FIDO UAF 协议规范 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-protocol-v1.0-ps-20141208.html

之前版本:

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

编写者:

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

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

艾瑞克·蒂芙尼（Eric Tiffany）, FIDO 联盟

贡献者:

迪尔克·巴尔凡茨（Dirk Balfanz）,谷歌（Google, Inc.）

布拉德•希尔 (Brad Hill)，贝宝（PayPal, Inc.）

杰夫•霍奇斯（Jeff Hodges），贝宝（PayPal, Inc.）

翻译者:

李俊（Simon Li），联想（Lenovo）

常秉俭（Nick Chang），联想（Lenovo）

本规范的英文版本是唯一官方标准；可能会存在非官方的译本。

摘要

通用鉴别框架（UAF）的目标是提供统一的可扩展的鉴别机制，该鉴别机制可

在避免其他现有鉴别方式的缺点的同时替代口令。

FIDO UAF 的设计目的主要是允许依赖方为其最终用户或交互选择现有最佳的鉴别机制，同时兼顾未来能够利用新兴设备的安全性能，而无需进行额外的集成。

本文档详细描述了 FIDO 架构，定义了所有 UAF 协议的消息流和内容，并阐述了设计选择背后的理论基础。

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

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，例如“UAF-TLV”。

公式中用 “|” 来表示按字节串联操作。

base64url 符号引用自*无填充*的“Base 64 Encoding with URL and Filename Safe Alphabet”[RFC4648]。

根据[WebIDL-ED]，字典（dictionary）成员是可选的，除非他们被明确标注为 required。

WebIDL 的字典（dictionary）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 的结构成员是一个表单，则不得为一个空表单。

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[WebIDL]，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。

通用鉴别框架（UAF）的目标是提供统一的可扩展的鉴别机制，该鉴别机制可在避免其他现有鉴别方式的缺点的同时替代口令。

UAF 协议的设计目的是使依赖方能够通过一套单一的统一的协议利用最终用户设备上多样化的异构的安全能力。

该协议主要是使依赖方为其用户或交互选择现有的最好的鉴别机制，同时也能够利用未来新兴设备的安全能力而无需进行额外的集成。

2.1 范围

文档详细描述了 FIDO 架构，并定义 UAF 协议为网络协议，文档同时定义了所有 UAF 消息流和内容，并阐述了设计选择背后的基本原理。

特定的应用程序级别的绑定超出了本文档的范围。本文档并不打算回答类似下面的问题：

- UAF 中的 HTTP 绑定会是什么样？
- Web 应用怎么与 FIDO UAF 客户端通信？
- FIDO UAF 客户端怎么与 FIDO 认证器通信？

这些问题的答案可以在其他的 UAF 规范中找到，例如 [UAFAppAPIAndTransport]、[UAFASM]、[UAFAuthnrCommands]。

2.2 架构

下图描绘了 UAF 协议中的所有实体。

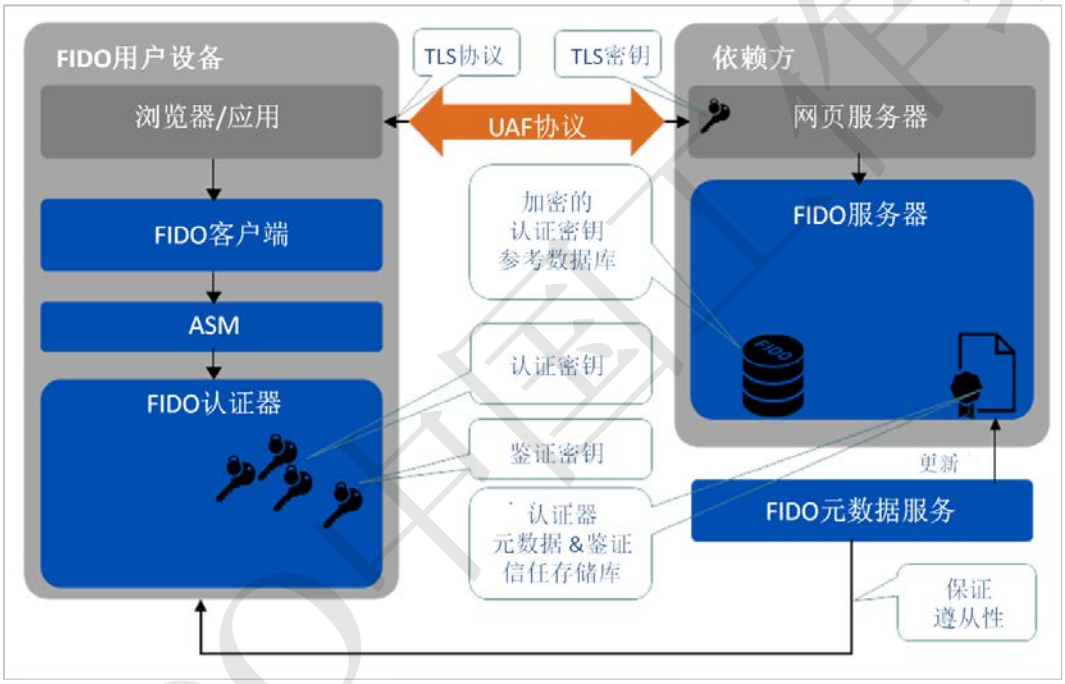

图 1 UAF 架构

在这些实体中，只有三个实体是直接创建或处理 UAF 协议消息的：

- FIDO 服务器，在依赖方的基础设施上运行；
- FIDO UAF 客户端，作为用户代理的一部分，运行在 FIDO 用户设备上；
- FIDO 认证器，集成在 FIDO 用户设备中。

本文档中假定 FIDO 服务器可以访问 UAF 认证器元数据[UAFAuthnrMetadata]，其中描述了所有将与 FIDO 服务器交互的认证器。

2.3 协议会话

UAF 核心协议包括四个 FIDO 客户端与 FIDO 服务器间的概念性会话。

- **注册**：UAF 协议允许依赖方使用依赖方的用户账户上注册 FIDO 认证器。依赖方可以制定策略来支持各种各样的 FIDO 认证器。FIDO UAF 客户端只注册与该策略相符合的现有的认证器。
- **鉴别**：UAF 协议允许依赖方提示最终用户使用之前注册过的 FIDO 认证器来进行鉴别。该鉴别可以在任何时候根据依赖方的意愿调用。
- **交易确认**：除了提供常规的鉴别提示，UAF 还提供提示用户确认特定交易的支持。

  该提示可以使用客户端交易确认显示，具有将附加信息传达给客户端并显示给最终用户的能力。该附加鉴别操作的目标是使依赖方能确保用户确认特定的交易信息（而不是鉴别一个与用户代理的会话）。
- **注销**：依赖方可以触发账户相关鉴别密钥材料的删除。

虽然本文档定义了 FIDO 服务器作为请求的发起者，实际情况中，UAF 的首个操作一般总是在用户代理（例如 HTTP）对依赖方的请求之后。

后面的小节对单个操作的协议会话进行了简要概述，更多详细描述参考注册操作，鉴别操作和注销操作。

2.3.1 注册

下图展示了 UAF 注册的消息流。

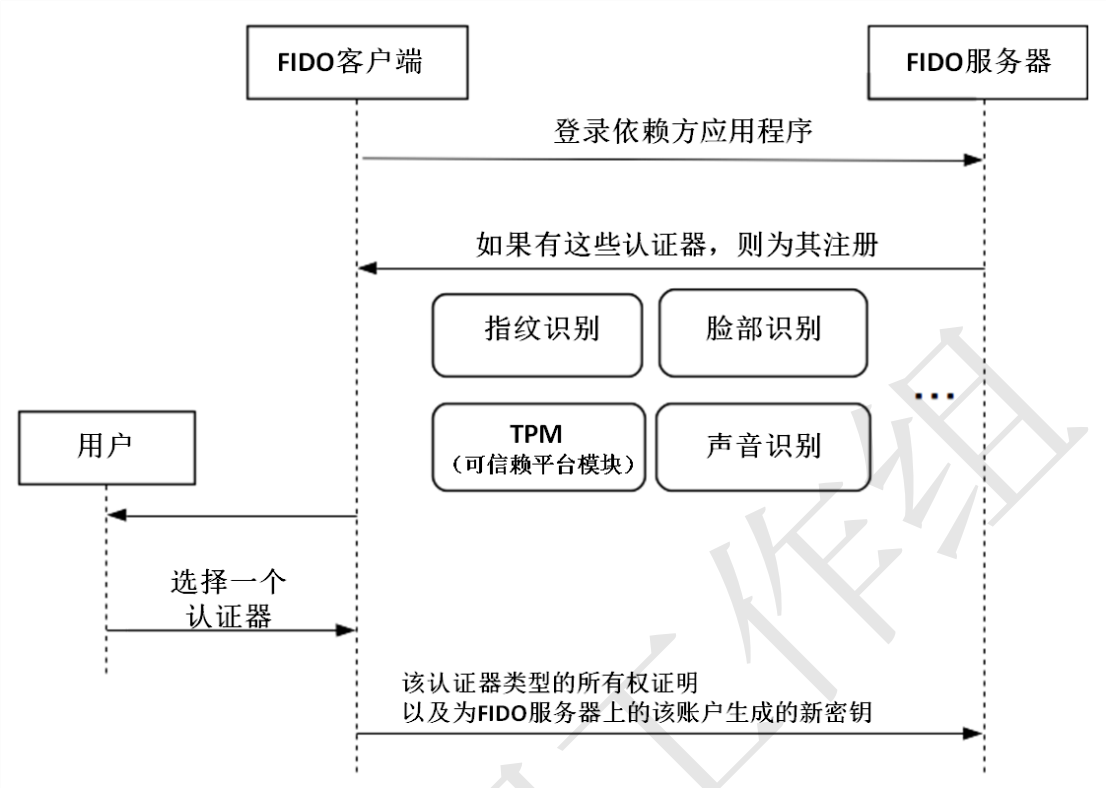

图 2 UAF 注册消息流

> 注释
>
> 客户端应用应当使用合适的 API 告诉 FIDO UAF 客户端操作的结果（参考[UAFAppAPIAndTransport]的 2.3.1 节），从而允许 FIDO UAF 客户端做一些“内务处理”任务。

2.3.2 鉴别

下图描绘了鉴别操作的消息流。

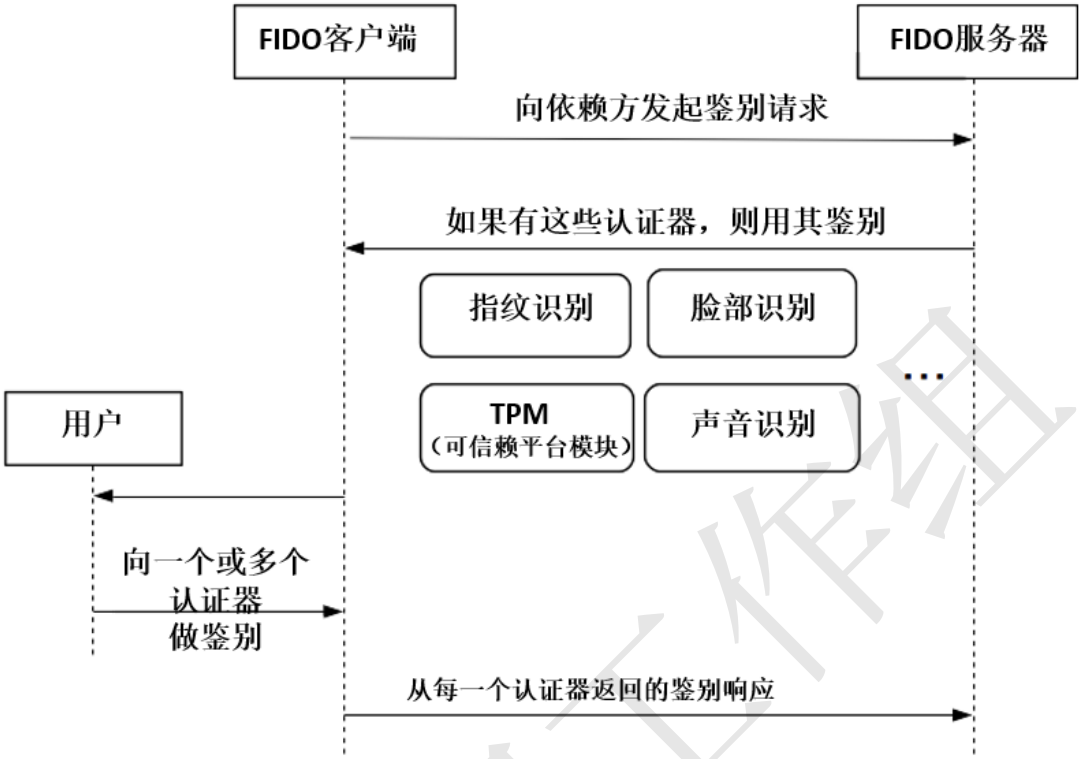

图 3 鉴别消息流

> 注释
>
> 客户端应用应当使用合适的 API 告诉 FIDO UAF 客户端操作的结果（参考[UAFAppAPIAndTransport]的 2.3.1 节），从而允许 FIDO UAF 客户端做一些“内务处理”任务。

2.3.3 交易确认

下图描绘了交易确认的消息流。

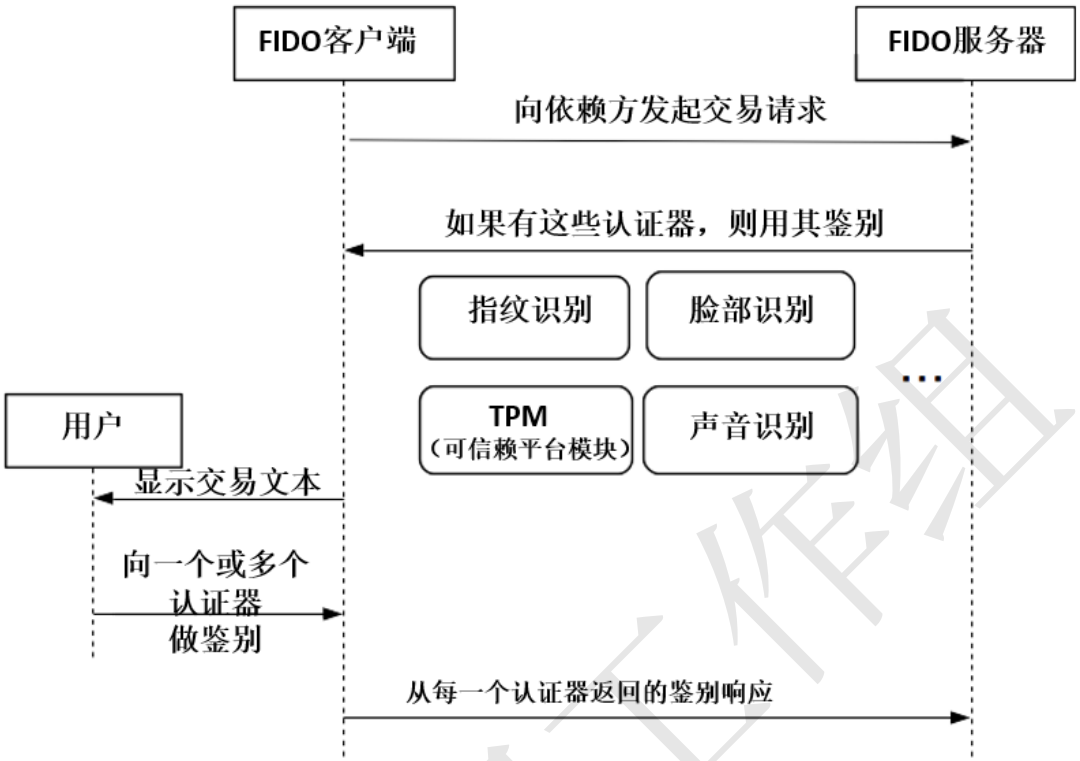

图 4 交易确认消息流

> 注释
>
> 客户端应用应当使用合适的 API 告诉 FIDO UAF 客户端操作的结果（参考 [UAFAppAPIAndTransport]的 2.3.1 节），从而允许 FIDO UAF 客户端做一些“内务处理”任务。

2.3.4 注销

下图描绘了注销的消息流。

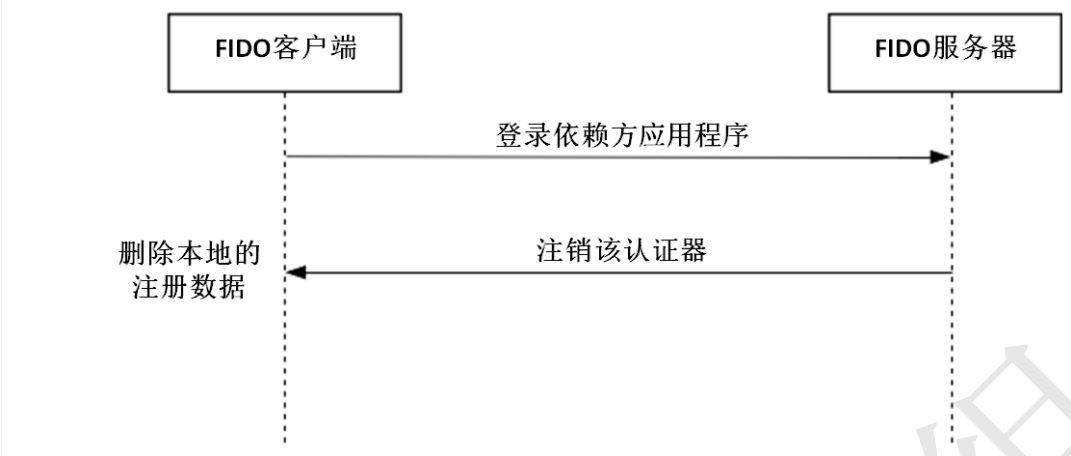

图 5 注销消息流

> **注释**
>
> 客户端应用应当使用合适的 API 告诉 FIDO UAF 客户端操作的结果（参考 [UAFAppAPIAndTransport]的 2.3.1 节），从而允许 FIDO UAF 客户端做一些“内务处理”任务。

3. 协议细节

本节是标准化的。

本节详细描述了 UAF 协议支持的操作。

除非明确声明，支持符合软件的所有协议元素是强制性的。

规范中所有的字符串都是根据 U+0000..U+007F Unicode 字符标准构建的。

除非另外说明，协议消息用 UTF-8 编码传送。

> **注释**
>
> 为了遵守[FIDOSecRef]中的设定，协议中所用数据必须用 FIDO UAF 客户端和依赖方之间建立的安全传输协议（例如 TLS/HTTPS）进行传输，参考 4.1.6 TLS 通信保护.

base64url(byte[8..64])代表 8-64 字节的数据用 base64url 形式编码。引用自*无填充*的"Base 64 Encoding with URL and Filename Safe Alphabet" [RFC4648]。

string[5] 是 5 个 Unicode 字符，声明中代表 UTF-8[RFC3629]已编码的字符串类型，通常是 WebIDL [WebIDL-ED] DOMString。

正如 UTF-8 代表可变长度，string[5]的最大字节长度是 string[4*5]。

除非另外声明，所有字符串都是区分大小写的。

本文档用 WebIDL [WebIDL-ED]定义 UAF 协议消息。

具体实现必须使用 UTF-8 编码的 JSON[RFC4627]序列化 UAF 协议消息用于传输。

3.1 共享结构和类型

本节定义了不同操作共享的类型和结构。

3.1.1 版本接口

描述了一个带有主要和次要字段的通用版本号。

WebIDL

```
interface Version{
    readonly attribute unsigned short     major;
    readonly attribute unsigned short     minor;
};
```

3.1.1.1 属性

major 类型为 unsigned short，只读

主要版本，本规范中是 1。

minor 类型为 unsigned short，只读

次要版本，本规范中是 0。

3.1.2 操作枚举

描述 UAF 消息或者消息请求的操作类型。

WebIDL

```
enum Operation{
    "Reg",
    "Auth",
    "Dereg"
};
```

枚举描述	
Reg	注册
Auth	鉴别或交易确认
Dereg	注销

3.1.3 OperationHeader 结构

描述了 UAF 消息请求和响应报头。

```
WebIDL
    dictionary OperationHeader{
        required Version        upv;
        required Operation      op;
        DOMString               appID;
        DOMString               serverData;
        Extension[]             exts;
    };
```

3.1.3.1 OperationHeader 结构成员

upv 类型为 required Version

UAF 协议版本。主要版本必须是 1，次要版本必须是 0。

op 类型为 required Operation

与该消息相关的 FIDO 操作名。

> **注释**
>
> "Auth"既用于鉴别也用于交易确认。

appID 类型为 DOMString

string[0..512]。

应用标识符是由依赖方声明的。

有三种方法可以设置 AppID [FIDOAppIDAndFacets]：

1. 如果请求中的元素缺失或为空，FIDO UAF 客户端必须设置 AppID

为请求者的 FacetID 。

2. 如果消息中的 appID 与请求者的 FacetID 相同，那么 FIDO UAF 客户端必须接受。
3. 如果是 HTTPS 协议的 URI，FIDO UAF 客户端必须使用其从特定的 URI 载入可信类型标识符列表。如果请求者的类型识别符与间接引用 URI 返回的列表中的任意一个可信类型识别符匹配，FIDO UAF 客户端必须接受这个请求。

> **注释**
>
> 认证器生成的新的密钥对会与应用识别符相关联。
>
> 安全性相关：FIDO UAF 客户端使用应用识别符来验证该应用是否有资格触发使用特定的 UAuth.Key。参考[FIDOAppIDAndFacets]。

serverData 类型为 DOMString

string[1..1536]。

依赖方创建的会话标识符。

> **注释**
>
> 依赖方可以不透明地存储注册会话有效期、使用的协议版本和 serverData 中其他有用信息。数据对 FIDO UAF 客户端是不透明的。FIDO 服务器可能会拒绝缺少数据或者包含非授权修改的响应。
>
> 依赖 serverData 完整性的服务器应该采用合适的安全措施，正如 FIDO 服务器注册请求生成规则和服务器数据和秘钥句柄章节中描述的一样。

exts 类型为 *Extension* 数组

UAF 扩展消息列表。

3.1.4 认证器鉴证标识符（AAID）类型定义

WebIDL

```
typedef DOMString AAID;
```

string[9]。

每个认证器必须有一个 AAID 用来全局地标识 UAF 认证器型号。AAID 必须能在所有认证器供应商制造的 UAF 认证器类型中识别出特定类型，特定类型的认

证器必须共享相同的安全特征（参考安全注意事项）。

AAID 是"V#M"字符串格式的，其中

"#"是分隔符；

"V" 代表认证器供应商代码，此代码包含 4 个十六进制数字；

"M"代表认证器型号代码，此代码包含 4 个十六进制数字。

AAID 的扩展巴科斯范式 [ABNF] 是：

```
AAID = 4(HEXDIG) "#" 4(HEXDIG)
```

> **注释**
>
> HEXDIG 是不区分大小写的，例如"03EF"和"03ef"是等同的。

FIDO 联盟负责分配认证器供应商代码。

认证器供应商负责分配认证器型号代码给认证器。认证器供应商必须分配唯一的 AAID 给有不同安全特征的认证器。

AAID 都是唯一的，每个 AAID 必须与不同的鉴别元数据文件相关联（[UAFAuthnrMetadata]）。

> **注释**
>
> 增加新的固件或软件功能、或改变底层的硬件保护机制通常会改变认证器的安全特征，因此需要使用一个新的 AAID 。参考([UAFAuthnrMetadata])。

3.1.5 KeyID 类型定义

WebIDL

```
typedef DOMString KeyID;
```

base64url(byte[32...2048])

KeyID 是特定 UAuth.Key 的唯一标识符（在一个 AAID 范围内）。KeyID 是由认证器生成，并在 FIDO 服务器完成注册。

(AAID, KeyID) 元组必须是认证器在依赖方注册的唯一标识。当 FIDO 服务器想要提供给特定的认证器特定的信息，就必须使用(AAID, KeyID) 元组。

KeyID 必须是 base64url 编码且包含在 UAF 消息中。

在递进式鉴证和注销的操作中，FIDO 服务器应该将 KeyID 提供给认证器以便后者能够定位合适的用户鉴证密钥并用其执行必要的操作 。

漫游认证器不含有内部存储并且不能依赖任何 ASM 来存储，其应该在注册操

作的过程中生成密钥句柄并且将其作为 AuthenticatorRegistrationAssertion.assertion.KeyID 的一部分（也可参考服务器数据和密钥句柄章节），在递进式鉴证（参考策略中的 MatchCriteria 结构）或注销操作（更多细节，可参考[UAFAuthnrCommands]）中，应该从 FIDO 服务器取回密钥句柄。

> **注释**
>
> KeyID 的确切结构和内容是特定于认证器的实现的。

3.1.6 ServerChallenge 类型定义

WebIDL

```
typedef DOMString ServerChallenge;
```

base64url(byte[8...64])。

ServerChallenge 是服务器提供的随机挑战。*安全性相关*：FIDO 服务器用挑战来验证传入响应是新的还是已经被处理过的。参考重放攻击保护章节。

ServerChallenge 应该混合到认证器的熵池中。*安全性相关*：无论何时，FIDO 服务器都应该尽可能地提供包含强加密随机性的挑战。参考服务器挑战和随机数章节。

> **注释**
>
> 遵循[SP800-63] 的要求，8 字节是挑战的最小长度，相当于[RFC6287]中要求的 20 位十进制数。

> **注释**
>
> 服务器挑战定义了最大长度，由此 SHA-512 的输出可以在不被截断的情况下使用。

> **注释**
>
> 正如[RFC4086]中描述的，建议将多个随机源产生的随机数混合，用来提高认证器生成的随机数的质量。

3.1.7 FinalChallengeParams 结构

```
WebIDL
dictionary FinalChallengeParams{
    required DOMString          appID;
    required ServerChallenge    challenge;
    required DOMString          facetID;
    required ChannelBinding     channelBinding;
};
```

3.1.7.1 FinalChallengeParams 结构成员

appID 类型为 required DOMString

string[1..512]。

该值必须取自 **OperationHeader** 的 appID 字段。

challenge 类型为 required ServerChallenge

该值必须取自请求的挑战字段（例如 RegistrationRequest.challenge，AuthenticationRequest.challenge）。

facetID 类型为 required DOMString

string[1..512]。

该值由 FIDO UAF 客户端决定并取决于调用的应用程序。参考[FIDOAppIDAndFacets]。*安全性相关*：facetID 由 FIDO UAF 客户端决定，用来验证通过间接引用调用应用程序 appID 检索到的可信类型列表。

channelBinding 类型为 required ChannelBinding

包含 FIDO 客户端发给 FIDO 服务器的 TLS 消息，绑定 TLS 通道和 FIDO 操作。

3.1.8 ChannelBinding 结构

ChannelBinding 包含通道绑定信息[RFC5056]。

注释

安全性相关: FIDO 服务器可能会验证通道绑定操作从而检测和预防 MIMT 攻击。

此时，可支持下面的通道绑定方法：

- TLS ChannelID(cid_pubkey) [ChannelID]
- serverEndPoint [RFC5929]
- tlsServerCertificate [RFC5929]
- tlsUnique [RFC5929]

进一步要求：

1. 如果数据与上述任意一种通道绑定方法有关，并且该方法在 FIDO UAF 客户端上可用，那么必须根据相关规范使用该方法。
2. FIDO 服务器必须支持上述的所有通道绑定方法。如果通道绑定验证失败，FIDO 服务器可能拒绝操作。

注释

- 如果网页浏览器或应用客户端可以访问通道绑定数据，该数据必须传递给 FIDO UAF 客户端以符合 [FIDOSecRef]中的假定。
- 如果网络服务器可以访问通道绑定数据，该数据必须传递给 FIDO 服务器以符合[FIDOSecRef]中的假定。FIDO 服务器依靠网络服务器提供精确的通道绑定信息。

WebIDL

```
dictionary ChannelBinding{
    DOMString     serverEndPoint;
    DOMString     tlsServerCertificate;
    DOMString     tlsUnique;
    DOMString     cid_pubkey;
};
```

3.1.8.1 ChannelBinding 结构成员

serverEndPoint 类型为 DOMString

如果 serverEndPoint 字段可用，则必须设置为 TLS 服务器证书哈希值的 base64url 编码。哈希函数必须按以下方式选择：

1. 如果证书的 signatureAlgorithm 使用单个哈希函数，并且哈希函数是 MD5 [RFC1321] 或 SHA-1 [RFC6234]，则使用 SHA-256 [FIPS180-4]；
2. 如果证书的 signatureAlgorithm 使用单个哈希函数，并且哈希函数既不是 MD5 [RFC1321] 也不是 SHA-1 [RFC6234]，则使用与证书的 signatureAlgorithm 相关的哈希函数；
3. 如果证书的 signatureAlgorithm 未使用哈希函数或者使用多个哈希函数，此类型通道绑定在当前版本未定义（如果遇到此类问题会对此类通道绑定进行更新以解决该问题）。

如果 TLS 服务器证书不能用于处理实体（例如 FIDO UAF 客户端），或不能确定为所描述的哈希函数，serverEndPoint 字段必须为空。

tlsServerCertificate 类型为 DOMString

如果 TLS 服务器证书对 FIDO UAF 客户端不可用，这个字段必须为空。

如果 TLS 服务器证书对 FIDO UAF 客户端可用，那么这个字段必须设置为 base64url 编码的 DER 编码后的 TLS 服务器证书。

tlsUnique 类型为 DOMString

必须设置为 base64url 编码的 TLS 通道 Finished 结构。但是如果该结构对 FIDO UAF 客户端不可用，则该字段必须为空[RFC5929]。

cid_pubkey 类型为 DOMString

如果客户端 TLS 栈不提供 TLS ChannelID [ChannelID]信息给处理实体（例如 web 浏览器或客户端应用），该字段必须设置为空。

如果 TLS ChannelID 信息被客户端 TLS 栈支持，但是没有被 TLS（网络）服务器签名，该字段必须设置为"unused"。

其余情况下，cid_pubkey 必须设置为 base64url 编码的经过序列化 [RFC4627]的使用 UTF8 编码的 JwkKey 结构。

3.1.9 JwkKey 结构

JwkKey 是一个用 JSON Web Key [JWK]编码的椭圆曲线公钥结构。

这个公钥是客户端 TLS 栈为特定的依赖方产生的 ChannelID 公钥。[ChannelID]规定只使用特定的椭圆曲线和特定的坐标类型。

WebIDL

```
dictionary JwkKey{
    required DOMString    kty = "EC";
    required DOMString    crv = "P-256";
    required DOMString    x;
    required DOMString    y;
};
```

3.1.9.1 JwkKey 结构成员

kty 类型为 required DOMString，默认值为 **"EC"**

代表 Channel ID 使用的密钥类型，目前[ChannelID]章节中只支持椭圆曲线，所以必须设为"EC" [JWA]。

crv 类型为 required DOMString，默认值为 **"P-256"**

代表定义公钥的椭圆曲线。目前[ChannelID] 章节中只支持美国国家标准及技术研究所（NIST）曲线 secp256r1，所以 crv 参数必须设置为"P-256"。

x 类型为 required DOMString

包含公钥（高位优先，32 字节值）x 轴坐标的 base64url 编码。

y 类型为 required DOMString

包含公钥（高位优先，32 字节值）y 轴坐标的 base64url 编码。

3.1.10 Extension 结构

FIDO 扩展在很多地方出现，包括 UAF 协议消息，认证器命令或由认证器签名的断言等。

每个扩展都有一个标识符，扩展标识符的命名空间是 FIDO UAF 全局的（例如命名空间不依赖于扩展消息在哪里出现）。

扩展的处理方式可被定义为：如果一个处理实体不理解特定扩展的意义，那么

它必须终止处理，或指定一种可以（安全）忽略未知扩展的方式。

在每个允许扩展的部分中都定义了扩展的处理规则。

通用扩展用于各种各样的操作中。

WebIDL

```
dictionary Extension{
    required DOMString    id;
    required DOMString    data;
    required boolean      fail_if_unknow;
};
```

3.1.10.1 Extension 结构成员

id 类型为 required DOMString

string[1..32]。

标识扩展。

data 类型为 required DOMString

包含服务器端和客户端之间语义一致的任意数据。该数据是以 base64url 编码的。

该字段可能为空。

fail_if_unknown 类型为 required Boolean

表明未知扩展必须被忽略(false)或必须导致一个错误(true)。

- false 值表示未知的扩展必须被忽略；
- true 值表示未知扩展必须导致一个错误。

> 注释
>
> FIDO UAF 客户端可能（a）处理扩展或（b）将扩展传递给 ASM。未知扩展必须予以传递通过。
>
> ASM 可能（a）处理扩展或（b）传递扩展给 FIDO 认证器。未知扩展必须予以传递通过。
>
> FIDO 认证器必须处理扩展或忽略（只有不知道如何处理而且没有设置 fail_if_unknown 时，才可以忽略）。如果 FIDO 认证器不能理解扩展的含

义，并且设定了 fail_if_unknown 值，就必须产生一个错误（参考上述对 fail_if_unknown 的定义）。

当传递扩展给下一个实体时，必须保留 fail_if_unknown 标记。（参考 [UAFASM] [UAFAuthnrCommands]）。

FIDO 协议消息是不签名的。如果安全性依赖于一个已知的或者处理过的扩展，那么该扩展应该在认证器断言中附有一个相关（有签名的）扩展（例如 TAG_UAFV1_REG_ASSERTION，TAG_UAFV1_AUTH_ASSERTION）。如果安全性提高了 (例如，FIDO 认证器根据元数据声明中的描述支持多个手指，但是在特定的情况下注册时使用的手指也用于鉴别)，就没有必要标记扩展为 fail_if_unknown（例如应该使用 0x3E1 标签 [UAFAuthnrCommands]）。如果安全性降低（例如 FIDO 认证器根据元数据声明中的描述只支持使用注册时的手指用于鉴别，但在特定的情况下使用不同的手指进行了鉴别），扩展必须标记为 fail_if_unknown(例如必须使用 tag 0x3E11 标签[UAFAuthnrCommands]）。

3.1.11 MatchCriteria 结构

代表用于服务器策略的匹配条件。

如果对象中的所有字段都被认为是匹配的（正如在特定字段中表明的一样），那么 MatchCriteria 对象则被认为是与一个认证器匹配的。

```
WebIDL
dictionary MatchCriteria{
    AAID[]              aaid;
    DOMString[]         vendorID;
    KeyID[]             keyIDs;
    unsigned long       userVerfication;
    unsigned short      keyProtection;
    unsigned short      matcherProtection;
    unsigned long       attachmentHint;
    unsigned short      tcDisplay;
    unsigned short[]    authenticationAlgorithms;
    DOMString[]         assertionSchemes;
    unsigned short[]    attestationTypes;
    unsigned short      authenticatorVersion;
    Extension[]         exts;
};
```

3.1.11.1 MatchCriteria 结构成员

aaid 类型为 *AAID* 数组

AAID 列表将匹配限制在某些特定 AAIDs 中。

如果至少一个数组中的 AAID 记录与 AuthenticatorInfo.aaid 相匹配，则匹配成功[UAFASM]。

> 注释
>
> 该字段与 MetadataStatement.aaid[UAFAuthnrMetadata]相对应。

vendorID 类型为 DOMString 数组

vendorID 将匹配限制在给定供应商的认证器型号中。AAID 的前 4 个字符是 vendorID（参考 **AAID**）。

如果至少一个 AAID 记录与数组中的 AuthenticatorInfo.aaid 的前 4 个字符相匹配，则匹配成功[UAFASM]。

> 注释
>
> 该字段与 MetadataStatement.aaid[UAFAuthnrMetadata]的前四个字符相对应。

keyIDs 类型为 *KeyID* 数组

认证器 KeyIDs 列表将被限制在一组给定的 KeyID 实例集中（参考 [UAFRegistry]）。

如果数组中至少一条记录匹配，则匹配成功。

> 注释
>
> 该字段与 AppRegistration.keyIDs 相对应[UAFASM]。

userVerification 类型为 unsigned long 数组

32 位标志的集合，当匹配应该被用户验证方法限制时需要设置（参考[UAFRegistry]）。

> 注释
>
> 如果以下条件成立，则与 AuthenticatorInfo.userVerification 匹配成功[UAFASM]（使用 JAVA 语言编写）：

```
if (
        // They are equal
        (AuthenticatorInfo.userVerification ==
MatchCriteria.userVerification) ||

        // USER_VERIFY_ALL is not set in both of them and they have
at least one common bit set
        (
            ((AuthenticatorInfo.userVerification &
USER_VERIFY_ALL) == 0) &&
            ((MatchCriteria.userVerification & USER_VERIFY_ALL)
== 0) &&
            ((AuthenticatorInfo.userVerification &
MatchCriteria.userVerification) != 0)
        )
    )
```

> 注释
>
> 该字段值按如下方式从 MetadataStatement.userVerificationDetails 中获得：
>
> 1.如果 MetadataStatement.userVerificationDetails 含有多条记录，那么：
>
> 1. 如果一条或多条记录 MetadataStatement.userVerificationDetails[i] 含有多条记录，那么立刻停止因为无法直接提取。必须通过提供匹配的 AAIDs 列表来产生 MatchCriteria 对象。
> 2. 如果所有 MetadataStatement.userVerificationDetails[i]只含有单条记录，那么用按位或操作将记录 MetadataStatement.userVerificationDetails[0][0].userVerification 到 MetadataStatement.userVerificationDetails[N-1][0].userVerification 的所有条目混合成为单独的值。
>
> 2.如果 MetadataStatement.userVerificationDetails 包含一条单独的记录，那么用按位或操作将记录 MetadataStatement.userVerificationDetails[0][0].userVerification 到 MetadataStatement.userVerificationDetails[N-1][0].userVerification 的所有条目混合成为单独的值，然后标记为 USER_VERIFY_ALL。
>
> 这个方法不允许匹配实现了用户验证方法的复杂组合的认证器，例如 PIN AND (Fingerprint OR Speaker Recognition)（参考上述推导原则）。如果这种特定的匹配规则是必需的，则需要通过提供匹配认证器的 AAID 来指定。

keyProtection 类型为 unsigned short

16 位的标记集，当匹配应该被使用的密钥保护限制时需要设置（参考 [UAFRegistry]）。

如果至少一位标记与 AuthenticatorInfo.keyProtection 的值匹配，则匹配成功 [UAFASM]。

> 注释
>
> 该字段与 MetadataStatement.keyProtection[UAFAuthnrMetadata]相对应。

matcherProtection 类型为 unsigned short

16 位的标记集，当匹配应该被匹配程序保护限制时需要设置。（参考[UAFRegistry]）。

如果至少一位标记与 AuthenticatorInfo.matcherProtection 的值相匹配，则匹配成功[UAFASM]。

> 注释
>
> 该字段与 MetadataStatement.matcherProtection 元数据声明相对应。参考[UAFAuthnrMetadata].

attachmentHint 类型为 unsigned long

32 位标记的集合，当匹配应该被认证器连接机制所限制时需要设置（参考[UAFRegistry]）。

如果至少一位标记与 AuthenticatorInfo.attachmentHint 值相匹配，则匹配成功[UAFASM]。

> 注释
>
> 该字段与 MetadataStatement.attachmentHint 元数据声明相对应。

tcDisplay 类型为 unsigned short

16 位的标记集，当匹配应该被交易确认显示可用性和类型限制时需要设置（参考[UAFRegistry]）。

如果至少一位标记与 AuthenticatorInfo.tcDisplay 值相匹配，则匹配成功[UAFASM]。

> 注释
>
> 该字段与 MetadataStatement.tcDisplay 元数据声明相对应。参考[UAFAuthnrMetadata]

authenticationAlgorithms 类型为 unsigned short

包含支持鉴证算法 TAG 值的数组，当匹配应该被支持的鉴证算法限制时需要设置 (参考[UAFRegistry], 前缀为 UAF_ALG_SIGN)。

如果数组中至少一条记录与 AuthenticatorInfo.authenticationAlgorithm 相匹配，则匹配成功[UAFASM]。

> 注释
>
> 该字段与 MetadataStatement.authenticationAlgorithm 元数据声明相对应。参考[UAFAuthnrMetadata]。

assertionSchemes 类型为 DOMString

支持断言机制的列表，当匹配应该被支持机制限制时需要设置。参考 UAF 支持的断言方案。

如果数组中至少一条记录与 AuthenticatorInfo.assertionScheme 相匹配，则匹配成功[UAFASM]。

> 注释
>
> 该字段与 MetadataStatement.assertionScheme 元数据声明相关。参考[UAFAuthnrMetadata]。

attestationTypes 类型为 unsigned short 数组

包含优先鉴证的 TAG 值的数组(参考[UAFRegistry]，前缀 UAF_ALG_SIGN)。必须保持条目的顺序，最优先的鉴证类型在最前面。

如果数组中至少一条记录与 AuthenticatorInfo.authenticationAlgorithm 中的一条记录相匹配，则匹配成功[UAFASM]。

> 注释
>
> 该字段与 MetadataStatement.attestationTypes 元数据声明相对应。参考[UAFAuthnrMetadata]。

authenticatorVersion 类型为 unsigned short

包含认证器版本号，当匹配应该被正在使用的认证器版本限制时需要设置。

如果该值*小于或等于* TAG_UAFV1_REG_ASSERTION 或 TAG_UAFV1_AUTH_ASSERTION 中 AuthenticatorVersion 字段的值或其他不同断言方案中的对应值，则匹配成功。

> 注释
>
> 因为 authenticatorVersion 的语义依赖于 AAID，authenticatorVersion 字段应该与 MatchCriteria 中的一个 aaid 结合。
>
> 该字段与 MetadataStatement.authenticatorVersion 元数据声明相对应。参

> 考[UAFAuthnrMetadata]。

exts 类型为 *Extension* 数组

匹配策略的扩展。

3.1.12 Policy 结构

包括一份有效认证器说明和一份无效认证器说明。

WebIDL

```
dictionary Policy{
    required MatchCriteria[][]    accepted;
    MatchCriteria[]               disallowed;
};
```

3.1.12.1 Policy 结构成员

accepted 类型为 required MatchCriteria 二维数组

该字段是一个二维数组，主要描述服务器接受 FIDO 注册或有特定目的的鉴证操作所需的必要认证器特征。

二维数组可以看作集合列表。列表中的元素是可选的 （“或”操作条件）。

集合中所有的元素必须被结合：

第一个数组索引表示“或”操作条件（如上述列表）。当任意认证器满足第一个索引中的 MatchCriteria 条件时，服务器可以接受相应操作。

第二个索引（如上述集合）中的 MatchCriteria 子数组表示多认证器必须通过注册或者鉴证来被服务器接受。

MatchCriteria 数组表示服务器的有序优先级。服务器必须把优先级别高的认证器放在前面，FIDO UAF 客户端应该遵守这些优先级，或者通过以同样的顺序将认证器选项显示给用户，或只按最高优先级的认证器来执行操作。

disallowed 类型为 MatchCriteria 数组

与包含在 disallowed 字段内的 MatchCriteria 匹配的认证器，必须从操作合适性中排除，不管是否与 accepted 列表中的 MatchCriteria 匹配。

3.2 服务器策略的处理规则

本节是标准化的。

当解析服务器策略时，FIDO UAF 客户端必须遵守下面的规则：

1. 注册过程：
    1. Policy.accepted 是一个组合列表。每个组合表明服务器希望用户注册的认证器的标准列表。
    2. 遵守 Policy.accepted[][]中条目的优先性。这个列表按最高优先级排序。
    3. 选择与现有有效认证器特征最匹配的组合标准。
    4. 收集有效认证器的信息。
    5. 忽略与 Policy.disallowed 标准匹配的认证器。
    6. 将收集的信息与策略中的匹配标准相匹配。（参考 MatchCriteria 结构获取更多匹配细节）
    7. 引导用户使用组合中指定的认证器进行注册。
2. 鉴别和交易确认过程：

    > **注释**
    >
    > Policy.accepted 是组合列表。每个组合表示一个标准集，每个标准集足够完整地鉴别当前挂起的操作。

    1. 遵守 Policy.accepted[][]中的条目的优先级。列表按照最高优先级排序。
    2. 选择标准现有有效认证器特征最匹配的组合。
    3. 收集有效认证器的信息。
    4. 忽略满足 Policy.disallowed 标准的认证器。
    5. 将收集到的信息策略中描述的匹配标准相匹配。
    6. 引导用户使用组合中指定的认证器进行鉴别。
    7. 只有当单一组合的所有标准都被完全满足时，服务器才会批准一个挂起的操作。

3.2.1 实例

本节是非规范性的。

例 1：与指纹识别或基于面部识别的认证器匹配的策略

```
{
    "accepted":
    [
        [{"userVerification":2}],
        [{"userVerification":16}]
    ]
}
```

例 2：匹配实施指纹识别和面部识别作为用户验证方法的可选组合的认证器策略

```
{
  "accepted":
  [
      [{ "userVerification": 18}]
  ]
}
```

组合这些 2 位标记和 USER_VERIFY_ALL (USER_VERIFY_ALL = 1024) 标记生成单独的 userVerification 值，此值会与实施指纹识别和面部识别的认证器相匹配作为用户验证方法的*强制*组合。

例 3: 匹配实施指纹识别和面部识别作为用户验证方法的强制组合的认证器策略

```
{
    "accepted": [ [{ "userVerification": 1042}] ]
}
```

下一个例子需要使用 2 个认证器：

例 4: 匹配基于指纹识别和面部识别认证器的组合的策略

```
{
  "accepted":
  [
    [
```

```
            { "userVerification": 2},
            { "userVerification": 16}
         ]
       ]
    }
```

除了 userVerification 外可以指定其他条件。

例 5:需要基于绑定的指纹识别和基于绑定的面部识别的认证器的组合策略

```
    {
      "accepted":
      [
            [
               { "userVerification": 2, "attachmentHint": 1},
               { "userVerification": 16, "attachmentHint": 1}
             ]
      ]
    }
```

只接受供应商 ID 为 1234 认证器的策略如下：

例 6:接受供应商 ID 为 1234 的认证器的策略

```
    {
      "accepted":
      [ [{ "vendorID": "1234"}] ]
    }
```

3.3 版本协商

UAF 协议包含多版本结构：UAF 协议版本、密钥注册数据版本、签名数据对象（由各自的标签识别，参考[UAFRegistry]）版本和 ASM 版本（参考[UAFASM]）。

注释

FIDO 服务器必须对密钥注册数据和已签名的数据对象进行分析和验证。只

有当 FIDO 服务器理解被验证者的编码和内容时才可能验证。每个 UAF 协议版本支持一套密钥注册数据和签名数据对象版本（称作断言方案）。同样的，每个 ASM 版本支持一套断言方案版本。

因此，FIDO UAF 客户端必须选择能够产生适当版本结构的认证器。

对于版本协商，FIDO UAF 客户端必须执行如下步骤：

1. 创建版本对集合(FC_Version_Set) ，包括 ASM 版本(asm_version) 和 UAF 协议版本 (upv)，增加所有的 FIDO UAF 客户端支持的版本对到 FC_Version_Set。
    - 例如 [{upv1, asm_version1}, {upv2, asm_version1}, ...]。
2. 把 FC_Version_Set 与包含在 UAF 消息中的 upv 集合相交（例如，只保留那些在 UAF 消息中包含其 upv 值的版本对）。
3. 选择 UAF 消息策略允许的认证器，对每个认证器：
    - 创建一组含有认证器支持的 asm_version 和兼容的 upv(s) 的认证器版本对 (Authnr_Version_Set) 。
        - 例如 [{upv1, asm_version1}, {upv2, asm_version1}, ...]。
    - 把 Authnr_Version_Set 与 FC_Version_Set 相交，从中选择版本最高的版本对。
        - 选取 upv 值最高的版本对。在所有版本对中，只保留具有最高 asm_version 值的。
    - 使用该版本对所对应的认证器。

注释

每个版本含有 major 和 minor 字段。比较两个版本时，先比较 Major 字段，如果相同再比较 Minor 字段。

每个 UAF 消息包含 upv 版本字段。UAF 协议版本协商是在 FIDO UAF 客户端和 FIDO 服务器之间进行。

3.4 注册操作

注释

注册操作允许 FIDO 服务器和 FIDO 认证器商定一个认证密钥。

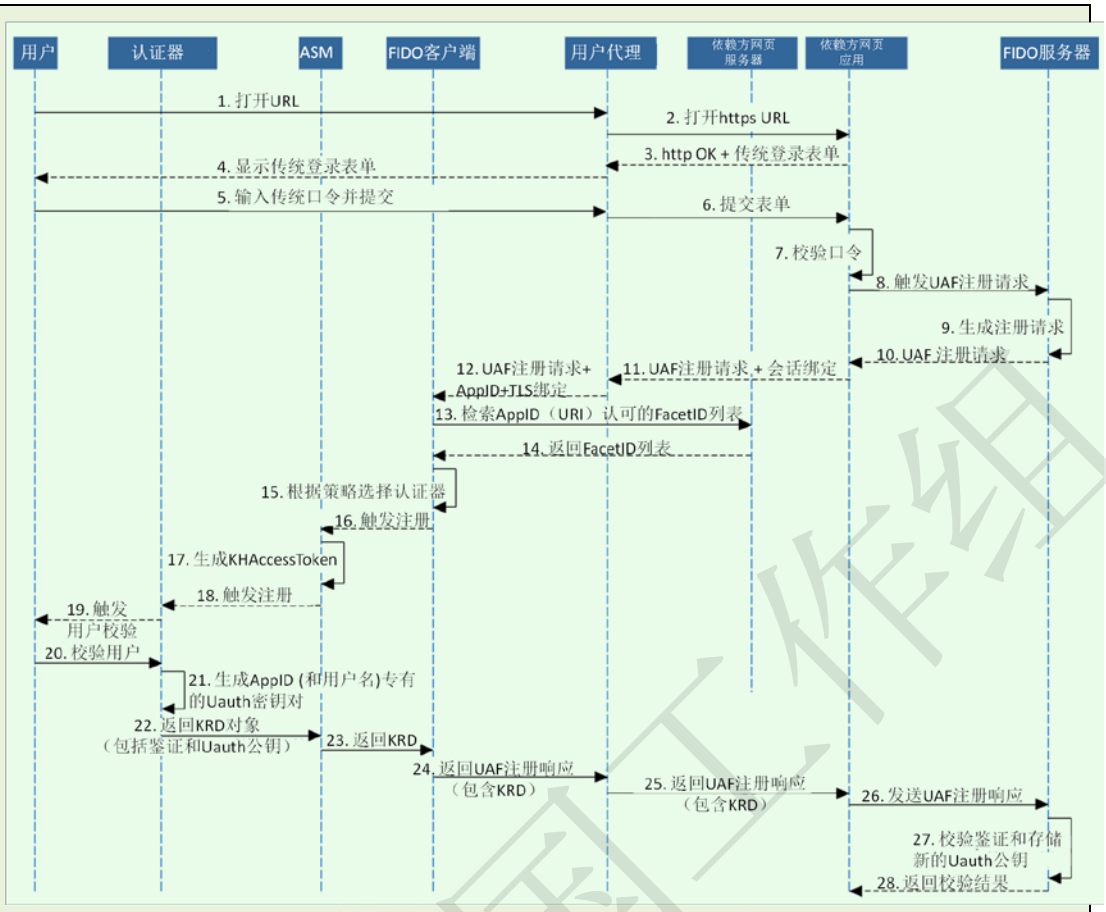

图 6 UAF 注册序列图

下图描绘了注册顺序的加密数据流。

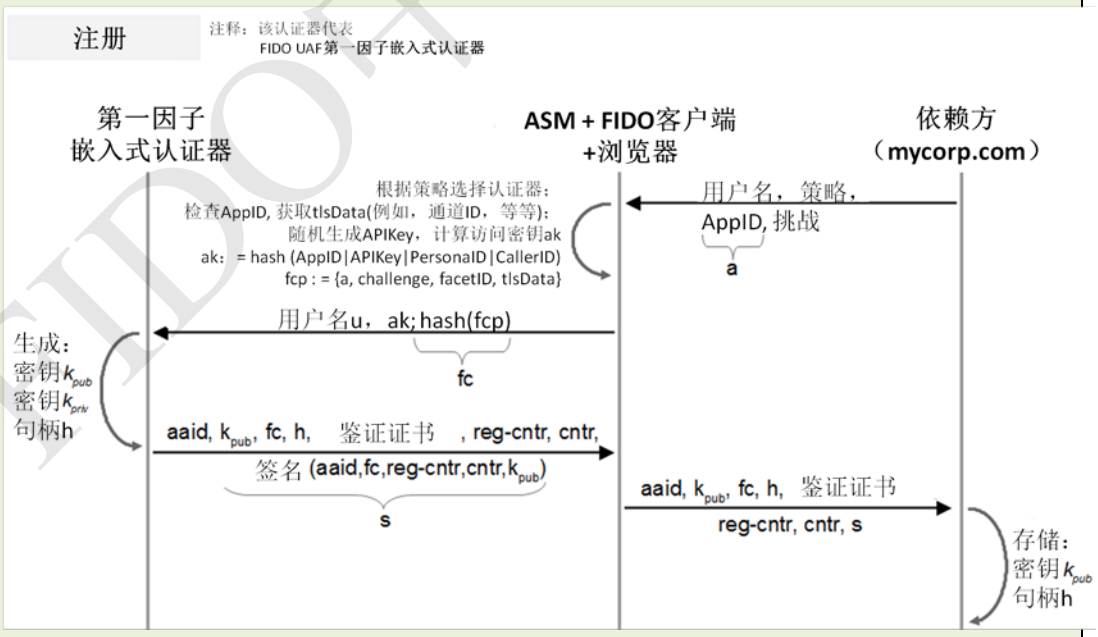

图 7 注册加密数据流

FIDO 服务器发送 AppID、认证器策略、ServerChallenge 和 Username 给

FIDO UAF 客户端。

FIDO UAF 客户端根据 ServerChallenge 和其他值计算 FinalChallengeParams (FCH)，发送 AppID、FCH 和 Username 给认证器。

认证器创建一个密钥注册数据对象（如，TAG_UAFV1_KRD，参考 [UAFAuthnrCommands]），其中包含 FCH 哈希值、新产生的用户公钥 (UAuth.pub)和其他值并对它进行签名（参考认证器鉴证获取更多信息）。随后 FIDO 服务器对 KRD 对象进行加密验证。

3.4.1 注册请求消息

UAF 注册请求消息以结构数组表示。每个结构含有特定协议版本的注册请求。数组中不得包含协议版本相同的两个结构。对于版本“1.0”来说，RegistrationRequest 结构定义了该请求。

例 7：UAF 注册请求

```
[{
      "header": {
        "upv": {
          "major": 1,
          "minor": 0
        },
        "op": "Reg",
  "appID": "https://uaf-test-1.noknoktest.com:8443/SampleApp/uaf/facets",
        "serverData":
"IjycjPZYiWMaQ1tKLrJROiXQHmYG0tSSYGjP5mgjsDaM17RQgq0
dl3NNDDTx9d-aSR_6hGgclrU2F2Yj-
12S67v5VmQHj4eWVseLulHdpk2v_hHtKSvv_DFqL4n
2IiUY6XZWVbOnvg"
      },
      "challenge":
"H9iW9yA9aAXF_lelQoi_DhUk514Ad8Tqv0zCnCqKDpo",
      "username": "apa",
      "policy": {
        "accepted": [
          [
            {
              "userVerification": 512,
              "keyProtection": 1,
              "tcDisplay": 1,
```

```
            "authenticationAlgorithms": [
              1
            ],
            "assertionSchemes": [
              "UAFV1TLV"
            ]
          }
        ],
        [
          {
            "userVerification": 4,
            "keyProtection": 1,
            "tcDisplay": 1,
            "authenticationAlgorithms": [
              1
            ],
            "assertionSchemes": [
              "UAFV1TLV"
            ]
          }
        ],
        [
          {
            "userVerification": 4,
            "keyProtection": 1,
            "tcDisplay": 1,
            "authenticationAlgorithms": [
              2
            ]
          }
        ],
        [
          {
            "userVerification": 2,
            "keyProtection": 4,
            "tcDisplay": 1,
            "authenticationAlgorithms": [
              2
            ]
          }
        ],
        [
          {
            "userVerification": 4,
```

```
            "keyProtection": 2,
            "tcDisplay": 1,
            "authenticationAlgorithms": [
               1,
               3
            ]
         }
      ],
      [
         {
            "userVerification": 2,
            "keyProtection": 2,
            "authenticationAlgorithms": [
               2
            ]
         }
      ],
      [
         {
            "userVerification": 32,
            "keyProtection": 2,
            "assertionSchemes": [
               "UAFV1TLV"
            ]
         },
         {
            "userVerification": 2,
            "authenticationAlgorithms": [
               1,
               3
            ],
            "assertionSchemes": [
               "UAFV1TLV"
            ]
         },
         {
            "userVerification": 2,
            "authenticationAlgorithms": [
               1,
               3
            ],
            "assertionSchemes": [
               "UAFV1TLV"
            ]
```

```
            },
            {
               "userVerification": 4,
               "keyProtection": 1,
               "authenticationAlgorithms": [
                  1,
                  3
               ],
               "assertionSchemes": [
                  "UAFV1TLV"
               ]
            }
         ]
      ],
      "disallowed": [
         {
            "userVerification": 512,
            "keyProtection": 16,
            "assertionSchemes": [
               "UAFV1TLV"
            ]
         },
         {
            "userVerification": 256,
            "keyProtection": 16
         },
         {
            "aaid": [
               "ABCD#ABCD"
            ],
            "keyIDs": [
               "RfY_RDhsf4z5PCOhnZExMeVloZZmK0hxaSi10tkY_c4"
            ]
         }
      ]
   }
}]
```

3.4.2 RegistrationRequest 结构

RegistrationRequest 包含单个的、版本化的注册请求。

```
WebIDL
    dictionary RegistrationRequest{
        required OperationHeader    header;
        required ServerChallenge    challenge;
        required DOMString          username;
        required Policy             policy;
    };
```

3.4.2.1 RegistrationRequest 结构成员

header 类型为 required OperationHeader

操作报头，Header.op 必须是“Reg”。

challenge 类型为 required ServerChallenge

服务器提供的挑战值。

username 类型为 required DOMString

string[1..128]。

可读的用户名是为了方便用户区分和选择相同依赖方的不同账户。

policy 类型为 required Policy

描述哪些认证器类型可以进行注册操作。

3.4.3 AuthenticatorRegistrationAssertion 结构

包含认证器对于 RegistrationRequest 消息的响应：

```
WebIDL
    dictionary AuthenticatorRegistrationAssertion{
        required DOMString                          assertionScheme;
        required DOMString                          assertion;
        DisplayPNGCharacteristicsDescriptor[]       toDisplayPNGCharacteristics;
        Extension[]                                 exts;
    };
```

3.4.3.1 AuthenticatorRegistrationAssertion 结构成员

assertionScheme 类型为 required DOMString

用于编码 assertion 的断言方案名称。参考 UAF 支持的断言方案。

> **注释**
>
> 该断言方案并不是签名对象的一部分，因此被认为是*疑似的*断言方案。

assertion 类型为 required DOMString

base64url(byte[1..4096])包含 TAG_UAFV1_REG_ASSERTION 对象，TAG_UAFV1_REG_ASSERTION 对象包含有特定 KeyRegistrationData (KRD)对象的断言方案，KRD 对象又包含新产生的 UAuth.pub 并会被鉴证私钥签名。

该断言必须由认证器产生并且只能用于注册操作。断言的格式根据断言方案的不同而不同（例如 "UAFV1TLV" 断言方案必须是 TAG_UAFV1_KRD）。

tcDisplayPNGCharacteristics 类型为 DisplayPNGCharacteristicsDescriptor 数组

所支持的交易 PNG 类型[UAFAuthnrMetadata]。

DisplayPNGCharacteristicsDescriptor 结构的定义参考[UAFAuthnrMetadata]。

exts 类型为 *Extension* 数组

含有认证器准备的扩展。

3.4.4 注册响应消息

UAF 注册响应消息以结构数组表示。每个结构含有特定协议版本的注册响应。数组中不得包含协议版本相同的两个结构。对于版本“1.0”来说，RegistrationResponse 结构定义了该响应。

例 8：注册响应

```
[{
    "assertions": [
      {
        "assertion": "AT7uAgM-
sQALLgkAQUJDRCNBQkNEDi4HAAABAQEAAAEKLiAA9tBzZC64ecgVQB
GSQb5QtEIPC8-Vav4HsHLZDflLaugJLiAAZMCPn92yHv1Ip-
iCiBb6i4ADq6ZOv569KFQCvYSJfNgNLggAAQAAAAEAAAAMLkEABJsvEt
UsVKh7tmYHhJ2FBm3kHU-
OCdWiUYVijgYa81MfkjQ1z6UiHbKP9_nRzIN9anprHqDGcR6q7O20q_yctZAH
PjUCBi5AACv8L7YlRMx10gPnszGO6rLFqZFmmRkhtV0TIWuWqYxd1jO0wxa
m7i5qdEa19u4sfpHFZ9RGI_WHxINkH8FfvAwFLu0BMIIB6TCCAY8CAQEwC
QYHKoZIzj0EATB7MQswCQYDVQQGEwJVUzELMAkGA1UECAwCQ0ExC
```

zAJBgNVBAcMAlBBMRAwDgYDVQQKDAdOTkwsSW5jMQ0wCwYDVQQL
DAREQU4xMRMwEQYDVQQDDApOTkwsSW5jIENBMRwwGgYJKoZIhvcN
AQkBFg1ubmxAZ21haWwuY29tMB4XDTE0MDgyODIxMzU0MFoXDTE3MD
UyNDIxMzU0MFowgYYxCzAJBgNVBAYTAlVTMQswCQYDVQQIDAJDQT
EWMBQGA1UEBwwNU2FuIEZyYW5jaXNjbzEQMA4GA1UECgwHTk5MLEl
uYzENMAsGA1UECwwEREFOMTETMBEGA1UEAwwKTk5MLEluYyBDQT
EcMBoGCSqGSIb3DQEJARYNbm5sQGdtYWlsLmNvbTBZMBMGByqGSM49
AgEGCCqGSM49AwEHA0IABCGBt3CIjnDowzSiF68C2aErYXnDUsWXOYxq
IP
im0OWg9FFdUYCa6AgKjn1R99Ek2d803sGKROivnavmdVH-
SnEwCQYHKoZIzj0EAQNJADBGAiEAzAQujXnSS9AIAh6lGz6ydypLVTsTnB
zqGJ4ypIqy_qUCIQCFsuOEGcRV-
o4GHPBph_VMrG3NpYh2GKPjsAim_cSNmQ",
            "assertionScheme": "UAFV1TLV"
          }
        ],
        "fcParams":
"eyJhcHBJRCI6Imh0dHBzOi8vdWFmLXRlc3QtMS5ub2tub2t0ZXN0LmNvbTo4
NDQzL1NhbXBsZUFwcC91YWYvZmFjZXRzIiwiY2hhbGxlbmdlIjoiSDlpVzl5
QTlhQVhGX2xlbFFvaV9EaFVrNTE0QWQ4VHF2MHpDbkNxS0RwbyIsImNoY
W5uZWxCaW5kaW5nIjp7fSwiZmFjZXRJRCI6ImNvbS5ub2tub2suYW5kcm9pZ
C5zYW1wbGVhcHAifQ",
        "header": {
  "appID": "https://uaf-test-1.noknoktest.com:8443/SampleApp/uaf/facets",
          "op": "Reg",
          "serverData":
"IjycjPZYiWMaQ1tKLrJROiXQHmYG0tSSYGjP5mgjsDaM17RQgq0
dl3NNDDTx9d-aSR_6hGgclrU2F2Yj-
12S67v5VmQHj4eWVseLulHdpk2v_hHtKSvv_DFqL4n
2IiUY6XZWVbOnvg",
          "upv": {
            "major": 1,
            "minor": 0
          }
        }
}]

> **注释**
>
> fcParams 中的分行显示主要是为了提高可读性。

3.4.5 RegistrationResponse 结构

包括与注册响应相关的所有字段。

```
WebIDL
dictionary RegistrationResponse {
    required OperationHeader                        header;
    required DOMString                              fcParams;
    required AuthenticatorRegistrationAssertion[]   assertions;
};
```

3.4.5.1 RegistrationResponse 结构成员

header 类型为 required OperationHeader

Header.op 必须是"Reg"。

fcParams 类型为 required DOMString

fcParams 字段是使用 UTF8 编码 FinalChallengeParams（参考 FinalChallengeParams 结构），将结果序列化[RFC4627]后再使用 base64url 进行编码。fcParams 字段中含有服务器用来验证 Final Challenge 需要的所有参数。

assertions 类型为 required AuthenticatorRegistrationAssertion 数组

每个被注册的认证器的响应数据。

3.4.6 注册处理规则

3.4.6.1 FIDO 服务器注册请求生成规则

该策略含有允许 MatchCriteria 的二维数组（参考策略）。该数组可以看作是认证器（由 MatchCriteria 识别）集合（二维）的列表（一维）。在特定集合中的所有认证器必须同时被注册以匹配策略。但是列表中的集合都是有效的，列表中的元素是可选的。

FIDO 服务器必须遵循下列步骤：

1. 构建适当的注册策略 p
    1. 对每个备选认证器集合，
        1. 创建 MatchCriteria 对象数组 v，它包含需要同时注册的认证器集合，该集合需要由*单独的* MatchCriteria 对象 m 识别。

1. 对于每个能够被*相同规则*识别的用于同时注册的认证器集合 a，创建一个 MatchCriteria 对象 m，当：
    - m.aaid 可能与 m.keyIDs、m.attachmentHint、m.authenticatorVersion、m.exts（其中一个或多个）结合，但是不得与其他 MatchCriteria 字段结合。
    - 如果未提供 m.aaid，必须至少提供 m.authenticationAlgorithms 和 m.assertionSchemes。
    2. 添加 m 到 v，例如 v[j+1]=m。
2. 添加 v 到 p.allowed 中，例如 p.allowed[i+1]=v。

2. 对所有无效认证器，创建 MatchCriteria 对象 m[]。
    1. 对现有用户每个已注册的 AAID
        1. 创建 MatchCriteria 对象 m，增加 AAID 和相应的 KeyIDs 到 m.aaid 和 m.KeyIDs。

           FIDO 服务器必须添加已注册的 AAIDs 和 KeyIDs 到 p.disallowed 字段来提示客户端不应该再注册相同信息。
    2. 创建 MatchCriteria 对象 m，将所有无效认证器的 AAID 添加到 m.aaid。

       一些认证器的状态(例如由元数据 TOC（Table-of-Content 内容列表文件） [UAFMetadataService]提供的)可能不被接受，这样的认证器应该添加到 p.disallowed 中。
    3. 如果需要，为其他不允许的条件（例如不支持的鉴别算法）创建 MatchCriteria m。
    4. 增加所有 m 到 p.disallowed。
2. 对每个支持的版本，创建带有合适的 r.header 的 RegistrationRequest 对象 r，
    1. FIDO 服务器不得假定 r.header.serverData 暗含任何完整性保护。

       依赖于 r.header.serverData 的完整性的 FIDO 服务器应该应用并校验给 serverData 的加密的安全消息鉴别码（MAC），也应该将 serverData 加密绑定到相关消息，例如重新包含 r.challenge，参考服务器数据和秘钥

句柄。

> **注释**
>
> 所有其他的 FIDO 组件（除了 FIDO 服务器）将把 r.header.serverData 看作非透明值，因此 FIDO 服务器可以实施任何适合的加密保护方法。

2. 产生一个随机的挑战并分配给 r.challenge。
3. 分配待注册用户的用户名给 r.username。
4. 分配 p 到 r.policy。
5. 添加 r 到有不同版本(RegistrationRequest)消息的数组 o 。

3. 发送 o 到 FIDO UAF 客户端。

3.4.6.2 FIDO UAF 客户端注册请求处理规则

FIDO UAF 客户端必须遵循下列步骤：

1. 选择主要版本为 1、次要版本为 0 的消息 m。
2. 解析消息 m。
3. 如果 UAF 消息中的必填字段为空或者某字段的类型和值不符，则拒绝该操作。
4. 用给定的策略筛选现有的认证器，并将已筛选的认证器呈现给用户。确保不包含 RegRequest.policy.disallowed[].keyIDs 中该用户已注册的认证器。
5. 获取请求应用的 FacetID。如果 AppID 缺失或为空，用 FacetID 作为 AppID。

   根据[FIDOAppIDAndFacets]中的算法，校验 FacetID 对于此 AppID 是被授权的。

   - 如果请求应用的 FacetID 没有被授权，拒绝该操作。
6. 如果 TLS 数据可用则获取该数据。
7. 创建 FinalChallengeParams 结构 fcp，并正确设定 fcp.AppID，fcp.challenge，fcp.facetID 和 fcp.channelBinding。使用 UTF8 编码 fcp，将结果序列化[RFC4627]后再使用 base64url 进行编码。

   - FinalChallenge = base64url(serialize(utf8encode(fcp)))

8. 对每个匹配 UAF 协议版本（参考版本协商部分）并且用户同意注册的认证器：
    1. 添加 AppID，Username，FinalChallenge，AttestationType 和所有其他必需的字段到 ASMRequest[UAFASM]。

       FIDO UAF 客户端必须遵守服务器策略并且找出最优的鉴别类型。单独的鉴证类型必须提供给 ASM。
    2. 发送 ASMRequest 给 ASM。

3.4.6.3 FIDO 认证器注册请求处理规则

参考[UAFAuthnrCommands]，“注册命令”章节。

3.4.6.4 FIDO UAF 客户端注册响应生成规则

FIDO UAF 客户端必须遵守下列步骤：

1. 创建 RegistrationResponse 消息。
2. 复制 RegistrationRequest.header 到 RegistrationResponse.header。
3. 设置 RegistrationResponse.fcParams 到 FinalChallenge (使用 UTF8 编码 FinalChallengeParoa，将结果序列化后再使用 base64url 进行编码)。
4. 将来自每个认证器的响应附加到 RegistrationResponse.assertions。
5. 向 FIDO 服务器发送 RegistrationResponse 消息。

3.4.6.5 FIDO 服务器注册响应处理规则

> 注释
>
> 以下处理规则假定认证器支持"UAFV1TLV"断言方案。目前 "UAFV1TLV" 是唯一被定义和支持的断言方案。当新的断言方案添加到 UAF 协议中时，本节将会扩展相应的处理规则。

FIDO 服务器必须遵守下面的步骤：

1. 解析消息。
    1. 如果不支持该协议版本（RegistrationResponse.header.upv），则拒绝该操作。

2. 如果 UAF 消息中的必填字段为空或者某字段的类型和值不符，则拒绝该操作。

2. 如果使用了 RegistrationResponse.header.serverData，校验其通过了任何具体实施的有效性检查，参考服务器数据和密钥句柄。
3. Base64url 解码 RegistrationResponse.fcParams 并转化为对象(fcp)。
4. 校验 fcp 中的每个字段确保其有效：
    1. 确保 fcp.appID 与 FIDO 服务器保存的一致。
    2. 确保 fcp.challenge 的确是由 FIDO 服务器为此操作生成并且没有过期。
    3. 确保 fcp.facetID 在可信 FacetIDs 列表中[FIDOAppIDAndFacets]。
    4. 确保 fcp.channelBinding 符合预期（参考 ChannelBinding 结构）。
    5. 如果以上任意检查失败，则拒绝该响应。
5. 对 RegistrationResponse.assertions 中的每个断言 a：
    1. 解析 a.assertion 中的 TLV 数据，假定其按照疑似的断言方案 a.assertionScheme 进行了编码，并确保它含有所有的必填字段（认证器元数据中指明的）并且使用了正确的语法。
        - 如果不符合上述要求，则继续验证下一个断言。
    2. 从断言中提取 AAID。

        > **注释**
        >
        > TAG_UAFV1_KRD 中的 AAID 包含在 a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.TAG_AAID 中。

    3. 验证 a.assertionScheme 是否与 Metadata(AAID).assertionScheme 匹配。
        - 如果不匹配，则继续下一个断言。
    4. 验证 AAID 是否与注册请求的策略相匹配。

        > **注释**
        >
        > 根据策略（如 AND 组合的情况），可能要求评估 RegistrationResponse 中的其他断言来决定 AAID 是否与该策略匹配。

        - 如果不匹配该策略，则继续下一个断言。

5. 使用 AAID 从认证器元数据[UAFAuthnrMetadata]中查找认证器特定的鉴别算法。
6. 用适用于该认证器类型的哈希算法对 RegistrationResponse.fcParams 哈希。在认证器元数据中的 AuthenticationAlgs 字段查找哈希算法，该哈希算法与带有 UAF_ALG_SIGN 前缀常量的第一条相关。
    - FCHash = hash(RegistrationResponse.fcParams)。
7. 如果 a.assertion 含有类型为 TAG_UAFV1_REG_ASSERTION 的对象，那么：
    1. 如果 a.assertion.TAG_UAFV1_REG_ASSERTION 含有 TAG_UAFV1_KRD 作为第一元素：
        1. 为此 AAID 获取 Metadata(AAID).AttestationType，确保 a.assertion.TAG_UAFV1_REG_ASSERTION 中含有 RegistrationRequest.policy 中的 MatchCriteria.attestationTypes 字段（如果该字段存在）中的最优鉴证标签。
            - 如果 a.assertion.TAG_UAFV1_REG_ASSERTION 未包含最优鉴证，则建议跳过这个断言继续下一个。
        2. 确保 a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.FinalChallenge == FCHash。
            - 如果比较失败，则继续下一个断言。
        3. 为此 AAID 获取 Metadata(AAID).AuthenticatorVersion，确保它小于等于 a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.AuthenticatorVersion 的值。
            - 如果 Metadata(AAID).AuthenticatorVersion 更高（例如认证器固件过期），则建议假定风险会增加。参考[UAFMetadataService] 中的"StatusReport 结构" 和"元数据 TOC 对象处理规则"章节。
        4. 检查是否接受 a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.

RegCounter，例如不支持（值为 0）或者不是非常高。

- 如果 a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.RegCounter 非常高，该断言可能会跳过并继续处理下一个。

5. 如果 a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD 含有 TAG_ATTESTATION_BASIC_FULL 标签：
    1. 如果 AAID 在元数据[UAFAuthnrMetadata]中对应的 AttestationRootCertificates 条目含有至少一个元素：
        1. 从 a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_ATTESTATION_BASIC_FULL 对象处获取所有 TAG_ATTESTATION_CERT 标签的内容，该事件是有序的（参考[UAFAuthnrCommands]）并且代表相关证书链后的鉴证证书。
        2. 从认证器元数据中获取对应 AAID 的 AttestationRootCertificates 字段的所有条目。
        3. 使用[RFC5280]中的证书路径有效性来校验鉴证证书及到鉴证根证书的整个证书链。
            - 如果验证失败，则继续下一断言。
        4. 使用鉴证证书（之前获取到的）校验 a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.TAG_ATTESTATION_BASIC_FULL.Signature
            - 如果验证失败，则继续下一断言。
    2. 如果此 AAID 的 Metadata(AAID).AttestationRootCertificates 为空，则继续下一断言。
    3. 标记断言为验证成功。

6. 如果 a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD 含有 TAG_ATTESTATION_BASIC_SURROGATE 类型的对象：
    1. 没有对 AAID 真实的鉴证，所以我们只假定 AAID 是真实的。
    2. 如果元数据中 AAID 的 AttestationRootCertificates 条目是空的，
        - 使用 a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.TAG_PUB_KEY 来校验 a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_ATTESTATION_BASIC_SURROGATE.Signature。
            - 如果校验失败，继续下一个断言。
    3. 如果元数据中 AAID 的 AttestationRootCertificates 条目不为空，则继续下一断言（AAID 显然希望另外一种不同的鉴证方法）。
    4. 标记断言为验证成功。
7. 如果 a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD 含有另外一个 TAG_ATTESTATION 标签，按照适用于该鉴证的处理规则验证鉴证。目前文档只定义了基础鉴证的处理规则。

2. 如果 a.assertion.TAG_UAFV1_REG_ASSERTION 含有除了 TAG_UAFV1_KRD 以外的对象作为第一元素，则遵循该对象的特定规则。
3. 提取 a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.PublicKey 到 PublicKey，a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.KeyID 到 KeyID，a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.Sign

Counter 到 SignCounter，

a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.TAG_ASSERTION_INFO.authenticatorVersion 到 AuthenticatorVersion，

a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.TAG_AAID 到 AAID。

8. 如果 a.assertion 不含有 TAG_UAFV1_REG_ASSERTION 类型的对象，则跳过这个断言（因为 UAF v1 中只定义了 TAG_UAFV1_REG_ASSERTION）。

6. 对每个成功经过校验的断言 a，
   - 存储 PublicKey、KeyID、SignCounter、AuthenticatorVersion、AAID 和 a.tcDisplayPNGCharacteristics 到与用户身份相关的记录中。如果带有相同的 AAID 和 KeyID 对的条目已经存在则校验失败（永远不应该发生）。

3.5 鉴别操作

注释

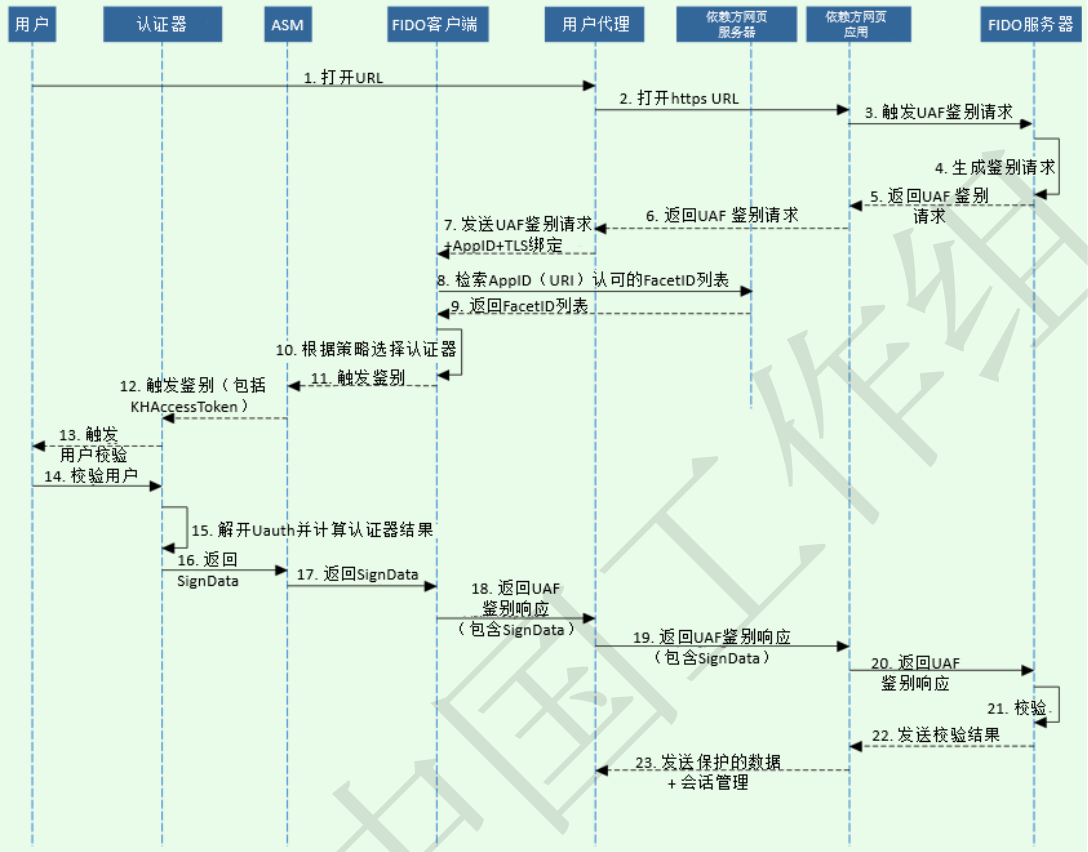

图 8 鉴别顺序图

在这个操作中，FIDO 服务器要求 FIDO UAF 客户端用服务端指定认证器鉴别用户，并返回鉴别响应。

为了使此操作成功，该认证器和依赖方必须进行预共享注册。

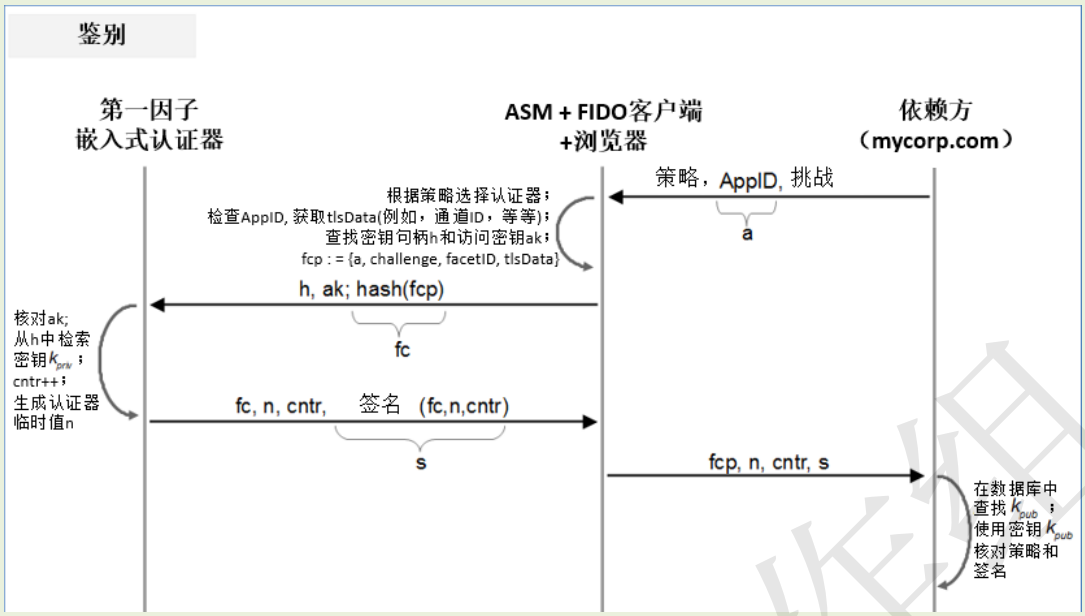

图 9 鉴别加密数据流

加密流图：

FIDO 服务器发送 AppID（参考 [FIDOAppIDAndFacets]）、认证器策略和 ServerChallenge 给 FIDO UAF 客户端。

FIDO UAF 客户端计算由 ServerChallenge 和其他值生成的 FinalChallengeParams 的哈希值，并发送 AppID 和经过哈希的 FinalChallengeParams 给认证器。

认证器创建含有最终挑战参数的哈希值和其他值的 SignedData 对象（参考 [UAFAuthnrCommands]中的 TAG_UAFV1_SIGNED_DATA），并用 UAuth.priv 私钥对其签名。这个断言随后被 FIDO 服务器进行了加密校验。

3.5.1 Transaction 结构

含有 FIDO 服务器提供的交易内容。

WebIDL

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

3.5.1.1 Transaction 结构成员

contentType 类型为 required DOMString

包括认证器元数据声明（参考[UAFAuthnrMetadata]）中支持的 MIME 内容类型。

本协议版本只支持 text/plain 或 image/png 值。

content 类型为 required DOMString

base64url(byte[1...])。

包含 base64url 编码的交易内容，根据 contentType 类型展示给用户。

如果 contentType 是"text/plain"，则内容必须是至多 200 个字符的经过 base64url 编码的 ASCII 码文本。

tcDisplayPNGCharacteristics 类型为 DisplayPNGCharacteristicsDescriptor

交易内容 PNG 类型。对于 DisplayPNGCharacteristicsDescriptor 结构的定义参考[UAFAuthnrMetadata]。如果 contentType 是"image/png"，该字段必须存在。

3.5.2 鉴别请求消息

UAF 鉴别请求消息以结构数组表示。每个结构含有特定协议版本的鉴别请求。数组中不得包含协议版本相同的两个结构。对版本 1.0 来说，AuthenticationRequest 结构定义了该请求。

例 9：UAF 鉴别请求

```
[{
      "header": {
        "upv": {
          "major": 1,
          "minor": 0
        },
        "op": "Auth",
  "appID": "https://uaf-test-1.noknoktest.com:8443/SampleApp/uaf/facets",
        "serverData": "5s7n8-
7_LDAtRIKKYqbAtTTOezVKCjl2mPorYzbpxRrZ-_3wWro
MXsF_pLYjNVm_l7bplAx4bkEwK6ibil9EHGfdfKOQ1q0tyEkNJFOgqdjV
mLioroxgThlj8Is
```

```
tpt7q"
      },
      "challenge":
"HQ1VkTUQC1NJDOo6OOWdxewrb9i5WthjfKIehFxpeuU",
      "policy": {
        "accepted": [
          [
            {
              "userVerification": 512,
              "keyProtection": 1,
              "tcDisplay": 1,
              "authenticationAlgorithms": [
                1
              ],
              "assertionSchemes": [
                "UAFV1TLV"
              ]
            }
          ],
          [
            {
              "userVerification": 4,
              "keyProtection": 1,
              "tcDisplay": 1,
              "authenticationAlgorithms": [
                1
              ],
              "assertionSchemes": [
                "UAFV1TLV"
              ]
            }
          ],
          [
            {
              "userVerification": 4,
              "keyProtection": 1,
              "tcDisplay": 1,
              "authenticationAlgorithms": [
                2
              ]
            }
          ],
          [
            {
```

```
            "userVerification": 2,
            "keyProtection": 4,
            "tcDisplay": 1,
            "authenticationAlgorithms": [
               2
            ]
         }
      ],
      [
         {
            "userVerification": 4,
            "keyProtection": 2,
            "tcDisplay": 1,
            "authenticationAlgorithms": [
               1,
               3
            ]
         }
      ],
      [
         {
            "userVerification": 2,
            "keyProtection": 2,
            "authenticationAlgorithms": [
               2
            ]
         }
      ],
      [
         {
            "userVerification": 32,
            "keyProtection": 2,
            "assertionSchemes": [
               "UAFV1TLV"
            ]
         },
         {
            "userVerification": 2,
            "authenticationAlgorithms": [
               1,
               3
            ],
            "assertionSchemes": [
               "UAFV1TLV"
```

```
                    ]
                },
                {
                    "userVerification": 2,
                    "authenticationAlgorithms": [
                        1,
                        3
                    ],
                    "assertionSchemes": [
                        "UAFV1TLV"
                    ]
                },
                {
                    "userVerification": 4,
                    "keyProtection": 1,
                    "authenticationAlgorithms": [
                        1,
                        3
                    ],
                    "assertionSchemes": [
                        "UAFV1TLV"
                    ]
                }
            ]
        ],
        "disallowed": [
            {
                "userVerification": 512,
                "keyProtection": 16,
                "assertionSchemes": [
                    "UAFV1TLV"
                ]
            },
            {
                "userVerification": 256,
                "keyProtection": 16
            }
        ]
    }
}]
```

3.5.3 AuthenticationRequest 结构

包含有 UAF 鉴别请求消息：

WebIDL

```
dictionary AuthenticationRequest {
    required OperationHeader    header;
    required ServerChallenge    challenge;
    Transaction[]               transaction;
    required Policy             policy;
};
```

3.5.3.1 AuthenticationRequest 结构成员

header 类型为 required OperationHeader

Header.op 必须是 "Auth"。

challenge 类型为 required ServerChallenge

服务器提供的挑战值。

transaction 类型为 *Transaction* 数组

用户明确确认的交易数据。

该列表含有不同内容类型和图形大小的相同交易内容。参考 [UAFAuthnrMetadata]获取更多关于交易确认显示特征的信息。

policy 类型为 required Policy

服务器提供的定义哪些认证器类型可以进行鉴别操作的策略。

3.5.4 AuthenticatorSignAssertion 结构

表示特定认证器产生的响应：

WebIDL

```
dictionary AuthenticatorSignAssertion {
    required DOMString    assertionScheme;
    required DOMString    assertion;
    Extension[]           exts;
};
```

3.5.4.1 AuthenticatorSignAssertion 结构成员

assertionScheme 类型为 required DOMString

用于编码 assertion 的断言方案名称。参考 UAF 支持的断言方案。

> **注释**
>
> 该断言方案并不是签名对象的一部分，因此被认为是*疑似的*断言方案。

assertion 类型为 required DOMString

base64url(byte[1..4096]) 包含含有由 UAuth.priv 生成的签名的断言，例如 TAG_UAFV1_AUTH_ASSERTION。

exts 类型为 *Extension* 数组

认证器准备的任意扩展。

3.5.5 AuthenticationResponse 结构

代表对挑战的响应，包括已注册过的认证器的签名断言集。

WebIDL

```
dictionary AuthenticationResponse {
    required OperationHeader                herder;
    required DOMString                      fcParams;
    required AuthenticatorSignAssertion[]   assertion;
};
```

3.5.5.1 AuthenticationResponse 结构成员

header 类型为 required OperationHeader

Header.op 必须是“Auth”。

fcParams 类型为 required DOMString

fcParams 字段是使用 UTF8 编码 FinalChallengeParams（参考 FinalChallengeParams 结构），将结果序列化[RFC4627]后再使用 base64url 进行编码。fcParams 字段中含有服务器来验证 Final Challenge 需要的所有参数。

assertions 类型为 required AuthenticatorSignAssertion 数组

与该操作相关的认证器响应列表。

3.5.6 鉴别响应消息

UAF 鉴别响应消息以结构数组表示。每个结构含有特定协议版本的鉴别响应。数组中不得包含协议版本相同的两个结构。对于版本“1.0”来说，AuthenticationResponse 结构定义了该响应。

例 10：UAF 鉴别响应


```
[{
    "assertions": [
      {
        "assertion": "Aj7WAAQ-
jgALLgkAQUJDRCNBQkNEDi4FAAABAQEADy4gAHwyJAEX8t1b2wO
xbaKOC5ZL7ACqbLo_TtiQfK3DzDsHCi4gAFwCUz-
dOuafXKXJLbkUrIzjAU6oDbP8B9iLQRmCf58fEC4AAAkuIABkwI-
f3bIe_Uin6IKIFvqLgAOrpk6_nr0oVAK9hIl82A0uBAACAAAABi5AADw
DOcBvPslX2bRNy4SvFhAwhEAoBSGUitgMUNChgUSMxss3K3ukekq1pa
G7Fv1v5mBmDCZVPt2NCTnjUxrjTp4",
        "assertionScheme": "UAFV1TLV"
      }
    ],
    "fcParams":
"eyJhcHBJRCI6Imh0dHBzOi8vdWFmLXRlc3QtMS5ub2tub2t0ZXN0LmN
vbTo4NDQzL1NhbXBsZUFwcC91YWYvZmFjZXRzIiwiY2hhbGxlbmdlIjoi
SFExVmtUVVFDMU5KRE9vNk9PV2R4ZXdyYjlpNVd0aGpmS0llaEZ4cG
V1VSIsImNoYW5uZWxCaW5kaW5nIjp7fSwiZmFjZXRJRCI6ImNvbS5ub2
tub2suYW5kcm9pZC5zYW1wbGVhcHAifQ",
    "header": {
  "appID": "https://uaf-test-1.noknoktest.com:8443/SampleApp/uaf/facets",
      "op": "Auth",
      "serverData": "5s7n8-
7_LDAtRIKKYqbAtTTOezVKCjl2mPorYzbpxRrZ-
_3wWroMXsF_pLYjNVm_l7bplAx4bkEwK6ibil9EHGfdfKOQ1q0tyEkNJF
OgqdjVmLioroxgThlj8Istpt7q",
      "upv": {
        "major": 1,
        "minor": 0
      }
    }
}]
```

> 注释
>
> fcParams 中的分行显示只是为了提高可读性。

3.5.7 鉴别处理规则

3.5.7.1 FIDO 服务器鉴别请求生成规则

策略含有一个允许 MatchCriteria 的二维数组（参考策略）。该数组可以以看作是认证器（由 MatchCriteria 识别）集合（二维）的列表（一维）。在特定集合中的所有认证器必须同时被注册以匹配策略。但是列表中的集合都是有效的，列表中的元素是可选的。

FIDO 服务器必须遵循下列步骤：

1. 构建适当的鉴别策略 p。
    1. 对每个备选认证器集合：
        1. 创建 MatchCriteria 对象 v 的一维数组，它包含需要同时鉴别的认证器集合，该集合需要由*单独的* MatchCriteria 对象 m 识别。
            1. 对于每个能够被*相同规则*识别的用于同时鉴别的认证器集合

                a，创建一个 MatchCriteria 对象 m，当：

                - m.aaid 可能与 m.keyIDs、m.attachmentHint、m.authenticatorVersion、m.exts（其中一个或多个）结合，但是不得与其他 MatchCriteria 字段结合。
                - 如果未提供 m.aaid，必须至少提供 m.authenticationAlgorithms 和 m.assertionSchemes。
                - 在递进鉴别的情况下，（例如因为之前的鉴别步骤，已经知道了用户的情况），为了避免在此依赖方有多个账户时的模糊性，Policy.accepted 中的每一项必须包括注册到此账户的认证器的 AAID 和 KeyID。
            2. 将 m 添加到 v 中，例如 v[j+1]=m。
        2. 将 v 添加到 p.allowed 中，例如 p.allowed[i+1]=v。
    2. 对所有无效认证器，创建 MatchCriteria 对象 m[]。

1. 创建 MatchCriteria 对象 m，将所有无效认证器的 AAID 添加到 m.aaid。

   一些认证器的状态（例如由元数据 TOC[UAFMetadataService]提供的）可能不被接受，这样的认证器应该添加到 p.disallowed 中。
2. 如果需要，为其它不允许的条件（例如不支持的鉴别算法）创建 MatchCriteria m。
3. 将所有 m 添加到 p.disallowed。

2. 对每个支持的版本，创建带有合适的 r.header 的 AuthenticationRequest 对象 r：

   1. FIDO 服务器不得承担对 r.header.serverData 的任何暗含的完整性保护。依赖于 r.header.serverData 的完整性的 FIDO 服务器应该申请和校验给 serverData 的加密的安全消息鉴别码（MAC），也应该将 serverData 加密绑定到相关消息，例如重新包含 r.challenge，参考服务器数据和密钥句柄。

      > **注释**
      >
      > 所有其它的 FIDO 组件（除了 FIDO 服务器）都会把 r.header.serverData 看作非透明值。因此 FIDO 服务器可以实现任何合适的加密保护方法。

   2. 产生一个随机挑战值并分配给 r.challenge。
   3. 如果这是交易确认操作，从每个参与的 AAID 的认证器元数据中查找 TransactionConfirmationDisplayContentTypes/ TransactionConfirmationDisplayPNGCharacteristics，生成相应的交易内容列表，并将此列表插入到 r.transaction 中。
      - 如果在注册过程中认证器报告了（动态的）AuthenticatorRegistrationAssertion.tcDisplayPNGCharacteristics，它的优先级必须高于认证器元数据中特定的（静态）值。
   4. 将 r.policy 设置为之前创建的新策略对象 p，例如 r.policy = p。

5. 将鉴别请求消息添加到数组。

3. 将鉴别请求消息数组发送到 FIDO UAF 客户端。

3.5.7.2 FIDO UAF 客户端鉴别请求处理规则

FIDO UAF 客户端必须遵循下列步骤：

1. 选择主要版本为 1，次要版本为 0 的消息 m。
2. 解析消息 m。
    - 如果 UAF 消息中的必填字段为空或者某字段的类型和值不符，则拒绝该操作。
3. 获取请求应用的 FacetID。如果 AppID 缺失或为空，将 AppID 设定为 FacetID 。

   根据[FIDOAppIDAndFacets]中的算法，校验 FacetID 对于此 AppID 是被授权的。
    - 如果请求应用的 FacetID 没有被授权，拒绝该操作。
4. 根据给定策略过滤有效认证器，并将过滤的列表呈现给用户。
    - 如果 AuthenticationRequest.policy.accepted 列表为空，则给用户推荐任意已注册的认证器用于鉴别。
5. 让用户选择其首选的认证器。
6. 如果 TLS 数据可用则获取该数据。
7. 创建 FinalChallengeParams 结构 fcp ，并正确设定 fcp.AppID，fcp.challenge，fcp.facetID 和 fcp.channelBinding 。使用 UTF8 编码 fcp，将结果序列化[RFC4627]后再使用 base64url 进行编码。
    - FinalChallenge = base64url(serialize(utf8encode(fcp)))
8. 对每个支持与消息版本 AuthenticationRequest.header.upv（参考版本协商）兼容的认证器接口版本（AIV）的认证器以及用户同意用于鉴别的认证器：
    1. 将 AppID，FinalChallege，Transactions（如果存在）和所有其它字段添加到 ASMRequest。
    2. 将 ASMRequest 发送到 ASM。

3.5.7.3 FIDO 认证器鉴别请求处理规则

参考[UAFAuthnrCommands]“签名命令”章节。

3.5.7.4 FIDO UAF 客户端鉴别响应生成规则

FIDO UAF 客户端必须遵循下列步骤：

1. 创建 AuthenticationResponse 消息。
2. 将 AuthenticationRequest.header 复制到 AuthenticationResponse.header。
3. 用合适的字段填写 AuthenticationResponse.FinalChallengeParams，然后将其字符串化。
4. 将来自每个认证器的响应附加到 AuthenticationResponse.assertions。
5. 向 FIDO 服务器发送 AuthenticationResponse 消息。

3.5.7.5 FIDO 服务器鉴别响应处理规则

> **注释**
>
> 以下处理规则假定认证器支持"UAFV1TLV"断言方案。目前 "UAFV1TLV" 是唯一被定义和支持的断言方案。当新的断言方案添加到 UAF 协议中时，本节将会扩展相应的处理规则。

FIDO 服务器必须遵循下列步骤：

1. 解析消息。
    1. 如果不支持该协议版本（AuthenticationResponse.header.upv），则拒绝该操作。
    2. 如果 UAF 消息中的必填字段为空或者某字段的类型和值不符，则拒绝该操作。
2. 如果使用了 AuthenticationResponse.header.serverData，校验其通过了任何具体实施的有效性检查。也可参考服务器数据和密钥句柄章节。
3. Base64url 解码 AuthenticationResponse.fcParams 并转变为对象(fcp)。
4. 校验 fcp 中的每个字段确保其有效：
    1. 确保 AppID 与 FIDO 服务器存储的一致。

2. 确保 FacetID 在可信 FacetID 列表中[FIDOAppIDAndFacets]。
3. 确保 ChannelBinding 符合预期（参考 ChannelBinding 结构章节）。
4. 校验客户端提交的 ServerChallenge 是由 FIDO 服务器生成的。
5. 如果以上任一检查失败，则拒绝该响应。

5. 对每个 AuthenticationResponse.assertions 中的断言 a：
   1. 解析 a.assertion 中的 TLV 数据，假定其按照疑似的断言方案 a.assertionScheme 进行了编码，并确保它含有所有的必填字段（认证器元数据中指明的）并且使用了正确的语法。
      - 如果不符合上述要求，继续下一个断言。
   2. 从断言中提取 AAID。

      > 注释
      >
      > TAG_UAFV1_SIGNED_DATA 中的 AAID 包含在 a.assertion.TAG_UAFV1_AUTH_ASSERTION.TAG_UAFV1_SIGNED_DATA.TAG_AAID 中。

   3. 校验 a.assertionScheme 是否与 Metadata(AAID).assertionScheme 匹配，
      - 如果不匹配，则继续下一个断言。
   4. 确保 AAID 与鉴别请求的策略相匹配。
      - 如果不匹配该策略，则继续下一个断言。
   5. 如果 a.assertion 含有类型为 TAG_UAFV1_AUTH_ASSERTION 的对象，那么：
      1. 如果 a.assertion.TAG_UAFV1_AUTH_ASSERTION 含有 TAG_UAFV1_SIGNED_DATA 作为第一元素。
         1. 为此 AAID 获取 Metadata(AAID).AuthenticatorVersion，确保它小于或等于 a.assertion.TAG_UAFV1_AUTH_ASSERTION.TAG_UAFV1_SIGNED_DATA.TAG_ASSERTION_INFO.AuthenticatorVersion 的值。
            - 如果 Metadata(AAID).AuthenticatorVersion 更大（例如认证器固件过期），建议假定鉴别风险增加。参考

[UAFMetadataService]中的“StatusReport 结构”和“元数据 TOC 对象处理规则”章节。

2. 提取 a.assertion.TAG_UAFV1_AUTH_ASSERTION.TAG_UAFV1_SIGNED_DATA.TAG_KEYID 作为 KeyID。
3. 定位与用户记录中（AAID，KeyID）相关的 UAuth.pub 公钥。
    - 如果不存在这样的记录，则继续下一个断言。
4. 对比校验 AAID 与注册时存储在用户记录中的 AAID。
    - 如果比较失败，则继续下一个断言。
5. 从认证器元数据中查找认证器特定鉴别算法（AuthenticationAlgs 字段）。
6. 检查签名计数器 a.assertion.TAG_UAFV1_AUTH_ASSERTION.TAG_UAFV1_SIGNED_DATA.SignCounter，确保该计数器要么不被认证器支持（例如该值和用户记录中的值都是 0），要么是递增的（与存储在用户记录中的值相比）。
    - 如果大于 0 但不递增，则继续下一个断言（这是一个克隆的认证器或之前使用过的克隆认证器）。
7. 用适用于该认证器类型的哈希算法对 AuthenticationResponse.FinalChallengeParams 哈希。在认证器元数据中的 AuthenticationAlgs 字段查找哈希算法，该哈希算法与带有 UAF_ALG_SIGN 前缀常量的第一条相关。
    - FCHash = hash(AuthenticationResponse.FinalChallengeParams)。
8. 确保 a.assertion.TAG_UAFV1_AUTH_ASSERTION.TAG_UAFV1_SIGNED_DATA.TAG_FINAL_CHALLENGE == FCHash。
    - 如果比较失败，则继续下一个断言。
9. 如果 a.assertion.TAG_UAFV1_AUTH_ASSERTION.TAG_UAFV1_SIG

NED_DATA.TAG_ASSERTION_INFO.authenticationMode == 2：

> **注释**
>
> AuthenticationResponse 中包含的交易哈希必须与相关 AuthenticationRequest 中特定的交易内容相匹配。FIDO 不限定任何特定的 FIDO 服务器 API，交易内容应当被依赖方的任意软件组件缓存，例如 FIDO 服务器或依赖方 Web 应用。

1. 确保依赖方缓存了一个交易。
    - 如果没有，则继续下一个断言。
2. 检查所有缓存的交易内容表单（可能同一交易有多个缓存 PNG 图像），并且使用适用于该认证器的哈希算法（与用于计算 FinalChallenge 的哈希算法相同）计算哈希值。
    - 对于每个 cachedTransaction，将 hash(cachedTransaction)添加到 cachedTransactionHashList。
3. 确保 a.TransactionHash 在 cachedTransactionHashList 中。
    - 如果不在列表中，则继续下一个断言。

10. 使用 UAuth.pub 公钥和适当的鉴别算法来验证 a.assertion.TAG_UAFV1_AUTH_ASSERTION.TAG_SIGNATURE：
    1. 如果签名验证失败，则继续下一个断言。
    2. 使用 a.assertion.TAG_UAFV1_AUTH_ASSERTION.TAG_UAFV1_SIGNED_DATA.SignCounter 更新用户记录中的 SignCounter。

2. 如果 a.assertion.TAG_UAFV1_AUTH_ASSERTION 含有除了 TAG_UAFV1_SIGNED_DATA 以外的对象作为第一元素，则遵循该对象的特定规则。

6. 如果 a.assertion 不包含一个 TAG_UAFV1_AUTH_ASSERTION 类型的对象，则跳过这个断言（因为 UAF v1 中只定义了

TAG_UAFV1_REG_ASSERTION）。

7. 将此断言 a 视为明确已完成校验的。

6. 处理所有明确已完成校验的鉴别断言 a。

3.6 注销操作

该操作可让 FIDO 服务器要求 FIDO 认证器删除与特定依赖方相关的密钥。

> **注释**
>
> 与其它案例一样，当用户在依赖方移除帐号时 FIDO 服务器应该也会触发此操作。

3.6.1 注销请求消息

FIDO UAF 注销请求消息以结构数组表示。每个结构包含一个特定协议版本的注销请求。数组中不得包含协议版本相同的两个结构。对于版本“1.0”来说，，DeregistrationRequest 结构定义了该请求。

例 11：UAF 注销请求

```
[{
    "header": {
      "op": "Dereg",
      "upv": {
        "major": 1,
        "minor": 0
      },
  "appID": "https://uaf-test-1.noknoktest.com:8443/SampleApp/uaf/facets"
    },
    "authenticators": [
      {
        "aaid": "ABCD#ABCD",
        "keyID": "ZMCPn92yHv1Ip-
iCiBb6i4ADq6ZOv569KFQCvYSJfNg"
      }
    ]
}]
```

> **注释**
>
> 没有注销响应对象。

3.6.2 DeregisterAuthenticator 结构

```
WebIDL
dictionary DeregisterAuthenticator {
    required AAID       aaid;
    required KeyID      keyID;
};
```

3.6.2.1 DeregisterAuthenticator 结构成员

aaid 类型为 required AAID

待注销认证器的 AAID。

keyID 类型为 required KeyID

与 UAuth.priv 相关的唯一的 KeyID。假定 KeyID 仅在同一 AAID 范围内是唯一的。

3.6.3 DeregistrationRequest 结构

```
WebIDL
dictionary DeregistrationRequest {
    required OperationHeader                header;
    required DeregisterAuthenticator[]      authenticators;
};
```

3.6.3.1 DeregistrationRequest 结构成员

header 类型为 required OperationHeader

Header.op 必须是 “Dereg”。

authenticators 类型为 required DeregisterAuthenticator 数组

待注销认证器列表。

3.6.4 注销处理规则

3.6.4.1 FIDO 服务器注销请求生成规则

FIDO 服务器必须遵守以下步骤：

1. 创建 m.header.upv 主要版本为 1，次要版本为 0 的注销请求消息 m。
2. 对于每个待注销认证器:
    1. 为每个待注销认证器创建 DeregisterAuthenticator 对象 o。
    2. 设置适当的 o.aaid 和 o.keyID。
    3. 将 o 附加到 m.authenticators。
    4. 在 FIDO 服务器账户数据库中删除相关的条目。
3. 向 FIDO UAF 客户端发送消息。

3.6.4.2 FIDO UAF 客户端注销请求处理规则

FIDO UAF 客户端必须遵循以下步骤：

1. 选择主要版本为 1，次要版本为 0 的消息。
2. 解析消息。
    - 如果 DeregistrationRequest 消息中的必填字段为空或某个字段的类型和值不符，则拒绝该操作。
3. 对于每个与消息版本 DeregistrationRequest.header.upv 一致的认证器，并且该认证器包含一个 AAID 与提供的 AAID 之一相同：
    1. 为注销功能创建合适的 ASMRequest 并发送给认证器。

3.6.4.3 FIDO 认证器注销请求处理规则

参考[UAFASM]“注销请求”。

4. 注意事项

本节是非规范性的。

4.1 协议核心设计要素

本节描述协议中用到的重要的设计元素。

4.1.1 认证器元数据

假定 FIDO 服务器能访问到所有支持的认证器列表和相应的元数据。认证器元数据[UAFAuthnrMetadata]包含这些信息，例如：

- 所支持的注册和鉴别方案。
- 鉴别因素、安装类型、所支持的内容类型和其他补充信息等等。

为了决定哪些认证器适合一个特定的交易，FIDO 服务器根据 AAID 查找认证器元数据列表并从中提取必要信息。

> **规范**
>
> 认证器元数据中的每一条目都必须用唯一的认证器鉴证标识符（AAID）识别。

4.1.2 认证器鉴证

认证器鉴证是注册期间确认认证器型号及身份有效性的过程。它可让依赖方加密校验 FIDO UAF 客户端所报告的认证器与其宣称的一样。

使用认证器鉴证，依赖方“example-rp.com” 能够校验以 AAID “1234#5678”上报的“example-Authenticator” 型号的认证器不是运行在 FIDO 用户设备上的恶意软件，而是真正的型号为"1234#5678"的认证器。

> **规范**
>
> FIDO 认证器应该支持下面描述的“基础鉴证”。 随着时间的推移，新的鉴证机制可能会加入到协议中。

规范

没有对鉴证密钥提供足够保护的 FIDO 认证器（非鉴证认证器）必须和经鉴证的认证器一样使用 UAuth.priv 密钥来正式生成 KeyRegistrationData 对象。此行为必须在认证器元数据中正确声明。

4.1.2.1 基础鉴证

规范

基础鉴证分为两种类型：

完整基础鉴证

基于鉴证私钥在同类认证器中共享（例如相同型号）。

替代基础鉴证

仅在语法上看作是基础鉴证。该鉴证对象是自签名的，例如使用 UAuth.priv 私钥签名，例如包含在鉴证对象中的 UAuth.pub 公钥所对应的密钥。因此，不提供密码验证的安全特性。但是如果认证器没有鉴证私钥，这是能够提供的最优方案。

4.1.2.1.1 完整基础鉴证

注释

为了遵循[FIDOSecRef]中的假定，FIDO 服务器必须可以访问一个信任锚以便校验鉴证公钥（例如鉴证证书可信存储）。同样的，在注册过程中认证器必须提供其鉴证签名。鉴别信任锚由 FIDO 服务器在带外共享（作为元数据的一部分），根据[UAFMetadataService]，不应当提供该共享过程。

注释

认证器鉴证私钥的保护措施取决于使用的特定认证器型号。

注释

FIDO 服务器必须用 KeyRegistrationData 对象提供的 AAID 从可信存储中加载合适的认证器鉴证根证书。

在完整基础鉴证模型中，为了提供非链接性（参考协议核心设计要素），大量的认证器必须共享相同的鉴证证书和鉴证私钥。认证器只能在批次生产级别或 AAID 级别被其鉴证证书识别而不是被单独识别。大量的认证器共享相同的鉴证证书提供了更好的隐私性，但是也使得相关的私钥成为更有吸引力的攻击目标。

注释

一个共享同一制造商和基本特征的给定的认证器集合，在使用之前的签证密钥发布不少于 100,000 台设备前，不得发布新的鉴证密钥。

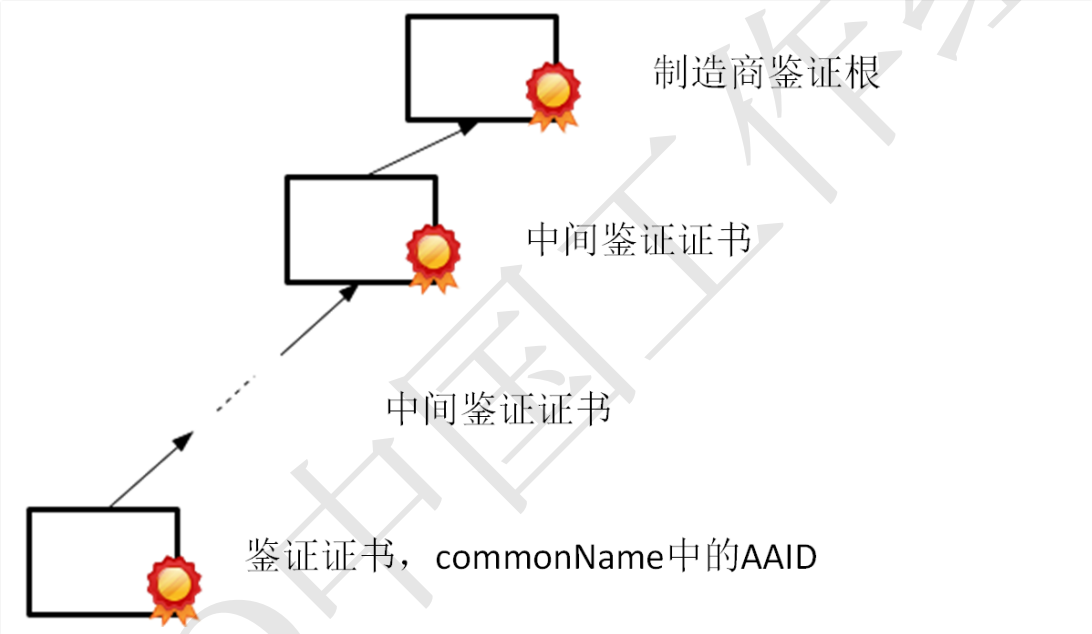

图 10 鉴证证书链

4.1.2.1.2 替代基础鉴证

规范

在此鉴证方式中，UAuth.priv 私钥必须用于对注册数据对象签名。此行为必须在认证器元数据中正确声明。

注释

没有对鉴证密钥提供足够保护的 FIDO 认证器（非鉴证认证器）必须使用此鉴证方法。

4.1.3 错误处理

> **注释**
>
> 当通过消息专用 API 产生或处理 UAF 消息时，FIDO 服务器必须将遇到的所有错误情况通知作为调用者的依赖方的 web 应用服务器（参考 FIDO 互操作性概览）。

> **规范**
>
> 当通过认证器特定模块（ASM）处理命令时，FIDO 认证器必须将遇到的所有错误情况通知 FIDO UAF 客户端（参考 FIDO 互操作性概览）。参考 [UAFASM] 和[UAFAuthnrCommands]。

4.1.4 断言方案

UAF 协议被设计用于兼容各种各样的现存认证器（可信平台模块，指纹传感器，安全元件等）以及将来为 FIDO 设计的认证器。因此，可扩展性是协议设计的核心功能。

有两个特别的方面需要小心可扩展性：

- 加密密钥准备（KeyRegistrationData）
- 加密鉴别和签名（SignedData）

KeyRegistrationData 和 SignedData 方案的结合称为断言方案。

UAF 协议允许插入新的断言方案。参考 UAF 支持的断言方案。

注册断言定义了加密密钥是如何以及以何种形式在认证器和 FIDO 服务器之间交换的。

鉴别断言定义了认证器是如何以及以何种形式产生加密签名的。

[UAFRegistry]中定义了通常支持的断言方案。

4.1.5 认证器中的用户名

FIDO UAF 支持认证器作为第一鉴别因子（例如更换用户名和密码）。在这种情

况下，认证器在内部存储用户名（在特定依赖方唯一标识一个帐号）。参考[UAFAuthnrCommands]的“签名命令”章节。

4.1.6 TLS 通信保护

> 注释
>
> 为保护 FIDO UAF 客户端与 FIDO 服务器之间的数据通信，FIDO UAF 客户端（或用户代理）和依赖方对所有协议元素都必须使用受保护的 TLS 通道。
>
> 1. TLS 连接的服务器端必须在依赖方。
> 2. TLS 连接的客户端必须在 FIDO UAF 客户端或者用户代理/应用。
> 3. TLS 客户端和服务器应该使用 TLS v1.2 或更高版本，如果 TLS v1.2 或更高版本不可用，应该仅使用 TLS v1.1。“匿名”和“未命名”TLS 加密套件是不允许的，必须拒绝；TLS 中应该避免不安全的加密算法（例如 MD5、RC4、SHA1）[[SP 800-131A]]。

我们建议：

1. 根据[RFC5280]第 6 节“证书路径验证”，TLS 客户端负责验证和确认服务器证书链。应该检查证书撤销状态（例如使用在线证书状态协议（OCSP）[RFC2560]或基于证书吊销列表（CRL）的有效性确认[RFC5280]）以及 TLS 服务器身份[RFC6125]。
2. TLS 客户端的可信证书根存储被正确维护，至少要求根存储中的 CA 每年通过 Web Trust 或欧洲电信标准化协会（ETSI）（ETSI TS 101 456 或 ETSI TS 102 042）的 SSL CA 审计。

更多关于如何使用 TLS 的建议，参考 [TR-03116-4]和[SHEFFER-TLS] 。

4.2 实施注意事项

4.2.1 服务器挑战值和随机数

> 注释
>
> ServerChallenge 需要合适的随机源以确保有效（参考[RFC4086]获取更多信

息）。用于产生服务器挑战值的（伪）随机数应该成功地通过[Coron99]中规定的随机性测试并且应该遵循[SP800-90b]给出的指导。

4.3 安全注意事项

没有“一刀切”的鉴别方法。FIDO 的目标是将用户校验方法与鉴别协议和鉴别服务器进行解耦合，并且支持范围广泛的用户校验方法和保障级别。FIDO 认证器应该能够利用现有计算硬件的能力，例如移动设备或者智能卡。

电子化的用户鉴别，其整体保障级别非常依赖于（a）相关用户设备的安全性和完整性（b）用于鉴别用户的鉴别方法。

当使用 FIDO 时，用户应可自由地使用任何可用的设备和各种各样的鉴别方法。依赖方需要关于设备和鉴别方法自身安全相关部分的可靠信息，以便来决定在特定的业务环境下，是否接受电子化鉴别的整体风险。FIDO 元数据[UAFMetadataService]用于提供这样的信息。

为 FIDO 服务器提供设备和鉴别方法自身安全相关部分的可靠信息，对于 UAF 协议而言是很重要的。

整体安全性由最弱的环节决定。在 FIDO 中，为了支持可扩展的安全，作为基础的 UAF 协议需要提供非常高的概念安全级别，以至于协议不是最弱的环节。

依赖方定义可接受的安全级别。FIDO 联盟预见了将有不同供应商提供多种多样的 FIDO UAF 客户端、FIDO 认证器和 FIDO 服务器。依赖方可以选择提供适当安全级别的 FIDO 服务器。他们也有能力接受满足给定业务环境的安全需求的 FIDO 认证器，通过增加适当的隐式鉴别措施来补偿安全级别缺陷，并且拒绝不满足他们要求的认证器。FIDO 不要求 FIDO 认证器具有非常高的保障级别，而是提供了认证器和用户校验方法进行竞争的基础。

“鉴别”对比“交易确认”。现有云服务通常基于鉴别。用户启动一个假定为可信的应用（例如用户代理）到云服务进行鉴别，目的是为了在应用和云服务之间建立一个经鉴别的通道。在鉴别后，应用能够使用鉴别通道来对云服务执行任意操作。服务提供者把所有这些操作都归属于该用户。本质上，用户提前鉴别由应用执行的所有操作，直到服务连接或鉴别过期。这是很方便的方法，因为用户不会为鉴别所需的手工操作分心。这适合于低风险结果的操作。

然而，在某些情况下，了解用户在鉴别前确实看过或者接受了特定的内容对于依赖方而言是重要的。该方法通常用于需要不可抵赖性时。该场景导致的需求被称为所见即所签（What You See Is What You Sign，简称 WYSIWYS）。

UAF 支持两种方法：称为“鉴别”和“交易确认”。技术上的区别是，“鉴别”时用户要确认一个随机挑战值，而在“交易确认”时，用户还要确认一个人类可读的内容，例如合同。从安全角度来看，在“鉴别”情况下，应用需要是可信的，因为一旦经过鉴别的通道建立，应用就可以执行任意操作。在“交易确认”的情况下，只有实现了所见即所签的交易确认显示组件需要是可信的，而不是整个应用。

显式鉴证安全组件。对依赖方来说，为了判断一次鉴别的相关风险，了解用户环境的某些的细节是很重要的。一般网络浏览器会发送一个“用户代理”字符串给网络服务器。但是任意应用都可以发送任意字符串例如“用户代理”给依赖方，所以此方法不提供强安全性。FIDO UAF 是基于加密鉴证概念的。据此概念，待验证的组件拥有加密的秘密并且凭此加密秘密来鉴别其身份。在 FIDO UAF 中，此加密秘密被称为“认证器鉴证密钥”。依赖方可以访问校验此鉴证所需的参考数据。

为了使依赖方能够恰当地判断与一次鉴别相关的风险，所有执行重要安全功能的组件都需要被鉴证。

在 FIDO UAF 中，重要安全功能在 FIDO 认证器中实现。安全功能有：

1. 保护鉴证密钥。
2. 生成和保护鉴别密钥，通常每个依赖方和依赖方的用户账户有一个。
3. 校验用户。
4. 提供 WYSIWYS 功能（“交易确认显示”组件）。

某些 FIDO 认证器可能在运行于 FIDO 用户设备上的软件中实现这些功能，其它的可能在“硬件”（例如运行于与 FIDO 用户设备隔离的硬件上的软件）中实现这些功能。某些 FIDO 认证器可能甚至被正式评估并且被某些国家或国际的方案所认可。每个 FIDO 认证器型号都有一个鉴证标识符（AAID），作为相关的安全特征的唯一识别方式。依赖方可以获得这些 FIDO 认证器的安全属性以及校验该鉴证所需的参考数据。

由其他校验者泄露的可恢复性。现有鉴别方案的重要问题之一是薄弱的服务器端实现，影响了典型用户到其它依赖方鉴别的安全。将不同依赖方的安全解耦合是 FIDO UAF 协议的目标。

将用户校验方法与鉴别协议解耦合。为了将用户校验方法与鉴别协议解耦合，FIDO UAF 是基于一个可扩展的加密鉴别算法的集合。加密秘密将在认证器完成用户校验后解锁。随后，该秘密用于认证器到依赖方的鉴别。根据现有的加密硬件和计算设备的能力来选择加密算法集合。该集合可被扩展以便支持新的加密硬件。

隐私保护。世界的不同地区有不同的隐私条例。FIDO UAF 协议应该在所有地区被接受，因此必须支持最高等级的数据保护。因此，FIDO UAF 不需要把生物特征数据传输给依赖方，也不需要在依赖方存储生物识别参考数据[ISOBiometrics]。此外，用于不同依赖方的加密秘密不允许这些依赖方将操作和相同的用户实体进行链接。UAF 支持此概念，称为不可链接性。因此，UAF 协议不需要可信第三方参与每一次交易。

依赖方可以使用发现接口[UAFAppAPIAndTransport]在 FIDO 用户设备上交互式地发现所有可用的 FIDO 认证器的 AAID。AAID 的组合添加到客户端为依赖方所提供的熵中。基于这些信息，依赖方可以在互联网上辨识客户端（参考“浏览器唯一性”Eff.org 和 https://wiki.mozilla.org/Fingerprinting）。为了最小化 FIDO 添加的熵，用户可以启用或禁用个别认证器，即使这些认证器已经嵌入到设备中（参考 [UAFAppAPIAndTransport]“隐私注意事项”章节）。

4.3.1 FIDO 认证器安全

参考 [UAFAuthnrCommands]。

4.3.2 加密算法

为了保证小设备的密钥长度足够小，并且私钥操作足够快，建议实施者优先考虑椭圆曲线数字签名算法（ECDSA）[ECDSA-ANSI]与 SHA-256 或 SHA-512 的结合。然而，RSA 算法也是支持的。一般支持的加密算法列表可参考

[UAFRegistry]“鉴别算法和密钥格式”。

椭圆曲线数字签名算法的一个特性是对每个签名的产生，都需要生成一个新的随机值。为了安全性的有效，这个值必须使用加密安全进程随机且均匀地从一个模数集合中选择。在此过程中哪怕有轻微偏差，也可能转化为对签名方案的攻击。

> **注释**
>
> 如果在所有可能的环境条件下都不能提供这样的随机值，就要使用确定性版本的椭圆曲线数字签名算法（参考[RFC6979]）。

4.3.3 应用隔离

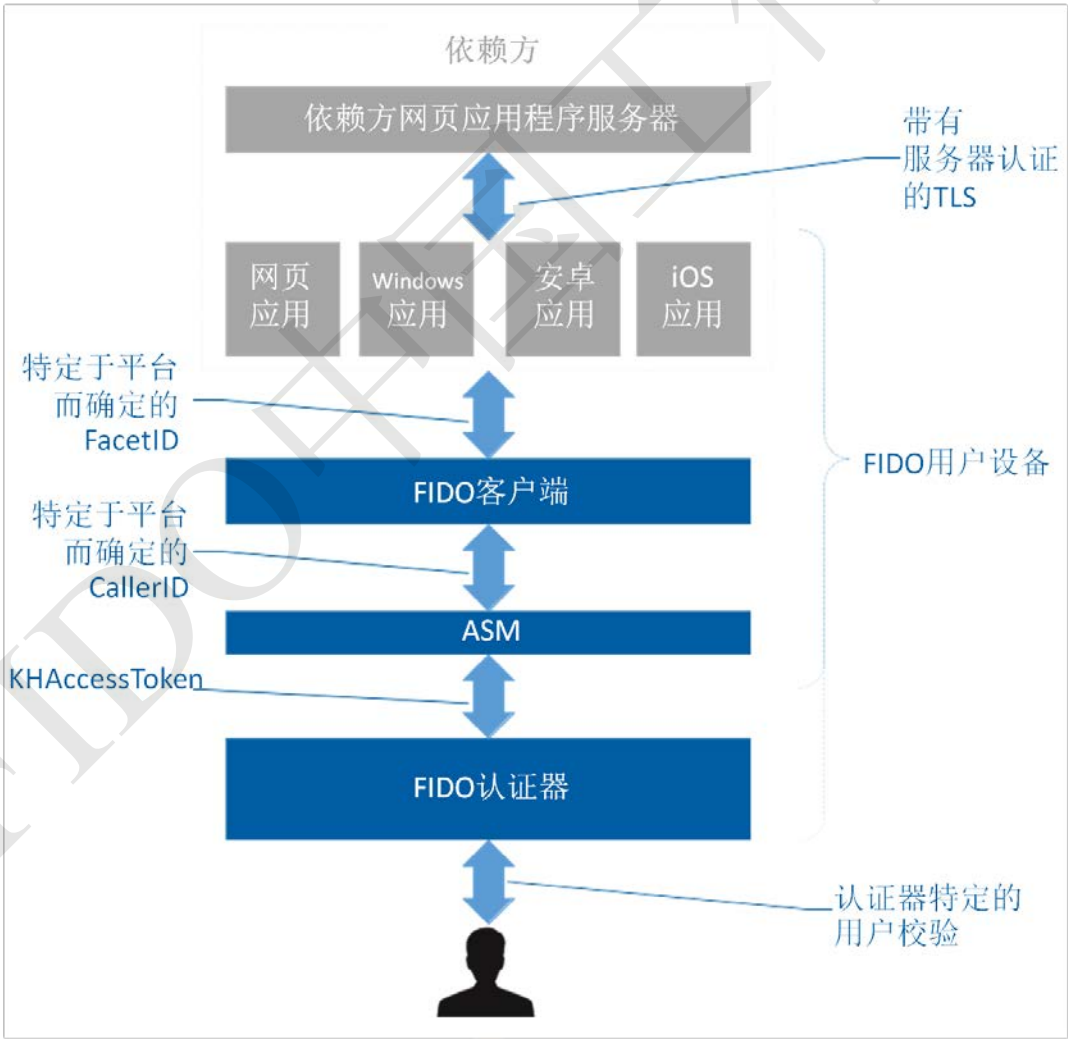

图 11 FIDO 实体校验概览

FIDO UAF 中实现了两个概念来防止恶意应用滥用 FIDO 认证器注册的 AppID

的特定密钥。第一个概念被称为“FacetID 声明"，第二个概念基于“KHAccessToken”。关于 FacetID 的概念请参考[FIDOAppIDAndFacets]。

4.3.3.1 使用密钥句柄访问令牌隔离

认证器可能在专用硬件中实现，因此可能不能校验调用软件的实体（例如 ASM）。

KHAccessToken 允许限制访问由 FIDO 认证器生成的给指定 ASM 的密钥。这是基于“首次使用时信任（Trust On First Use 简称 TOFU）”的概念。

FIDO 认证器能够将 UAuth.Key 与调用者（例如 ASM）提供的密钥进行绑定。该密钥称为 KHAccessToken。

此技术允许确保注册密钥只有最初注册的调用者能够访问。移动平台上的恶意应用不能通过绕过相关的 ASM 来访问密钥（假定此 ASM 最初注册了这些密钥）。

KHAccessToken 通常是特定于 AppID、PersonaID、ASMToken 和 CallerID 的。参考[UAFASM]。

> 注释
>
> 在某些平台上，为了与 FIDO 认证器通信，ASM 可能还另外需要特定的许可。某些平台不提供在应用间可靠地强制访问控制的方法。

4.3.4 TLS 绑定

多种通道绑定方法已经被提出（例如[RFC5929]和[ChannelID]）。

UAF 依赖 TLS 服务器鉴别来将鉴别密钥与 AppID 进行绑定。存在以下威胁：

1. 攻击者可能通过使用与依赖方相同的 AppID 而欺骗性地获取 TLS 服务器证书，而且他们也许能篡改 DNS 系统。
2. 攻击者也许能够窃取依赖方的 TLS 服务器私钥和证书，而且他们也许能篡改 DNS 系统。

存在以下功能需求：

1. UAF 交易可能跨越多个 TLS 会话。因此，[RFC5929]中定义的“tls-unique”可能难于实现。

2. 数据中心可能使用 SSL 集中器。
3. 数据中心可能为使用不同 TLS 证书的 TLS 终端实施负载均衡。因此，[RFC5929]中定义的“tls-server-end-point”（例如 TLS 服务器证书的哈希）可能是不合适的。
4. 很遗憾，在一个特定的但又相当常见的环境下，TLS 服务器证书的哈希（如同在“tls-server-end-point”中的）限制了通道绑定的实用性。如果客户端在一个可信的（对该客户端而言）代理后面操作，可信代理充当 TLS 中间人，客户端将会看到一个不同于服务器正在使用的证书。对于想要检查所有进出流量的高安全态势的公司和军事网络来说，这其实是相当常见的。如果 FIDO 服务器只得到哈希值，就没有办法将其和攻击区别开。从性能的角度来看，如果发送整个证书是可接受的，那么服务器可以检查它并判断其是否为来自非标准发布者（很可能管理上可信的）的有效名称的证书，或不同姓名的证书（几乎确定表明为一个转发攻击）。

参考 ChannelBinding 结构获取更多信息。

4.3.5 会话管理

FIDO 不定义任何特定的会话管理方法。然而一些 FIDO 功能依赖于由依赖方网页应用实现的强会话管理。

FIDO 注册

在通过传统的凭证完成对现有用户的鉴别之后，网页应用可能会触发 FIDO 注册。于是会话就被用于维护鉴别状态直到 FIDO 注册完成。

FIDO 鉴别

FIDO 鉴别成功后，会话用于在用户代理或移动应用执行操作期间维持鉴别状态。

应该遵循最优的做法来实现强会话管理（例如[OWASP2013]）。

4.3.6 角色

FIDO 通过使用依赖方特定密钥支持不同依赖方的账户的非链接性[AnonTerminology]。

有时用户在一个特定依赖方有多个账户，并且甚至想要维持这些账户间的非链接性。

目前，这是很困难的，需要某些严格实施的措施。

FIDO 不想增加更多的复杂性来维护同一个依赖方的多账户间的非链接性。

在漫游认证器的情况下，建议对不同的角色（例如“商务”、“私人”）使用不同的认证器。这样做是合理的，因为漫游认证器通常很小且不会过于昂贵。

在绑定认证器的情况下，就有所不同了。FIDO 建议此情况下采用“角色”的概念。

认证器的所有相关数据都与一个角色（例如“商务”或“私人”）相关联。认证器的某些管理接口（非 FIDO 标准的）可以允许维护和切换角色。

> **规范**
>
> 认证器必须只能“知道”或“识别”与那一时刻活动的角色相关的数据（例如鉴别密钥、用户名、密钥标识符等）。

根据此概念，用户可以切换到“私人”角色并注册新的账户。当切换回“商务”角色时，账户就不能被认证器识别了，直到用户再次切换到“私人”角色。

为了支持角色特征，FIDO 认证器特定模块 API 支持使用角色标识符（PersonaID）来识别认证器所使用的角色。角色的管理及与用户的通信超出了 FIDO 的范围。

4.3.7 服务器数据和密钥句柄

包含在 UAF 请求中的 serverData 字段（参考 Operation Header 结构）的数据被发送给 FIDO UAF 客户端，并会作为相关 UAF 响应消息的一部分返回给 FIDO 服务器。

> **注释**
>
> FIDO 服务器不应该假定对这种数据的任何隐式的完整性保护，也没有任何隐式的会话绑定。FIDO 服务器必须明确地将 serverData 与活跃会话进行绑定。

> **注释**
>
> 在某些情景中，理想的情况是保护可以存储在任意的地方的敏感数据（例如在 serverData 或 KeyHandle 中）。在这种情况下，这些敏感数据的机密性和完整性必须加以保护。这可以通过使用合适的加密算法来完成，例如使用适当加密模式的 AES，例如 CBC 或 CTR[CTRMode]。这种加密模式需要被正确地使用。例如，对于 CBC，每次加密采用新的随机初始化向量是必需的。为了获取块的长度的整数值，数据可能需要先进行填补。可以通过在密文中添加 MAC（消息鉴别码）或者数字签名来完成完整性保护，使用与加密不同的密钥，例如使用 HMAC[FIPS198-1]。或者，也可以使用诸如 AES-GCM [SP800-38D]和 AES-CCM [SP800-38C]这样的鉴别加密方案。这样的方案在一个算法中使用一个密钥提供了完整性和机密性保护。

> **注释**
>
> 在保护 serverData 时，MAC 或数字签名计算应该包含某些与相关消息绑定的数据，例如在已鉴别的 serverData 中重新包含挑战值。

4.3.8 通过 UAF 应用 API 或元数据提取认证器信息进行比对

一些认证器属性（例如用户校验方法、密钥保护、交易确认显示等）可从元数据[UAFAuthnrMetadata]或通过 FIDO UAF 应用 API 获得。这些包含在元数据中的属性是由可信源授权并提供的。当有疑问时，应该基于将从元数据中提取的属性和通过 FIDO UAF 应用 API 提取的数据进行比对而决定。

然而，通过 FIDO UAF 应用 API 提取的属性提供了一个预期来自于认证器的良好“暗示”。这样的“暗示”很好地适合于促进和优化用户体验。

4.3.9 策略校验

FIDO UAF 响应消息没有把相关 FIDO UAF 请求消息中收到的全部参数都包含进待签名对象中。因此，任何中间人攻击都能修改这些记录。

如果被修改的值是不可被接受的，FIDO 服务器会检测到这些变化。

例如，中间人攻击也许用一个指定为最弱的可能的 FIDO 认证器的策略来替换一个通用策略。如果这个最弱的可能的 FIDO 认证器与最初的策略不匹配，FIDO 服务器会检测到这个变化。（参考注册响应处理规则和鉴别响应处理规则）

4.3.10 重放攻击保护

对于重放攻击保护，FIDO UAF 协议指定两种不同的方法：

1. 安全传输层协议(TLS)。
2. 服务器挑战值。

如果实施无误，TLS 协议自身能防护重放攻击。

另外，每个协议消息在 ServerChallenge 字段中包含一些随机字节。FIDO 服务器应该只接受包含有效 ServerChallenge 值的传入的 FIDO UAF 消息。这是通过校验由客户端发送的 ServerChallenge 值是否是之前由 FIDO 服务器产生的来完成的。

应当说明的是，某些（尽管不太可能）情形下，FIDO 服务器生成的随机数可能不是唯一的，这种情况下，同样的 ServerChallenge 可能出现不止一次，这就使得重放攻击更难检测。

4.3.11 防护克隆认证器

FIDO UAF 依赖于带有特定于型号（通过 AAID 识别）的安全特色的认证器管理并保护的 UAuth.Key。当仅有一个带有特定 UAuth.Key 实例的认证器存在时，安全性更好。FIDO UAF 指定了一些保护措施来防止认证器克隆。

首先，如果 UAuth 私钥处于适当的保护措施下，由于这样的密钥不易于提取所以很难被克隆。

其次，UAF 指定了签名计数器（参考鉴别响应处理规则和[UAFAuthnrCommands]）。该计数器随着每次签名操作递增。如果使用了克隆的认证器，以后使用原来的认证器其所含签名计数器的值小于或等于之前的（恶意）操作。FIDO 服务器可以检测到这样的事件。

4.3.12 反欺诈标志

存在攻击者为欺诈而滥用 FIDO 认证器的可能性，更具体地说他们可能：

1. 为某个账号将认证器注册到依赖方。
2. 进行欺骗。
3. 注销认证器。
4. 为另一个账号将认证器注册到依赖方。
5. 进行欺骗。
6. 注销认证器。
7. 以此类推.....

> 注释
>
> 认证器可能支持注册计数器（RegCounter）。RegCounter 每次注册后会增加，因此在这样的欺诈场景中会变得非常高。参考[UAFAuthnrCommands]。

4.4 互操作性注意事项

FIDO 支持网页应用、移动应用和本地计算机应用，这些应用被称为支持 FIDO 功能的应用。

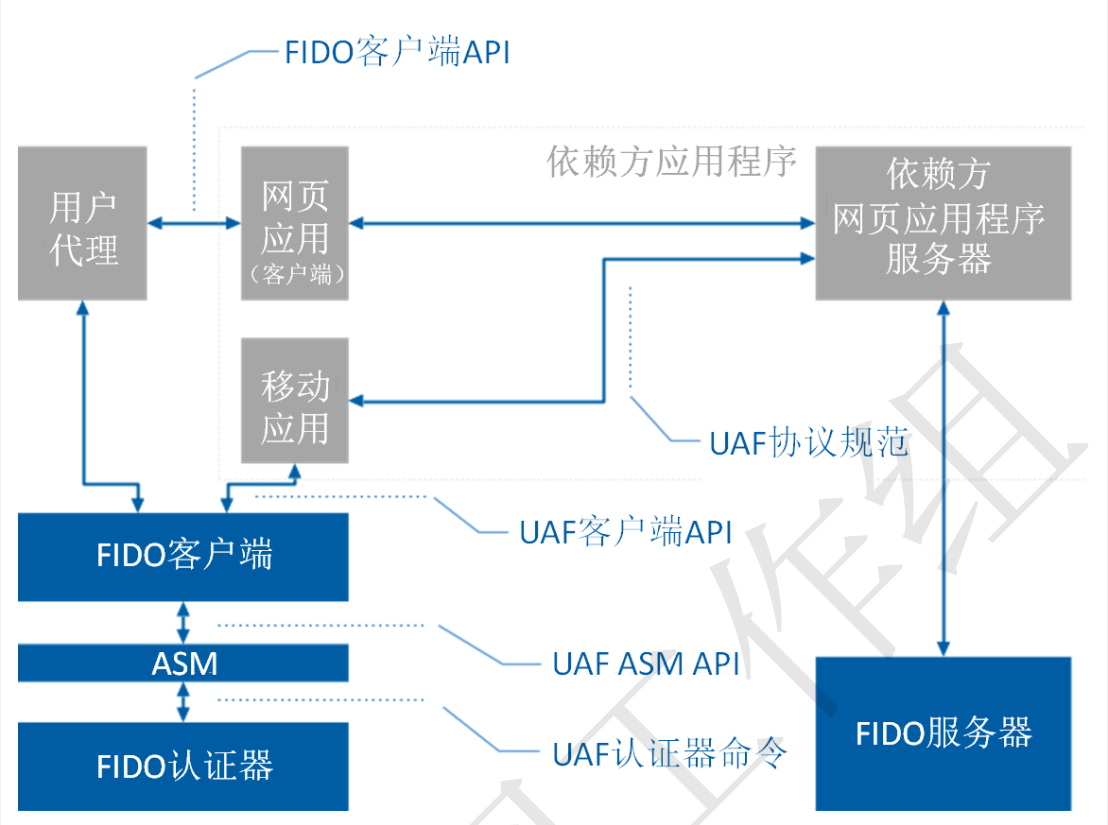

图 12 FIDO 互操作性概览

网页应用通常由网页应用服务器和相关的网页应用程序组成。网页应用程序代码（例如 HTML 和 JavaScript）通过用户代理在客户端呈现和执行。网页应用程序代码通过一套 JavaScript API 与用户代理对话，例如 HTML DOM。FIDO DOM API 在[UAFAppAPIAndTransport]中定义。网页应用程序和依赖方网页应用服务器之间的协议通常是专有的。

移动应用担任用户代理和网页应用程序（客户端）的角色。移动应用和依赖方网页应用服务器之间的协议通常是专用的。

本地计算机应用担任用户代理和网页应用程序（客户端）的角色。这些应用通常是独立于任意依赖方网页应用服务器的。

建议支持 FIDO 的应用按照本文档中规定的格式使用 FIDO 消息。

建议支持 FIDO 的应用使用[UAFAppAPIAndTransport]中定义的 UAF HTTP 绑定。

> 注释
>
> KeyRegistrationData 和 SignedData 对象[UAFAuthnrCommands]是由 FIDO 认证器生成和签名的，必须由 FIDO 服务器校验。如果在传输期间该值被修

> 改，则验证失败。
>
> ASM API[UAFASM]在桌面计算机和移动设备上指定了标准化的 API 来访问认证器特定模块（ASM）。
>
> [UAFAuthnrCommands]文档没有规定特定的协议或 API，它列出了需要从 FIDO 认证器来回传递的最小数据集和特定的消息格式。

5. UAF 支持的断言方案

本节是标准化的。

5.1 “UAFV1TLV”断言方案

该断言方案可让认证器和 FIDO 服务器交换由认证器产生的非对称鉴别密钥。该断言方案使用标签长度值（TLV）压缩编码来对认证器产生的注册和鉴别断言进行编码。这是 UAF 协议的默认断言方案。

[UAFRegistry]中定义了标签和算法。

对每个依赖方，认证器必须使用适合于元数据声明[UAFAuthnrMetadata]中指定的鉴别算法的专用密钥对（UAuth.pub/UAuth.priv）。

遵守协议的 FIDO 服务器必须支持文档[UAFRegistry]中所列的所有的鉴别算法和密钥格式。

遵守协议的认证器必须至少支持[UAFRegistry]中列出的一种鉴别算法和一种密钥格式。

5.1.1 KeyRegistrationData

参考[UAFAuthnrCommands]“TAG_UAFV1_KRD”章节。

5.1.2 SignedData

参考[UAFAuthnrCommands]“TAG_UAFV1_SIGNED_DATA”章节。

6. 定义

参考[FIDOGlossary]。

7. 图标目录

A. 参考文献

A.1 参考标准

[ABNF]

D. Crocker, Ed.; P. Overell. Augmented BNF for Syntax Specifications: ABNF. January 2008. Internet Standard.

URL: https://tools.ietf.org/html/rfc5234

[ChannelID]

D. Balfanz Transport Layer Security (TLS) Channel IDs. (Work In Progress) URL: http://tools.ietf.org/html/draft-balfanz-tls-channelid

[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

[FIDOAppIDAndFacets]

D. Balfanz, B. Hill, R. Lindemann, D. Baghdasaryan, *FIDO AppID and Facets v1.0*. FIDO Alliance Proposed Standard. URLs: HTML: fido-appid-and-facets-v1.0-ps-20141208.html PDF: fido-appid-and-facets-v1.0-ps-20141208.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

[FIPS180-4]

FIPS PUB 180-4: Secure Hash Standard (SHS). National Institute of Standards and Technology, March 2012, URL: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

[JWA]

M. Jones JSON Web Algorithms (JWA). Internet-Draft (Work in progress.) URL: http://tools.ietf.org/html/draft-ietf-jose-json-web-algorithms

[JWK]

Mike Jones. JSON Web Key (JWK). 28 May 2013. Internet Draft. URL: http://tools.ietf.org/html/draft-ietf-jose-json-web-key-11

[RFC1321]

R. Rivest, The MD5 Message-Digest Algorithm (RFC 1321), IETF, April 1992, URL: http://www.ietf.org/rfc/rfc1321.txt

[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

[RFC3629]

F. Yergeau. UTF-8, a transformation format of ISO 10646. November 2003. Internet Standard. URL: https://tools.ietf.org/html/rfc3629

[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

[RFC4627]

D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). July 2006. Informational. URL: https://tools.ietf.org/html/rfc4627

[RFC4648]

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

[RFC5056]

N. Williams, On the Use of Channel Bindings to Secure Channels (RFC 5056), IETF, November 2007, URL: http://www.ietf.org/rfc/rfc5056.txt

[RFC5280]

D. Cooper, S. Santesson, s. Farrell, S.Boeyen, R. Housley, W. Polk; Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, IETF, May 2008, URL: http://www.ietf.org/rfc/rfc5280.txt

[RFC5929]

J. Altman, N. Williams, L. Zhu, Channel Bindings for TLS (RFC 5929), IETF, July 2010, URL: http://www.ietf.org/rfc/rfc5929.txt

[RFC6234]

D. Eastlake 3rd, T. Hansen, US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF) (RFC 6234), IETF, May 2011, URL: http://www.ietf.org/rfc/rfc6234.txt

[RFC6979]

T. Pornin, Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA) (RFC6979), IETF, August 2013, URL:http://www.ietf.org/rfc/rfc6979.txt

[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

[UAFASM]

D. Baghdasaryan, J. Kemp, R. Lindemann, B. Hill, R. Sasson, *FIDO UAF Authenticator-Specific Module API*. FIDO Alliance Proposed Standard. URLs: HTML: fido-uaf-asm-api-v1.0-ps-20141208.html

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

[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

[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

[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

[AnonTerminology]

"Anonymity, Unlinkability, Unobservability, Pseudonymity, and Identity Management - A Consolidated Proposal for Terminology", Version 0.34,. A. Pfitzmann and M. Hansen, August 2010. URL: http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf

[CTRMode]

H. Lipmea, P. Rogaway, D. Wagner, Comments to NIST concerning AES Modes of Operation: CTR-Mode Encryption. National Institute of Standards and Technology, accessed March 11, 2014, URL: http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ctr/ctr-spec.pdf

[ECDSA-ANSI]

Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), ANSI X9.62-2005. American National Standards Institute, November 2005, URL: http://webstore.ansi.org/RecordDetail.aspx?sku=ANSI+X9.62%3A2005

[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

[FIPS198-1]

FIPS PUB 198-1: The Keyed-Hash Message Authentication Code (HMAC). National Institute of Standards and Technology, July 2008, URL:http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf

[ISOBiometrics]

Project Editor, Harmonized Biometric Vocabulary. ISO/IEC JTC 1. 15 November 2007, URL: http://isotc.iso.org/livelink/...

[OWASP2013]

OWASP 2013. OWASP Top 10 - 2013. The Ten Most Critical Web Application Security Risks

[RFC2560]

M. Myers; R. Ankney; A. Malpani; S. Galperin; C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. June 1999. Proposed Standard. URL: https://tools.ietf.org/html/rfc2560

[RFC6125]

P. Saint-Andre, J. Hodges, Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS) (RFC 6125), IETF, March 2011, URL: http://www.ietf.org/rfc/rfc6125.txt

[RFC6287]

D. M'Raihi, J. Rydell, S. Bajaj, S. Machani, D. Naccache, OCRA: OATH Challenge-Response Algorithm (RFC 6287), IETF, June 2011, URL:http://www.ietf.org/rfc/rfc6287.txt

[SHEFFER-TLS]

Y. Sheffer, R. Holz, P. Saint-Andre Recommendations for Secure Use of TLS and DTLS. Internet-Draft (Work in progress.) URL: https://tools.ietf.org/html/draft-sheffer-tls-bcp

[SP800-38C]

M. Dworkin, NIST Special Publication 800-38C: Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. National Institute of Standards and Technology, July 2007, URL: http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf

[SP800-38D]

M. Dworkin. NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. November 2007

URL:http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf

[SP800-63]

W. Burr, D. Dodson, E. Newton, R. Perlner, W.T. Polk, S. Gupta and E. Nabbus, NIST Special Publication 800-63-2: Electronic Authentication Guideline. National Institute of Standards and Technology, August 2013, URL: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-2.pdf

[TLS]

T. Dierks; E. Rescorla. The Transport Layer Security (TLS) Protocol, Version 1.2. August 2008. RFC 5246.

URL: http://tools.ietf.org/html/rfc5246

[TR-03116-4]

Technische Richtlinie TR-03116-4: eCard-Projekte der Bundesregierung: Teil 4 – Vorgaben für Kommunikationsverfahren im eGovernment. Bundesamt für Sicherheit in der Informationstechnik, 2013,

URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03116/BSI-TR-03116-4.pdf

[UAFMetadataService]

R. Lindemann, B. Hill, D. Baghdasaryan, *FIDO UAF Metadata Service v1.0*. FIDO Alliance Proposed Standard. URLs:

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

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

[WebIDL]

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