Universal 2nd Factor (U2F) Overview

Specification Set: fido-u2f-v1.0-rd-20140209 REVIEW DRAFT

Editors:

Sampath Srinivas, Google
Dirk Balfanz, Google
Eric Tiffany, FIDO Alliance

Contributors:

Member entities of the U2F working group

Abstract:

The FIDO U2F protocol enables relying parties to offer a strong cryptographic 2nd factor option for end user security. The relying party's dependence on passwords is reduced. The password can even be simplified to a 4 digit PIN. End users carry a single U2F device which works with any relying party supporting the protocol. The user gets the convenience of a single "keychain" device and convenient security. This document is an overview of the U2F protocol and is a recommended first-read before reading detailed protocol documents.

Status:

This Specification has been prepared by FIDO Alliance, Inc. **This is a Review Draft Specification and is not intended to be a basis for any implementations as the Specification may change**. Permission is hereby granted to use the Specification solely for the purpose of reviewing the Specification. No rights are granted to prepare derivative works of this Specification. Entities seeking permission to reproduce portions of this Specification for other uses must contact the FIDO Alliance to determine whether an appropriate license for such use is available.

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

THIS FIDO ALLIANCE SPECIFICATION IS PROVIDED "AS IS" AND WITHOUT ANY WARRANTY OF ANY KIND, INCLUDING, WITHOUT LIMITATION, ANY EXPRESS OR IMPLIED WARRANTY OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Table of Contents

1 What Is This Document?

This document provides an overview of the FIDO Universal 2nd Factor (U2F). It is intended to be read **before** the reader reads the detailed protocol documents listed below. It is intended to give the reader context for reading the detailed documents. This document is intended as an interpretive aid – it is not normative.

After reading this overview, it is recommended that the reader go through the detailed protocol documents listed below in the order they are listed. That order starts the reader at the top layer which is the U2F API and progresses down to lower layers such as the transport framing to the U2F device.

1. **FIDO U2F Javascript API**
2. **FIDO U2F Raw Message Formats**
3. **FIDO U2F USB Framing of APDUs**
4. **FIDO U2F Application Isolation through Facet Identification**
5. **FIDO U2F Implementation Considerations**
6. **FIDO Security Reference**

A glossary of terms used in the FIDO specifications is also available:

7. **FIDO Glossary**

These documents may all be found on the FIDO Alliance website at http://fidoalliance.org/specifications/download/

2 Background

The FIDO Alliance mission is to change the nature of online strong authentication by:

- Developing technical specifications defining open, scalable, interoperable mechanisms that supplant reliance on passwords to securely authenticate users of online services.
- Operating industry programs to help ensure successful worldwide adoption of the specifications.
- Submitting mature technical specifications to recognized standards development organization(s) for formal standardization.

The core ideas driving the FIDO Alliance's efforts are 1) ease of use, 2) privacy and security, and 3) standardization. The primary objective is to enable online services and websites, whether on the open Internet or within enterprises, to leverage native security features of end-user computing devices for strong user authentication and to reduce the problems associated with creating and remembering many online credentials.

There are two key protocols included in the FIDO architecture that cater to two basic options for user experience when dealing with Internet services. The two protocols share many of underpinnings but are tuned to the specific intended use cases.

Universal 2nd Factor (U2F) Protocol

The U2F protocol allows online services to augment the security of their existing password infrastructure by adding a strong second factor to user login. The user logs in with a username and password as before. The service can also prompt the user to present a second factor device at any time it chooses. The strong second factor allows the service to simplify its passwords (e.g. 4–digit PIN) without compromising security.

During registration and authentication, the user presents the second factor by simply pressing a button on a USB device or tapping over NFC. The user can use their FIDO U2F device across all online services that support the protocol leveraging built–in support in web browsers.

This document that you are reading gives an overview of the U2F protocol.

Universal Authentication Framework (UAF) Protocol

The UAF protocol allows online services to offer password-less and multi-factor security. The user registers their device to the online service by selecting a local authentication mechanism such as swiping a finger, looking at the camera, speaking into the mic, entering a PIN, etc. The UAF protocol allows the service to select which mechanisms are presented to the user.

Once registered, the user simply repeats the local authentication action whenever they need to authenticate to the service. The user no longer needs to enter their password when authenticating from that device. UAF also allows experiences that combine multiple authentication mechanisms such as fingerprint + PIN.

Please refer to the FIDO website for an overview and documentation set focused on the UAF protocol.

3 Goal: Strong Authentication and Privacy for the Web

The U2F eco-system is designed to provide strong authentication for users on the web while preserving the user's privacy. The user carries a "U2F device" device as a second factor.

When the user **registers** the U2F device at an account at a particular origin (such as www.company.com) the device creates a new key pair usable only at that origin and gives the origin the public key to associate with the account. When the user **authenticates** (i.e., logs in) to the origin, in addition to username and password, the origin (in this case, www.company.com) can check whether the user has the U2F device by verifying a signature created by the device.

The user is able to use the same device across multiple sites on the web – it thus serves as the user's physical web keychain – with multiple (virtual) keys to various sites provisioned from one physical device. Using the open U2F standard, any origin will be able to use any browser (or OS) which has U2F support to talk to any U2F compliant device presented by the user to enable strong authentication.

The U2F device registration and authentication operations are exposed through Javascript APIs built into the browser and, in following phases, native APIs in mobile OSes.

The U2F device can be embodied in various form factors, such as stand alone USB devices, stand alone Near Field Communication (NFC) device, stand alone BlueTooth LE devices, built-on-board the user's client machine/mobile device as pure software or utilizing secured crypto capabilities. It is strongly preferable to have hardware backed security, but it is not a requirement. However, as we shall see the protocol provides an attestation mechanism which allows the accepting online service or website to identify the class of device and either accept it or not depending on the particular site's policy.

The specs for U2F are in two layers. The upper layer specifies the cryptographic core of the protocol. The lower layer specifies how the user's client will communicate U2F cryptographic requests to the U2F device over a particular transport protocol (e.g., USB, NFC, BlueTooth LE, built-in on a particular OS, etc).

The current spec set from the U2F group specifies the upper layer (which is unchanged regardless of transport) and the lower layer for the USB transport. Later phases of the protocol spec will specify transports for U2F over NFC, BlueTooth and when built-in (i.e, where the U2F capability is built into the device and accessed locally via the OS).

As one of the founders of the U2F working group in FIDO, Google is working to build U2F support into the Chrome browser and will offer U2F as a 2nd factor option on Google accounts to help the start-up of the open ecosystem.

A critical factor for success will be that a U2F device "just works" with any modern client device owned by the user without needing additional driver or middleware setup. In this spirit, the USB U2F device is designed to work out of box with existing consumer operating systems with no driver installs or software changes. A U2F device-aware browser is able to discover and communicate with U2F devices using standard built-in OS APIs. To this end, in the first USB based deliverable, we are leveraging the built-in driverless libUSB device support in all modern OSes.

4 Site-Specific Public/Private Key Pairs

The U2F device and protocol need to guarantee user privacy and security. At the core of the protocol, the U2F device has a capability (ideally, embodied in a secure element) which mints an **origin-specific** public/private key pair. The U2F device gives the public key and a *Key Handle* to the origin online service or website during the user registration step.

Later, when the user performs an authentication, the origin online service or website sends the *Key Handle* back to the U2F device via the browser. The U2F device uses the *Key Handle* to identify the user's private key, and creates a signature which is sent back to the origin to verify the presence of the U2F device. Thus, the *Key Handle* is simply an identifier of a particular key on the U2F device.

The key pair created by the U2F device during registration is **origin specific.** During registration, the browser sends the U2F device a hash of the origin (combination of protocol, hostname and port). The U2F device returns a public key and a *Key Handle.* Very importantly, the U2F device encodes the requesting origin into the *Key Handle.*

Later, when the user attempts to authenticate, the server sends the user's *Key Handle* back to the browser. The browser sends this *Key Handle* and the hash of the origin which is requesting the authentication. The U2F device ensures that it had issued this *Key Handle* to that particular origin hash before performing any signing operation. If there is a mismatch no signature is returned.

This origin check ensures that the public keys and *Key Handles* issued by a U2F device to a particular online service or website cannot be exercised by a different online service or website (i.e., a site with a different name on a valid SSL certificate). This is a critical privacy property – assuming the browser is working as it should, a site can verify identity strongly with a user's U2F device only with a key which has been issued to that particular site by that particular U2F device. If this origin check was **not** present, a public key and *Key Handle* issued by a U2F device could be used as a "supercookie" which allows multiple colluding sites to strongly verify and correlate a particular user's identity.

5 Alerting the User: U2F Device “activation” & Browser Infobars

The U2F device device has a physical “test of user presence”. The user touches a button (or sensor of some kind) to “activate” the U2F device and this feeds into the device’s operation as follows:

- **Registration:** The U2F device responds to a request to generate a key pair only if it has been “activated”. Separately, the browser implementation might ensure that the javascript “ask the U2F device to issue a key pair” call always results in the user seeing an infobar dialog which asks if he/she indeed wants to allow the current site to register the U2F device.
- **Authentication:** During authentication, the browser sends some data down to the U2F device that it needs to sign (more about this later). The U2F device needs to see a “test of user presence” before it will sign – i.e, the user has to press a button on the device for example. This ensures that a signature happens only with the user’s permission. It also ensures that that malware cannot exercise the signature when the user is not present.

  When the user attempts to authenticate for the first time to a particular origin (i.e. the javascript call for “Get me a signature from the U2F device” is exercised), the browser may put up an infobar which asks if the user would like to allow the site to talk to the U2F device. In this case, the browser should also present a “Remember this” option with the infobar so that the browser can remember the permission and not ask every time. This setting can be reset (as with other browser settings).

In summary, the user will have to touch a button to register, and may also be warned by the browser. The relying party can put up screens which will walk the user through these steps. Registration is a very high value operation – it gives an origin a capability to very strongly verify a user and it needs to be taken very seriously. During authentication (or more generally, whenever the online service or website needs to strongly verify the user by requesting a signature), the user needs to activate the device to demonstrate user presence before the signature can happen.

6 Man-In-The-Middle Protections During Authentication

If a man-in-the-middle (MITM) tries to intermediate between the user and the origin during the authentication process, the U2F device protocol can detect it in most situations.

Say a user has correctly registered a U2F device with an origin and later, a MITM on a different origin tries to intermediate the authentication. In this case, the user's U2F device won't even respond, since the MITM's (different) origin name will not match the *Key Handle* that the MITM is relaying from the actual origin. U2F can also be leveraged to detect more sophisticated MITM situations as we shall see below.

As one of the return values of the U2F "sign" call, the browser returns an object which contains information about what the browser sees about the origin (we will call this the "client data" object). This "client data" includes:

a) the random challenge sent by the origin,

b) The origin host name seen by the browser for the web page making the javascript call, and

c) [optionally] if the ChannelID extension to TLS is used, the connection's channelID public key.

The browser sends a hash of this "client data" to the U2F device. In addition to the hash of the "client data", as discussed earlier, the browser sends the hash of the origin and the *Key Handle* as additional inputs to the U2F device.

When the U2F device receives the client data hash, the origin hash and the *Key Handle* it proceeds as follows: If it had indeed issued that *Key Handle* for that origin the U2F device proceeds to issue a signature across the hashed "client data" which were sent to it. This signature is returned back as another return value of the U2F "sign" call.

The site's web page which made the U2F "sign" call sends the return values – both the "client data" the signature back to the origin site (or equivalently, relying party). On receiving the "client data" and the signature, the relying party's first step, of course, is to verify that the signature matches the data as verified by the user's origin-specific public key. Assuming this matches, the relying party can examine the "client data" further to see if any MITM is present as follows:

- If "client data" shows that an incorrect origin name was seen by the user
  - an MITM is present
  - (albeit a sophisticated MITM which had also intermediated the registration and thus got the *Key Handle* issued by the U2F device to match the MITM's own origin name, and the MITM is now trying to intermediate an authentication. As noted earlier, an MITM intermediating only at authentication time and not at registration would fail since the U2F device would refuse to sign due to origin mismatch with the *Key Handle* relayed from the original origin by the MITM).

- else If “client data” shows a ChannelID OR origin used a ChannelID for the SSL connection:
    - If ChannelID in “client data” does not match the ChannelID the origin used, an MITM is present
    - (albeit a very sophisticated MITM which possesses an actual valid SSL cert for the origin and is thus indistinguishable from an “origin name” perspective)

It is still possible to MITM a user’s authentication to a site if the MITM is

a) able to get a server cert for the actual origin name issued by a valid CA, and
b) ChannelIDs are NOT supported by the browser.

But this is quite a high bar.

An MITM case which the U2F device does NOT protect against is as follows: Consider an online service or website which accepts plain password but allows users to self-register and step up to U2F 2nd factor. An MITM with a different origin which is present between the user and the actual site from the time of registration can register the U2F device on to itself and not pass this registration to the actual origin, which would still see the user as just needing a password. Later, for authentications, the MITM can accept the U2F device and just do an authentication with password to the actual origin.

Assuming the user does not notice the wrong (different) origin in the URL, the user would think they are logging in to the actual origin with strong authentication and are thus very secure but in reality, they are actually being MITMed.

7 Allowing for Inexpensive U2F Devices

A key goal of this program is to enable extremely inexpensive yet secure devices. To enable new secure element chips to be as inexpensive as possible it is important to allow them to have minimal or no onboard memory.

A U2F device allows for this. The *Key Handle* issued by the U2F device does not have to be an index to the private key stored on board the U2F device secure element chip. Instead, the *Key Handle* can "store" (i.e., contain) the private key for the origin and the hash of the origin encrypted with a "wrapping" key known only to the U2F device secure element. When the *Key Handle* goes back to the secure element it "unwraps" it to "retrieve" the private key and the origin that it was generated for.

As another alternative, the U2F device could store this "wrapped" information in a table in off-chip memory outside the secure element (which is presumably cheaper). This memory is still on board the U2F device. In this case, the *Key Handle* sent to the origin would be an index into this table in off-chip memory. As another possibility in the design spectrum, the *Key Handle* might only encode the origin and an index number, while the private key might still be kept on board – this would, of course, imply the number of keys is limited by the amount of memory.

8 Verifying That a U2F Device Is "genuine"

The U2F device protocol is open. However, for effective security, a U2F device has to be built to certain standards – for example, if the *Key Handle* contains private keys encrypted with some manufacturer specific method, this has to be certified as well implemented, ideally by some 'certification body' such as FIDO. In addition, the actual cryptographic engine (secure element) should ideally have some strong security properties.

With these considerations in mind, a relying party needs to able to identify the type of device it is speaking to in a strong way so that it can check against a database to see if that device type has the certification characteristics that particular relying party cares about. So, for example, a financial services site may choose to only accept hardware-backed U2F devices, while some other site may allow U2F devices implemented in software.

Every U2F device device has a shared "Attestation" key pair which is present on it – this key is shared across a large number of U2F device units made by the same vendor (this is to prevent individual identifiability of the U2F device). Every public key output by the U2F device during the registration step is signed with the attestation private key.

The intention is that the public keys of all the "Attestation" key pairs used by each vendor will be available in the public domain – this could be implemented by certificates chaining to a root public key or literally as a list. We will work within FIDO to decide the details on how certified vendors can publish their attestation public keys.

When such an infrastructure is available, a particular relying party – say, a bank – might choose to accept only U2F devices from certain vendors which have the appropriate published certifications. To enforce this policy, it can verify that the public key from a U2F device presented by the user is from a vendor it trusts.

In practice, for high quality U2F devices we expect that the attestation key would be burnt into the on-board secure element – the actual key to be burnt in would be provided by the vendor to the secure element manufacturer for every batch of chips, say about 100,000 units.

Note that the attestation key's presence only guarantees who the vendor is for a well built U2F device – it is one part of the story, albeit a very crucial part. As to whether the U2F device is indeed secure, that guarantee comes from certifications where third parties inspect the implementation by the vendor. In summary, attestation is a strong identifier of the certifications.

In this context, it's worth noting that a U2F device which stores keys on board rather than exporting them in the *Key Handle* are, in principle, most secure, since it is not vulnerable to any potential vendor specific vulnerabilities in the design of the encryption of the data in the *Key Handle*. However, a good design with an encrypted *Key Handle* will be well above the bar in security while also being cheaper.

At this time, the encryption used to embed private keys in the *Key Handle* are technically not part of the specified protocol. However, strong best practice guidelines are specified in the sample client side javacard applet available in U2F working group materials. It may be appropriate to include a review of particular implementations as part of a U2F certification within FIDO.

Note that it is still possible for a vendor to build a U2F compliant device which is not certified and whose attestation keys are **not** published in a “certification database”. A relying party could still choose to accept such devices – but it will do so with the full knowledge that that particular device type is not in the certification database.

8.1 Counters as a Signal for Detecting Cloned U2F Devices

The vendor attestation is one method by which an origin can assess a U2F device. In practice, we do not want to prevent other protocol compliant vendors, perhaps even those without any formal secure element, perhaps even completely software implementations. The problem with these non-secure-element based devices, of course, is that they could potentially be compromised and cloned.

The U2F device protocol incorporates a usage counter to allow the origin to detect problems in some circumstances. The U2F device remembers a count of the number of signature operations it has performed – either per key pair (if it has sufficient memory) or globally (if it has a memory constraint, this leaks some privacy across keys) or even something in between (e.g., buckets of keys sharing a counter, with a bit less privacy leakage). The U2F device sends the actual counter value back to the browser which relays it to the origin after every signing operation. The U2F device also concatenates the counter value on to the hash of the client data before signing so that the origin can strongly verify that the counter value was not tampered with (by the browser).

The server can compare the counter value that the U2F device sent it and compare it against the counter value it saw in earlier interactions with the same U2F device. If the counter value has moved backward, it signals that there is more than one U2F device with the same key pair for the origin (i.e., a clone of the U2F device has been created at some point).

The counter is a strong signal of cloning but cannot detect cloning in every case – for example, if the clone is only one which is used after the cloning operation and the original is never used, this case cannot be detected.

9 Client Malware Interactions with U2F Devices

As long as U2F devices can be accessed directly from user space on the client OS, it is possible for malware to create a keypair using a fake origin and exercise the U2F device. The U2F device will not be able to distinguish ‘good’ client software from ‘bad’ client software. On a similar note, it is possible for malware to relay requests from Client machine #1 to a U2F device attached to client machine #2 if the malware is running on both machines. This is conceptually no different from a shared communication channel between the Client machine (in this case #1) and the U2F device (which happens to be on machine #2). It is not in scope to protect against this situation.

Protection against malware becomes more possible if the U2F client is built into the OS system layer as opposed to running in user space. The OS can obtain exclusive access to U2F devices and enforce methods to ensure origin matches.

10 U2F Device User Experience

As described earlier access to the U2F device is manifested in two javascript functions available in the browser – one for creating a key pair and one for generating a signature. These are used by an origin online service or website to create a user flow.

10.1 Registration: Creating a Key Pair

The to-be-registered user is verified by the origin site (with username and password or whatever other means). The registration page rendered by the origin in the browser calls the javascript function for creating a key pair. When the javascript function is called, the user may see a browser infobar warning which he/she has to approve. After user approval, the key pair generation request is sent to every U2F device attached to the computer.

The first U2F device attached to the computer which has a positive "test of user presence" (i.e., the first attached U2F device on which the user presses the button) responds to this request. The browser packages the response from the U2F device (key handle, public key etc) and returns it to the web page as return results of the javascript function call. The registration web page sends these to the origin site and the origin sites stores this information indexed by the user's account to complete the registration process.

10.2 Authentication: Generating a Signature

The user starts the authenticaion process typically with username and password (or with just the username, if the site only wants a U2F device verification). The origin site renders an intermediate authentication page into which it sends the user's *Key Handle* and a nonce. It then calls the javascript function to create a signature. The parameters for the function call are the *Key Handle* and the nonce.

When the signature function is called, the browser may show an infobar asking for the user's approval (the user may choose to ask the browser to skip this in future). After the user's approval, the browser talks to all the U2F devices attached to the computer as described earlier and assembles their responses.

The javascript function call returns the "client data" object and the first signature response from a U2F device that replied. The intermediate authentication web page sends the "client data" and the U2F device responses on to the relying party, which determines if any of the signatures matches what it expects.

Note that depending on the U2F implementation multiple devices could reply for a particular *Key Handle*. For example, consider the case where the *Key Handle* is implemented purely as an index into memory on board the U2F device (and thus was just, say, a small integer). The user may have registered multiple U2F devices to a particular account on a particular origin and some of those devices could have used the same index integer as *Key Handle* for that particular account on that particular origin.

Note that though the user does not necessarily have to see the intermediate page described above. If the correct U2F device is present, then the signatures can be obtained and sent back to the origin and the authentication is completed. The user needs to see intermediate screens only for error conditions ("Please insert your U2F device", "We require you to activate your U2F device" etc).

11 U2F Device Usage Scenarios

Though the description so far has been in context of a particular user using a single device across multiple accounts, the usage scenarios enabled are broader.

11.1 Sharing a U2F Device Among Multiple Users

Note that a U2F device has no concept of a user – it only knows about issuing keys to origins. So a person and their spouse could share a U2F device and use it for their individual accounts on the same origin. Indeed, as far as the U2F device is concerned the case of two users having accounts on the same origin is indistinguishable from the case of the same user having two accounts on that origin.

Needless to say, the general case where multiple persons share a single U2F device and each person has accounts on whatever origins they choose is similarly supported in U2F.

11.2 Registering Multiple U2F Devices to the Same Account

U2F does not limit the user to have a single device registered on a particular account on a particular site. So for example, a user might have a U2F device mounted permanently on two different computers, where each U2F device is registered to the same account on a particular origin – thus allowing both computers to login securely to that particular origin.

If a user has registered multiple U2F devices to a particular account, then during authentication all the *Key Handles* are sent by the origin to the intermediate page. The intermediate page call the signature javascript function with the array of *Key Handles* and sends the aggregated response back to the origin. Each attached activated U2F device signs for those *Key Handles* in the array that it recognizes. The user authentication experience is unchanged.

As an optimization, note that when a origin detects a particular *Key Handle* is used successfully to authenticate from a particular browser, it can remember that *Key Handle* for future reference by setting a cookie on that browser and trying that *Key Handle* first before attempting other *Key Handle*s.

12 U2F Privacy Considerations: A Recap

As the reader would have noticed, user privacy is a fundamental design consideration for the U2F protocol. The various privacy related design points are reiterated here:

1. A U2F device does not have a global identifier visible across online services or websites.
2. A U2F device does not have a global identifier within a particular online service or website
    - Example 1: If a person loses their U2F device, the finder cannot "point it at a website" to see if some accounts get listed. The device simply does not know.
    - Example 2: If person A and B share a U2F device and they have each registered their accounts on site X with this device, there isn't any way for the site X to guess that the two accounts share a device based on the U2F protocol alone.
3. A key issued to a particular online service or website can only be exercised by that online service or website.
    - Since a key is essentially a strong identifier this means U2F does not give any signal which allows online services or websites to strongly cross-identify shared users.
4. A user has to activate the U2F device (i.e., "press the button") before it will issue a key pair (for registration) or sign a challenge.
5. The browser may notify the user before they form a U2F relationship with an online service or website
    - An infobar could appear whenever the "issue a key" javascript call is made.
    - An infobar (with a once-only option) could appear when the "sign with this key" javascript call is made for a particular origin

The infobar approach puts a decision burden on the users - this is a downside and the infobar UX design has to be done with care.

13 Other Privacy Related Issues

13.1 An Origin Can Discover that Two Accounts Share a U2F Device

The origin specific key issuance still leaves one possible privacy leak – which is the case where a person with a single U2F device uses it to generate keys to two separate accounts with the same origin. Say the two different accounts are associated with usernames u_1 and u_2 in the site's name space. Now when u_1 is attempting to authenticate, the origin can send down *KeyHandle_2* to the U2F device. If it returns a valid signature, it can infer that u_1 and u_2 belong to the same person or two persons who share the same computer who happen to have their U2F devices plugged in simultaneously. This is true even if the users have taken precautions to hide their client identity from the origin server (using an anonymizing proxy, incognito mode etc).

It is possible to enhance the U2F device specification to catch this case but it complicates the user experience and we chose not to do so. Users who are concerned about this line of attack would need to use different U2F devices for different accounts on the same site and plug in only the relevant U2F device and no other when initiating a session for a particular account.

13.2 Revoking a Key From an Origin

Say a user registers their U2F device on an online service or website which has unsavory practices without the user realizing that the online service or website is unsavory. Later the user wants to cut off association with that site. It should ideally be possible for the user to "delink" the key such that the U2F device starts behaving as if it no longer owns the key. Thus the site cannot strongly verify the user even if it can do social engineering to make the user click past warnings.

It is possible for a vendor to design a U2F device which can be "reset" – in that it stops honoring any key it has issued before the reset. This might mean the earlier *Key Handles* need to have a generation count and a reset makes the U2F device reject all keys older than the current generation count. Alternatively, if the U2F device uses a key wrapping mechanism, a "reset" could throw away the old wrapping key and replace it. This renders all earlier keys issued by the device useless, since the device can no longer make any sense of them.

However, if the secure element is stateless and has no hard reset ability, all this 'revocation' logic has to be implemented as blacklists in firmware outside the secure element (for eg, code on the USB intermediator). In such a case it is possible for a dedicated attacker (e.g., a spy service) to extract the secure element and verify if it indeed does work against keys it has issued in the past. One revocation safeguard available to the user is physical destruction of the U2F device – this could be useful in sensitive high value situations (e.g., a political dissident).

14 Considerations for Immediate Future: Non-USB Transports

As discussed earlier, USB based devices will be followed immediately by other transports which are becoming available widely for local communication – in specific, NFC and Bluetooth LE and built-in U2F devices.

If the communication between the computer (or more generally, the user's client device) and the device happens over an unencrypted public channel, there is potential for some degree of privacy leakage against a motivated dedicated eavesdropper. Specifically, every time a user logs in to a particular origin, the *Key Handle* for that origin and the hash of the origin are sent to the U2F device by the user's computer. Someone listening to this traffic over time can build a behavior profile of when a user (identified by the MAC address of the U2F device) logs into a particular site (identified by the origin hash). They could even identify which account is being used (as identified by the *Key Handle*).

For any transport which could be eavesdropped by a third party, we need to consider a transport level encryption between the user's computer and the U2F device. As a historical note, early versions of the U2F working draft protocols tried to lift this notion of "channel protection" into the protocol but this was taken out since it added too much weight relative to the benefit.

15 Expanding U2F to Non-browser Apps

The discussion above has been focused on the browser as the client side vehicle, with a Javascript API to talk to U2F devices. However, it is perfectly sensible to have app on a mobile OS such as Android talk to U2F devices over a system API.

When building a native system API, we still need a notion of “origin”. For example, if foo.com’s app mints a key on a particular U2F device, then bar.com’s app should not be able to exercise that key. Even more importantly, if the user uses the foo.com web app on a computer and foo.com’s app on a mobile device, the user needs to be able to use the same U2F device with both. This means that there has to be mechanism where the origin sent down to the U2F device by the browser for the foo.com web page matches the origin sent down to the U2F device by the mobile OS for the foo.com app.

This is achieved by specifying a level of indirection using the notion of an “application id”, which is a generalization of the origin concept. The “application id” is a publicly fetchable https URL where a particular origin (such as foo.com) lists its various “facets” – for example, it may list the hostname “www.foo.com” and the identifier for the signatures of foo.com’s android app. The application id https URL is assumed to be under the control of the origin – in other words, only it can change the list of “facets”.

The origin website or online service sends its “application id” down as a parameter to the U2F API on the web page. The browser fetches the content of the “application id” URL and ensures that the actual origin it sees for the web page calling the U2F API is indeed listed in the “facets” in the “application id” URL. For example, if a page served off www.foo.com makes a U2F API call, then this host name needs to be listed as a facet in the “application id” which is passed down. Similarly when a particular mobile app passes a “application id” to a U2F API on a mobile OS, the OS checks if the code signing signature of that particular app is listed as a facet in the “application id”. After these check if the “facet” is indeed in the “application id” as expected, the hash of the “application id” is sent down to the U2F device, rather than the hash of the “origin”. This ensures that foo.com’s web page and foo.com’s mobile app both are seen as the same site by the U2F device. As mentioned earlier, the “application id” is a generalized notion of an origin.