IMPLEMENTATION DRAFT

FIDO AppID and Facet Specification

FIDO Alliance Implementation Draft 02 February 2017

This version:
https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-appid-and-facets-v1.1-id-20170202.html
Previous version:
http://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-appid-and-facets-v1.0-ps-20141208.html
Editor:
Rolf Lindemann, Nok Nok Labs, Inc.
Contributors:
Brad Hill, PayPal, Inc.
Dirk Balfanz, Google, Inc.
Davit Baghdasaryan, Nok Nok Labs, Inc.

Abstract

The FIDO family of protocols introduce a new security concept, *Application Facets*, to describe the scope of user credentials and how a trusted computing base which supports application isolation may make access control decisions about which keys can be used by which applications and web origins.

This document describes the motivations for and requirements for implementing the Application Facet concept and how it applies to the FIDO protocols.

Status of This Document

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

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

Table of Contents

1. Notation

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

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

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

This document applies to both the U2F protocol and the UAF protocol. UAF specific terminology used in this document is defined in [FIDOGlossary].

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

1.1 Key Words

The key words “must”, “must not”, “required”, “shall”, “shall not”, “should”, “should not”, “recommended”, “may”, and “optional” in this document are to be interpreted as described in [RFC2119].

2. Overview

This section is non-normative.

Modern networked applications typically present several ways that a user can interact with them. This document introduces the concept of an *Application Facet* to describe the identities of a single logical application across various platforms. For example, the application MyBank may have an Android app, an iOS app, and a Web app accessible from a browser. These are all facets of the MyBank application.

The FIDO architecture provides for simpler and stronger authentication than traditional username and password approaches while avoiding many of the shortfalls of alternative authentication schemes. At the core of the FIDO protocols are challenge and response operations performed with a public/private keypair that serves as a user's credential.

To minimize frequently-encountered issues around privacy, entanglements with concepts of "identity", and the necessity for trusted third parties, keys in FIDO are tightly scoped and dynamically provisioned between the user and each Relying Party and only optionally associated with a server-assigned username. This approach contrasts with, for example, traditional PKIX client certificates as used in TLS, which introduce a trusted third party, mix in their implementation details identity assertions with holder-of-key cryptographic proofs, lack audience restrictions, and may even be sent in the cleartext portion of a protocol handshake without the user's notification or consent.

While the FIDO approach is preferable for many reasons, it introduces several challenges.

- What set of Web origins and native applications (facets) make up a single logical application and how can they be reliably identified?
- How can we avoid making the user register a new key for each web browser or application on their device that accesses services controlled by the same target entity?
- How can access to registered keys be shared without violating the security guarantees around application isolation and protection from malicious code that users expect on their devices?
- How can a user roam credentials between multiple devices, each with a user-friendly Trusted Computing Base for FIDO?

This document describes how FIDO addresses these goals (where adequate platform mechanisms exist for enforcement) by allowing an application to declare a credential scope that crosses all the various facets it presents to the user.

2.1 Motivation

FIDO conceptually sets a scope for registered keys to the tuple of (Username, Authenticator, Relying Party). But what constitutes a Relying Party? It is quite common for a user to access the same set of services from a Relying Party, on the same device, in one or more web browsers as well as one or more dedicated apps. As the Relying Party may require the user to perform a costly ceremony in order to prove her identity and register a new FIDO key, it is undesirable that the user should have to repeat this ceremony multiple times on the same device, once for each browser or app.

2.2 Avoiding App-Phishing

FIDO provides for user-friendly verification ceremonies to allow access to registered keys, such as entering a simple PIN code and touching a device, or scanning a finger. It should not matter for security purposes if the user re-uses the same verification inputs across Relying Parties, and in the case of a biometric, she may have no choice.

Modern operating systems that use an "app store" distribution model often make a promise to the user that it is "safe to try" any app. They do this by providing strong isolation between applications, so that they may not read each others' data or mutually interfere, and by requiring explicit user permission to access shared system resources.

If a user were to download a maliciously constructed game that instructs her to activate her FIDO authenticator in order to "save your progress" but actually unlocks her banking credential and takes over her account, FIDO has failed, because the risk of phishing has only been moved from the password to an app download. FIDO must not violate a platform's promise that any app is "safe to try" by keeping good custody of the high-value shared state that a registered key represents.

2.3 Comparison to OAuth and OAuth2

The OAuth and OAuth2 of protocols were designed for a server-to-server security model with the assumption that each application instance can be issued, and keep, an "application secret". This approach is ill-suited to the "app store" security model. Although it is common for services to provision an OAuth-style application secret into their apps in an attempt to allow only authorized/official apps to connect, any such "secret" is in fact shared among everyone with access to the app store and can be trivially recovered thorough basic reverse engineering.

In contrast, FIDO's facet concept is designed for the "app store" model from the start. It relies on client-side platform isolation features to make sure that a key registered by a user with a member of a well-behaved "trusted club" stays within that trusted club, even if the user later installs a malicious app, and does not require any secrets hard-coded into a shared package to do so. The user must, however, still make good decisions about which apps and browsers they are willing to preform a registration ceremony with. App store policing can assist here by removing applications which solicit users to register FIDO keys to for Relying Parties in order to make illegitmate or fraudulent use of them.

2.4 Non-Goals

The *Application Facet* concept does not attempt to strongly identify the calling application to a service across a network. Remote attestation of an application identity is an explicit non-goal.

If an unauthorized app can convince a user to provide all the information to it required to register a new FIDO key, the Relying Party cannot use FIDO protocols or the Facet concept to recognize as unauthorized, or deny such an application from performing FIDO operations, and an application that a user has chosen to trust in such a manner can also share access to a key outside of the mechanisms described in this document.

The facet mechanism provides a way for registered keys to maintain their proper scope when created and accessed from a *Trusted Computing Base* (TCB) that provides isolation of malicious apps. A user can also roam their credentials between multiple devices with user-friendly TCBs and credentials will retain their proper scope if this mechanism is correctly implemented by each. However, no guarantees can be made in environments where the TCB is user-hostile, such as a device with malicious code operating with "root" level permissions. On environments that do not provide application isolation but run all code with the privileges of the user, (e.g. traditional desktop operating systems) an intact TCB, including web browsers, may successfully enforce scoping of credentials for web origins only, but cannot meaningfully enforce application scoping.

3. The AppID and FacetID Assertions

When a user performs a Registration operation [UAFArchOverview] a new private key is created by their authenticator, and the public key is sent to the Relying Party. As part of this process, each key is associated with an `AppID`. The `AppID` is a URL carried as part of the protocol message sent by the server and indicates the target for this credential. By default, the audience of the credential is restricted to the *Same Origin* of the `AppID`. In some circumstances, a Relying Party may desire to apply a larger scope to a key. If that `AppID` URL has the `https` scheme, a FIDO client may be able to dereference and process it as a `TrustedFacetList` that designates a scope or audience restriction that includes multiple facets, such as other web origins within the same DNS zone of control of the AppID's origin, or URLs indicating the identity of other types of trusted facets such as mobile apps.

> NOTE
>
> Users may also register multiple keys on a single authenticator for an `AppID`, such as for cases where they have multiple accounts. Such registrations may have a Relying Party assigned username or local nicknames associated to allow them to be distinguished by the user, or they may not (e.g. for 2nd factor use cases, the user account associated with a key may be communicated out-of-band to what is specified by FIDO protocols). All registrations that share an `AppID`, also share these same audience restriction.

3.1 Processing Rules for AppID and FacetID Assertions

3.1.1 Determining the FacetID of a Calling Application

In the Web case, the FacetID must be the Web Origin [RFC6454] of the web page triggering the FIDO operation, written as a URI with an empty path. Default ports are omitted and any path component is ignored.

An example FacetID is shown below:

```
https://login.mycorp.com/
```

In the Android [ANDROID] case, the FacetID must be a URI derived from the Base64 encoding SHA-1 hash of the APK signing certificate [APK-Signing]:

```
android:apk-key-hash:<base64_encoded_sha1_hash-of-apk-signing-cert>
```

The SHA-1 hash can be computed as follows:

EXAMPLE 1: Computing an APK signing certificate hash

```
# Export the signing certificate in DER format, hash, base64 encode and trim '='

keytool -exportcert \
        -alias <alias-of-entry> \
        -keystore <path-to-apk-signing-keystore> &>2 /dev/null | \
        openssl sha1 -binary | \
        openssl base64 | \
        sed 's/=//g'
```

The Base64 encoding is the the "Base 64 Encoding" from Section 4 in [RFC4648], with padding characters removed.

In the iOS [iOS] case, the FacetID must be the BundleID [BundleID] URI of the application:

```
ios:bundle-id:<ios-bundle-id-of-app>
```

3.1.2 Determining if a Caller's FacetID is Authorized for an AppID

1. If the AppID is not an HTTPS URL, and matches the FacetID of the caller, no additional processing is necessary and the operation may proceed.
2. If the AppID is null or empty, the client must set the AppID to be the FacetID of the caller, and the operation may proceed without additional processing.
3. If the caller's FacetID is an `https://` Origin sharing the same host as the AppID, (e.g. if an application hosted at `https://fido.example.com/myApp` set an AppID of `https://fido.example.com/myAppId`), no additional processing is necessary and the operation may proceed. This algorithm may be continued asynchronously for purposes of caching the Trusted Facet List, if desired.
4. Begin to fetch the Trusted Facet List using the HTTP GET method. The location must be identified with an HTTPS URL.
5. The URL must be dereferenced with an anonymous fetch. That is, the HTTP GET must include no cookies, authentication, Origin or Referer headers, and present no TLS certificates or other forms of credentials.
6. The response must set a MIME Content-Type of "application/fido.trusted-apps+json".
7. The caching related HTTP header fields in the HTTP response (e.g. “Expires”) should be respected when fetching a Trusted Facets List.
8. The server hosting the Trusted Facets List must respond uniformly to all clients. That is, it must not vary the contents of the response body based on any credential material, including ambient authority such as originating IP address, supplied with the request.

9. If the server returns an HTTP redirect (status code 3xx) the server must also send the HTTP header `FIDO-AppID-Redirect-Authorized: true` and the client must verify the presence of such a header before following the redirect. This protects against abuse of open redirectors within the target domain by unauthorized parties. If this check has passed, restart this algorithm from step 4.
10. A Trusted Facet List may contain an unlimited number of entries, but clients may truncate or decline to process large responses.
11. From among the objects in the `trustedFacet` array, select the one with the `version` matching that of the protocol message version.
12. The scheme of URLs in `ids` must identify either an application identity (e.g. using the `apk:`, `ios:` or similar scheme) or an `https:` Web Origin [RFC6454].
13. Entries in `ids` using the `https://` scheme must contain only scheme, host and port components, with an optional trailing /. Any path, query string, username/password, or fragment information must be discarded.
14. All Web Origins listed must have host names under the scope of the same least-specific private label in the DNS, using the following algorithm:
    1. Obtain the list of public DNS suffixes from https://publicsuffix.org/list/effective_tld_names.dat (the client may cache such data), or equivalent functionality as available on the platform.
    2. Extract the host portion of the original AppID URL, before following any redirects.
    3. The least-specific private label is the portion of the host portion of the AppID URL that matches a public suffix plus one additional label to the left.
    4. For each Web Origin in the TrustedFacets list, the calculation of the least-specific private label in the DNS must be a case-insensitive match of that of the AppID URL itself. Entries that do not match must be discarded.
15. If the TrustedFacets list cannot be retrieved and successfully parsed according to these rules, the client must abort processing of the requested FIDO operation.
16. After processing the `trustedFacets` entry of the correct `version` and removing any invalid entries, if the caller's FacetID matches one listed in `ids`, the operation is allowed.

3.1.3 TrustedFacets structure

The JSON resource hosted at the AppID URL consists of a dictionary containing a single member, `trustedFacets` which is an array of `TrustedFacets` dictionaries.

WebIDL

```
dictionary TrustedFacets {
    Version     version;
    DOMString[] ids;
};
```

3.1.3.1 Dictionary `TrustedFacets` Members

`version` of type Version
: The protocol version to which this set of trusted facets applies. See [UAFProtocol] for the definition of the `version` structure.

`ids` of type array of DOMString
: An array of URLs identifying authorized facets for this AppID.

3.1.4 AppID Example 1:

".com" is a public suffix. "https://www.example.com/appID" is provided as an AppID. The body of the resource at this location contains:

EXAMPLE 2

```
{
  "trustedFacets" : [{
    "version": { "major": 1, "minor" : 0 },
    "ids": [
    "https://register.example.com", // VALID, shares "example.com" label
    "https://fido.example.com",     // VALID, shares "example.com" label
    "http://www.example.com",       // DISCARD, scheme is not https:
    "http://www.example-test.com",  // DISCARD, "example-test.com" does not match
    "https://www.example.com:444"   // VALID, port is not significant
    ]
  }]
}
```

For this policy, "https://www.example.com" and "https://register.example.com" would have access to the keys registered for this AppID, and "https://user1.example.com" would not.

3.1.5 AppID Example 2:

"hosting.example.com" is a public suffix, operated under "example.com" and used to provide hosted cloud services for many companies. "https://companyA.hosting.example.com/appID" is provided as an AppID. The body of the resource at this location contains:

EXAMPLE 3

```
{
  "trustedFacets" : [{
    "version": { "major": 1, "minor" : 0 },
    "ids": [
        "https://register.example.com",              // DISCARD, does not share  "companyA.hosting.example.com" label
        "https://fido.companyA.hosting.example.com", // VALID, shares "companyA.hosting.example.com" label
        "https://xyz.companyA.hosting.example.com",  // VALID, shares "companyA.hosting.example.com" label
        "https://companyB.hosting.example.com"       // DISCARD, "companyB.hosting.example.com" does not match
     ]
  }]
}
```

For this policy, "https://fido.companyA.hosting.example.com" would have access to the keys registered for this AppID, and "https://register.example.com" and "https://companyB.hosting.example.com" would not as a public-suffix exists between these DNS names and the AppID's.

3.1.6 Obtaining FacetID of Android Native App

This section is non-normative.

The following code demonstrates how a FIDO Client can obtain and construct the FacetID of a calling Android native application.

EXAMPLE 4: AndroidFacetID

```
private String getFacetID(Context aContext, int callingUid) {

    String packageNames[] = aContext.getPackageManager().getPackagesForUid(callingUid);

    if (packageNames == null) {
        return null;
    }

    try {
        PackageInfo info = aContext.getPackageManager().getPackageInfo(packageNames[0], PackageManager.GET_SIGNATURES);

        byte[] cert = info.signatures[0].toByteArray();
        InputStream input = new ByteArrayInputStream(cert);

        CertificateFactory cf = CertificateFactory.getInstance("X509");
        X509Certificate c = (X509Certificate) cf.generateCertificate(input);

        MessageDigest md = MessageDigest.getInstance("SHA1");

        return "android:apk-key-hash:" +
                  Base64.encodeToString(md.digest(c.getEncoded()), Base64.DEFAULT | Base64.NO_WRAP | Base64.NO_PADDING);
    }
    catch (PackageManager.NameNotFoundException e) {
        e.printStackTrace();
    }
    catch (CertificateException e) {
        e.printStackTrace();
    }
    catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    catch (CertificateEncodingException e) {
        e.printStackTrace();
    }

    return null;
}
```

3.1.7 Additional Security Considerations

The UAF protocol supports passing FacetID to the FIDO Server and including the FacetID in the computation of the authentication response.

Trusting a web origin facet implicitly trusts all subdomains under the named entity because web user agents do not provide a security barrier between such origins. So, in AppID Example 1, although not explicitly listed, "https://foobar.register.example.com" would still have effective access to credentials registered for the AppID "https://www.example.com/appID" because it can effectively act as "https://register.example.com".

The component implementing the controls described here must reliably identify callers to securely enforce the mechanisms. Platform inter-process communication mechanisms which allow such identification should be used when available.

It is unlikely that the component implementing the controls described here can verify the integrity and intent of the entries on a `TrustedFacetList`. If a trusted facet can be compromised or enlisted as a *confused deputy* [FIDOGlossary] by a malicious party, it may be possible to trick a user into completing an authentication ceremony under the control of that malicious party.

3.1.7.1 Wildcards in TrustedFacet identifiers

This section is non-normative.

Wildcards are not supported in TrustedFacet identifiers. This follows the advice of RFC6125 [RFC6125], section 7.2.

FacetIDs are URIs that uniquely identify specific security principals that are trusted to interact with a given registered credential. Wildcards introduce undesirable ambiguitiy in the defintion of the principal, as there is no consensus syntax for what wildcards mean, how they are expanded and where they can occur across different applications and protocols in common use. For schemes indicating application identities, it is not clear that wildcarding is appropriate in any fashion. For Web Origins, it broadly increases the scope of the credential to potentially include rogue or buggy hosts.

Taken together, these ambiguities might introduce exploitable differences in identity checking behavior among client implementations and would necessitate overly complex and inefficient identity checking algorithms.

A. References

A.1 Normative references

[FIDOGlossary]
R. Lindemann, D. Baghdasaryan, B. Hill, J. Hodges, *FIDO Technical Glossary*. FIDO Alliance Implementation Draft. URLs:
HTML: https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-glossary-v1.1-id-20170202.html
PDF: https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-glossary-v1.1-id-20170202.pdf

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

[RFC4648]
S. Josefsson, *The Base16, Base32, and Base64 Data Encodings (RFC 4648)*, IETF, October 2006, URL: http://www.ietf.org/rfc/rfc4648.txt

[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

[RFC6454]
A. Barth, *The Web Origin Concept (RFC 6454)*, IETF, June 2011, URL: http://www.ietf.org/rfc/rfc6454.txt

[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: https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-protocol-v1.1-id-20170202.html

PDF: https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-protocol-v1.1-id-20170202.pdf

A.2 Informative references

[ANDROID]
The Android™ Operating System. Google, Inc., the Open Handset Alliance and the Android Open Source Project (Work in progress) URL: http://developer.android.com/
[APK-Signing]
Signing Your Applications.The Android™ Operating System. Google, Inc., the Open Handset Alliance and the Android Open Source Project (Accessed 11-March-2014) URL: http://developer.android.com/tools/publishing/app-signing.html
[BundleID]
"Configuring your Xcode Project for Distribution", section "About Bundle IDs", Apple, Inc. Accessed March 11, 2014. URL: https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/ConfiguringYourApp/ConfiguringYourApp.html
[UAFArchOverview]
S. Machani, R. Philpott, S. Srinivas, J. Kemp, J. Hodges,*FIDO UAF Architectural Overview*. FIDO Alliance Proposed Standard. URLs:
HTML: https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-overview-v1.1-id-20170202.html
PDF: https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-overview-v1.1-id-20170202.pdf
[iOS]
iOS Dev Center Apple, Inc. (Accessed March 11, 2014) URL:https://developer.apple.com/devcenter/ios/index.action