器协同工作的 ASM。

9.2 安全元件

为了在安全元件（SE）中支持 UAF，需要设计特别的 Applet (SE 中的可信应用)，它可以实现本文档中规定的 UAF 认证器功能，也能实现一些类型的用户校验技术（生物识别校验，PIN 码校验或其它类似机制）。
必须额外提供一个和 SE 认证器协同工作的的 ASM。

9.3 可信平台模块

TPMs 通常具有内置的鉴证功能。但是，当前 TPMs 中的鉴证模块和 UAF 的鉴证模块是不兼容的。UAF 未来的改进可能会包含兼容鉴证方案。
TPMs 通常有一种内置的，可以被 UAF 利用的 PIN 码校验功能。为了在当前的 TPM 模块上支持 UAF，供应商需要编写具有下列特性的 ASM：

- 通过调用 TPM APIs 将 UAF 数据转换为 TPM 数据。
- 使用 TPMs API 创建断言。
- 把自身作为有效的 UAF 认证器报告给 FIDO UAF 客户端。

FIDO 联盟必须创造（参考[UAFAuthnrMetadata]）并发布一种专为 TPMs 设计的特殊断言。当 FIDO 服务器收到此断言方案的断言时，它会将接收到的数据看作 TPM 生成的数据并进行相应的解析或验证。

9.4 不可靠传输

本文档中描述的命令结构假定了一个可信传输，但是其不支持在应用层发现问题和解决问题，例如不可靠排序、不可靠丢弃、不可靠修改或对消息的不可靠修改。如果 ASM 和认证器之间的传输层是不可靠的，ASM 和认证器之间的非规范性的私有协议需要检测并改正这些错误。

A. 安全准则

本节是非规范性的。

类别	指导方针
应用标识符和密钥标识符	在没有经过用户校验之前，认证器不允许在明文中返回应用标识符和密钥标识符。 如果攻击者得到了漫游认证器的物理控制权，读取应用标识符和密钥标识符仍然是很困难的。
鉴证私钥	认证器应该把鉴证私钥作为非常敏感的数据进行保护。认证器的整体安全性取决于这些密钥的安全保护级别。 推荐在防篡改的硬件模块（如[SecureElement]）中存储和操作此密钥。 注册断言假定认证器对被鉴证密钥签名的数据有独有控制。 FIDO 认证器必须保证鉴证私钥： 1. 仅仅用来鉴证鉴别密钥，该鉴别密钥是由认证器使用 FIDO 定义的数据结构和 KeyRegistrationData 生成和保护的。 2. 在认证器的安全边界外不可访问。 在两个依赖方不能联合注册、鉴别或者进行其他交易的前提下才可以实现鉴证（参考[UAFProtocol]）。
产品认证	供应商应该使认证器尽量通过各种安全标准认证，例如[FIPS140-2]，[CommonCriteria]或其它类似的。通过此类认证对认证器的 UAF 应用起着积极的推动作用。
密码算法内核	密码算法内核是一个认证器模块，认证器在此模块上实现对 UAF 来说是必要的加密功能（密钥生成、签名、包裹等），并且从该模块访问鉴别私钥、鉴证私钥和包裹密钥。 对于安全可选方案，此模块应该和鉴别私钥、鉴证私钥和包裹密钥具有相同的安全界限。如果它在一个不同的安全界限内，

	那么实现方案必须保证其具有与在相同的模块内同样安全级别。 强烈建议在可信执行环境中[TEE]生成、存储和操作密钥。 在可能受到物理攻击和旁路攻击的情形下，推荐使用防篡改硬件模块。 基于软件的认证器必须保证使用编译保护和编译混淆技术来保护此模块，并且使用白盒加密技术来保护相关的密钥。 认证器需要采用高质量熵源的随机数生成器，这样能够： 1. 生成鉴别密钥。 2. 生成签名。 3. 计算认证器生成的挑战。 认证器随机数生成器（RNG）应该满足：不能被控制或破坏从而使得它产生可预测的输出。 如果认证器没有足够的熵生成强随机的数，认证器会破坏安全性。查看 随机数 获得更多信息。
密钥句柄	强烈建议在将密钥句柄与包裹秘钥绑定时使用可信赖加密算法。如 AES-GCM 和 AES-CCM 算法就非常适合此操作。
活体检测	用户校验方法应该包含活体检测[NSTCBiometrics]，即用来确认提交的样本来自真正的（活体）用户的技术。 在基于 PIN 码匹配的场景中，为了保证恶意软件不能模仿 PIN 码输入，应该使用 TEE 安全显示[TEESecureDisplay]。
匹配器	从定义上讲，匹配器组件是认证器的一部分。这没有给认证器的实现增加任何限制，但是实现者应该确保有合适的安全界限来绑定匹配器和认证器的其他部分。 对匹配器模块的篡改可能导致严重的安全隐患。强烈建议此模块放置在认证器安全边界内，并且具备检测篡改的能力。 强烈建议在可信执行环境[TEE]或安全元件[SecureElement]中运行匹配器模块。

	具有分离的匹配器和加密内核模块的认证器应该实现这种机制：允许加密内核模块安全接收来自于指示用户本地验证状态的匹配模块的断言。 基于软件的认证器（如果不在可信执行环境中）必须确保使用编译保护和编译混淆技术来保护该模块。 当认证器收到无效的用户校验令牌时，它应该将其看作攻击，并使获得的用户校验令牌失效。 用户校验令牌应该具有不超过 10 秒的生存周期。 认证器必须为其匹配器实现反恶意攻击防护。 基于生物特征的认证器必须保护获取的生物特征数据（例如，指纹）以及参考数据（模板），并确保生物特征数据不会离开认证器的安全界限。 匹配器必须只接收由用户注册的校验参考数据，即它们不得包含默认的 PIN 码和默认的生物参考数据。
私钥(鉴别私钥和鉴证私钥)	本文档要求：（a）鉴证密钥只能用在鉴证过程中（b）鉴别密钥只能用在 FIDO 鉴别过程中。相关需要签名的对象（例如密钥注册数据和签名数据）被设计来减少下面攻击类型的可能性： 1. 它们具有特别 FIDO 对象标注的标签。 2. 它们包含认证器生成的随机数。因此，所有的需要签名的对象具有高概率是唯一的。 3. 他们具有一个结构，该结构允许非常少的字段包含不受控制的值，即这个值既不是认证器生成的，也没有经过认证器验证。
随机数	FIDO 认证器使用随机数生成器产生鉴别密钥对、客户端挑战值，也有可能创建椭圆曲线数字签名算法（ECDSA）的签名。弱随机数会使得 FIDO 易受某些攻击。对 FIDO 认证器来说，好的随机数是非常重要的。

	认证器使用的（伪）随机数应该成功通过[Coron99]中声明的随机性检测，并且满足 [SP800-90b]中给定的指导原则。 此外，认证器还可能选择使用 FIDO 服务器通过请求中发送的 ServerChallenge 中提供的混合熵生成随机数（参考[UAFProtocol]）。 当混合多个熵源时，应该使用合适的混合函数，如[RFC4086]中描述的。
注册计数器	RegCounter 为依赖方提供反欺诈信号。依赖方可以通过使用 RegCounter检测出被重复注册的认证器。 如果使用 RegCounter，确保： 1. 任意注册命令操作都会使计数器增加。 2. 不能被操作或修改（例如通过 API 调用等）。 注册计数器应该作为全局计数器使用，即它会覆盖所有 AppIDs 调用的注册。进行任何注册操作，此全局计数器应该加 1。 注释：Deregistration命令操作不会减少注册计数器的值。
签名计数器	当攻击者能够从注册过的认证器中提取鉴别私钥时，此密钥可以独立于原始认证器使用。这被认为是对认证器的克隆。 好好保护鉴别私钥是保护避免克隆认证器的一个方法。在某些情况下，保护方法是不够充分的。 如果认证器维护签名计数器 SignCounter，那么 FIDO 服务器会有更好的方法来检测克隆认证器。 如果使用 SignCounter，要确保： 1. 任意鉴别或交易确认操作都会使计数器增加。 2. 不能被操作或修改（例如通过 API 调用等）。 为了保护用户的隐私，应该为每个鉴别私钥单独提供签名计数器。 无论何时，当对应鉴别密钥签名了断言时，与这个密钥对应

	的 SignCounter应该增加 1。 无论何时，当对应鉴别密钥被删除时，与这个密钥对应的 SignCounter应该被删除。 如果认证器不能处理很多不同的签名计数器，那么应该实现支持所有私钥的全局签名计数器。当鉴别密钥进行签名声明时，全局 SignCounter应该增加随机正数值。
交易确认显示	交易确认显示必须确保用户在场并且看到提供的交易确认内容，例如，交易确认内容是没有被其他的显示元素覆盖并且是清晰可辨的。有关威胁和应对措施的例子详见[CLICKJACKING]。参考 [TEESecureDisplay]获取更多的指导。
鉴别私钥	认证器必须把鉴别私钥作为**最**敏感的数据进行保护。认证器的整体安全性很大程度上取决于这些密钥的安全保护级别。 强烈建议在可信执行环境中生成、存储、操作此密钥。 在可能受到物理攻击和旁路攻击的情形下，推荐使用防篡改硬件模块。 FIDO 认证器必须确保鉴别私钥： 1. 对依赖方的指定账户来说是唯一的（依赖方通过 AppID 定义）。 2. 生成基于足够的熵生成的好随机数。FIDO 服务器在注册和认证操作提供的挑战值应该在熵池中进行混合运算以提供额外的熵。 3. 不直接显示，即总是在 FIDO 认证器的专属控制下。 4. 仅在明确的鉴别模式过程中使用，包括： 1. 鉴别到对应的应用（通过 AppID 定义的）。 2. 交易确认到对应的应用（通过 AppID 定义的）。 3. 仅用来创建 FIDO 定义的数据结构，如 KRD, SignData。

用户名	用户名在任何情况下不得明文返回，除了 SIGN 命令描述的情况。在任何其他情况中，用户名必须与 KeyHandle一起存储。
校验参考数据	校验参考数据（例如指纹模板或 PIN 参考值）被定义为认证器的一部分。这没有给认证器的实现增加任何限制，但是实现者应该确保有合适的安全边界来绑定匹配器和认证器的其他部分。
包裹密钥	如果认证器有一个包裹密钥（Wrap.sym），认证器应该把包裹密钥作为非常敏感的数据进行保护。认证器的整体安全性取决于这些密钥的安全保护级别。 包裹密钥强度必须等于或高于与存储在 RawKeyHandle 中的秘密的强度。参考[SP800-57]和[SP800-38F]获取关于选择正确保护算法和正确执行方法的信息。 强烈建议在可信执行环境中生成、存储和操作该密钥。 在可能受到物理攻击和旁路攻击的情形下，推荐使用防篡改硬件模块。 如果认证器使用包裹密钥，它必须确保解包错误密钥句柄和包含无效内容（例如无效源提供的密钥句柄）的数据对调用方是无法分辨的。

B. 图表列表

C. 参考文献

C.1 参考规范

[Coron99]

J. Coron and D. Naccache An accurate evaluation of Maurer's universal test. LNCS 1556, February 1999,

URL: http://www.jscoron.fr/publications/universal.pdf

[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

[ITU-X690-2008]

X.690: Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), (T-REC-X.690-200811). International Telecommunications Union, November 2008

URL: http://www.itu.int/rec/T-REC-X.690-200811-I/en

[SP800-90b]

Elaine Baker and John Kelsey, NIST Special Publication 800-90b: Recommendation for the Entropy Sources Used for Random Bit Generation. National Institute of Standards and Technology, August 2012, URL: http://csrc.nist.gov/publications/drafts/800-90/draft-sp800-90b.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

C.2 参考资料

[CLICKJACKING]

D. Lin-Shung Huang, C. Jackson, A. Moshchuk, H. Wang, S. Schlechter Clickjacking: Attacks and Defenses. USENIX, July 2012, URL: https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final39.pdf

[CommonCriteria]

CommonCriteria Publications. CCRA Members, Work in progress, accessed March 2014. URL: http://www.commoncriteriaportal.org/cc/

[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

[FIPS140-2]

FIPS PUB 140-2: Security Requirements for Cryptographic Modules. National Institute of Standards and Technology, May 2001,

URL: http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

[NSTCBiometrics]

NSTC Subcommittee on Biometrics, Biometrics Glossary. National Science and Technology Council. 14 September 2006, URL: http://biometrics.gov/Documents/Glossary.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

[RFC4086]

D. Eastlake 3rd, J. Schiller, S. Crocker Randomness Requirements for Security (RFC 4086), IETF, June 2005, URL: http://www.ietf.org/rfc/rfc4086.txt

[SP800-38F]

M. Dworkin, NIST Special Publication 800-38F: Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping. National Institute of Standards and Technology, December 2012, URL: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf

[SP800-57]

Recommendation for Key Management – Part 1: General (Revision 3). SP800-57. July 2012. U.S. Department of Commerce/National Institute of Standards and Technology. URL: http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf

[SecureElement]

GlobalPlatform Card Specifications GlobalPlatform. Accessed March 2014. URL: https://www.globalplatform.org/specifications.asp

[TEE]

GlobalPlatform Trusted Execution Environment Specifications GlobalPlatform. Accessed March 2014.

URL: https://www.globalplatform.org/specifications.asp

[TEESecureDisplay]

GlobalPlatform Trusted User Interface API Specifications

GlobalPlatform. Accessed March 2014.

URL: https://www.globalplatform.org/specifications.asp

[TPM]

TPM Main Specification Trusted Computing Group. Accessed March 2014.

URL: http://www.trustedcomputinggroup.org/resources/tpm_main_specification