## Work in progress

This is an early draft of a chapter of a book on the foundations of cryptographic authentication being coauthored by Francisco Corella, Sukhi Chuhan and Veronica Wojnas. Please send comments to the authors.

# 10. FIDO and passkeys

## Chapter summary

FIDO2 is a technology developed and specified by the FIDO Alliance in collaboration with the W3C that provides returning user authentication by proof of possession of a private key, known as a passkey, with optional cryptographic attestation to the relying party that the passkey is stored in a certified authenticator that has verified a PIN and/or a biometric provided by the user. Passkeys can be stored in security keys, platform authenticators built into the operating system, roaming authenticators in mobile phones, and password managers or web browsers. Whereas passwords can be easily transferred between password managers, passkeys cannot be exported. To address the danger that users may be locked into a particular browser or password manager, the FIDO Alliance has initiated the specification of a credential exchange protocol that will allow passkeys to be securely transferred between password managers.

## 10.1 The FIDO Alliance

The FIDO Alliance [1] is an open industry association that was launched in 2013 and has been promoting the replacement of username and password for returning user authentication with cryptographic authentication using a key pair. To that purpose, the Alliance specifies cryptographic authentication protocols, known as FIDO protocols, and provides certification services.

Basic authentication to a website with a key pair can be trivially implemented using the Web Storage API and either the Web Crypto API or any of several JavaScript cryptographic libraries: when the user creates an account, the frontend of the site on the user's browser generates a key pair, stores the private key in localStorage, and registers the public key with the backend; then, when the user returns to the site, the frontend signs a challenge from the backend and submits the signature to the back end.

The value added by the FIDO Alliance is protection against an attacker who gains access to the user's browser. Today, laptops and mobile devices are routinely locked when unattended, but this was not the case in 2013. So, in the consumer space, the value added by the FIDO authentication protocols is less useful today. That may be one reason why, as stated in a candid white paper [2] published in March 2022, the Alliance hadn't "attained large-scale adoption of FIDO-based authentication in the consumer space". We shall come back to this crucial white paper in Section 10.3 when we discuss authenticators, and then again in Section 10.4, when we discuss passkeys.

However, whether or not FIDO is useful in consumer use cases, there are use cases where the value added by FIDO is not only useful but required. Employees with access to sensitive data may be required by their employers to protect the private key that they use to authenticate to the corporate network, especially if the corporation has a BYOD policy that allows employees to user their own devices. FIDO protocols support several kinds of cryptographic attestation, including untraceable Elliptic Curve Direct Anonymous Attestation (ECDAA) [3], that allows employees to demonstrate cryptographically that they are storing the private key in an authenticator certified by the FIDO Alliance.

The FIDO Alliance has so far specified three cryptographic authentication protocols , described in the next section: first, UAF and U2F, which the Alliance specified by itself, then FIDO2, developed and jointly specified in collaboration with the W3C, as we shall see in Section 10.2.3.

## 10.2 Protocols

### 10.2.1 UAF

UAF is an acronym for "Universal Authentication Framework", so-named because it specifies an architecture for designing authentication protocols rather than a particular protocol. The UAF architecture has three components:

- The UAF Authenticator, which runs on the user's device, stores the private key, and is equipped with a biometric sensor.
- The UAF Server, which registers the public key.
- The UAF Client, which orchestrates authentication by communicating with the Authenticator and the Server.

UAF does not specify how the Authenticator is implemented. A benefit of UAF is that it can leverage any biometric facility available on the user's device for authentication.

UAF does not specify how the Client is implemented either: it could be the browser, or a browser plugin, of a native app, or a component of the operating system of the user's device. UAF specifies an API between the Client and the Authenticator and an API between the Client and the Server. But these are not Application Programming Interfaces in the literal sense of the acronym, because UAF does not specify how the Client communicates with the Authenticator or the Server or what programming language should be used.

To authenticate, the Client forwards a challenge received from the Server to the Authenticator, and the Authenticator signs the challenge with the private key after authenticating the user with a biometric and/or a PIN. The UAF protocol is advertised as a passwordless authentication protocol, that replaces password authentication with biometric authentication, or with multi-factor authentication.

## 10.2.2 U2F

U2F is an acronym for "Universal 2nd Factor" and it is intended to be used in conjunction with authentication with username and password. It is stated in Yubico's website [4] that "U2F was developed by Yubico and Google, and contributed to the FIDO Alliance after it was successfully deployed for Google employees".

The private key is generated and used in a hardware device, called a security key, that is often implemented as a USB flash, that is often implemented as a USB flash drive but may also communicate with the user's device over BLE or NFC. However, the private key is not stored in the security key. Instead, it is encrypted, or wrapped, with a key encryption key (KEK), and the ciphertext is exported and used as the credential ID. The KEK never leaves the device. Not storing the private key allows a security key with limited storage to support an unlimited number of relying parties.

To authenticate, the JavaScript frontend of the relying party communicates with the security key through an API built into the browser or provided by a browser plugin, sending the credential ID and a challenge received from the backend. The user presses a button on the security key to demonstrate that a human is authorizing its use, then the security key uses the KEK to decrypt the credential ID and obtain the private key, signs the challenge with the private key, and returns the signature, which the frontend forwards to the backend of the relying party.

## 10.2.3 FIDO2

FIDO2 is an extension of U2F that incorporates two aspects of the UAF architecture. As in U2F, the JavaScript frontend of the relying party receives a challenge from the backend and sends it to the authenticator, which returns a signature on the challenge that the front-end submits the backend. However, instead of just pressing a button to authorize the signature, the user authenticates with a biometric and/or a PIN as in UAF. And although the authenticator can be a flash drive as in U2F, it can also be included in the user's device.

FIDO2 is sometimes characterized as making U2F passwordless, which is correct, since U2F was positioned as a second factor, while UAF was positioned as replacing password authentication with biometric authentication. This characterization, however, obscures the fact that all three protocols, UAF, U2F and FIDO2, are replacing password authentication with cryptographic authentication, something that has not been emphasized by the FIDO Alliance until the advent of the term "passkey".

FIDO2 consists of two distinct protocols, the Client To Authenticator Protocol v2 (CTAP2), specified by the FIDO Alliance, and the Web Authentication API (WebAuthn), specified by the W3C.

CTAP2 is an extension of the protocol that a browser or browser plugin that implements the U2F API uses to communicate with the security key, which is now called CTAP1.

CTAP2 is backwards compatible with CTAP1, and that makes it possible for WebAuthn to use U2F credentials. This backwards compatibility has recently led Google to deprecate the U2F client API [5]. But neither U2F nor UAF have been deprecated.

CTAP2 is used by the browser to interact with roaming authenticators, including security keys containing both FIDO2 and U2F credentials over USB, BLE or NF, and mobile phones over BLE or NFC.

WebAuthn is an extension of the W3C Credential Management API, and as such provides a function navigator.credentials.create() for creating a credential, and a function navigator.credentials.get() for authenticating with a credential.

The function navigator.credentials.create() takes inputs that specify, in particular:

- The type of attestation to be provided, if any.
- A challenge to be signed by an attestation private key.
- Information about the relying party.
- Information about the user, including user.id, meant to be a unique randomly generated identifier used by the relying party for its own purposes.

- Whether the authenticator must perform user verification.
- The kind of credential to be created; we shall come back to this input in Section 10.4.

The function navigator.credentials.get() takes inputs that specify, in particular:

- The challenge to be signed with the private key to authenticate the user.
- An input called allowCredentials comprising a list of credential IDs. We shall come back to this input in Section 10.4.

## 10.3 Types of authenticators

The support that the W3C provided to the FIDO Alliance by taking ownership of the WebAuthn specification motivated OS providers to implement platform authenticators that leverage the authentication mechanism used for unlocking the platform, using it for authorizing access to the authenticator. Thus, following the publication of WebAuthn Level 1 [6], there were two kinds of authenticators: security keys, which support both FIDO2 and U2F credentials, and platform authenticators built into the operating system, which support FIDO2 credentials.

However, the above-cited white paper mentioned as a reason for lack of adoption in the consumer space the problems that consumers faced with platform authenticators, specifically “having to re-enroll each new device”, and there being “no easy ways to recover from lost or stolen devices”. The white paper proposed two solutions to these problems: using a mobile phone as a roaming authenticator, and using “multi-device FIDO credentials”. The first solution was easy to implement, since CTAP2 already supported BLE and NFC, and it has already been implemented by native apps such as IDmelon [7].

As a way of implementing multi-device credentials, the white paper suggested syncing credentials between OS-based authenticators on different platforms where the OS was provided by the same OS vendor. This was not a practical solution, since syncing credentials between operating system instances is not something that is usually done. But browsers and passwords managers were already syncing passwords, and they found it easy to sync credentials, that came to be called “passkeys”, in addition to passwords.

So, four kinds of FIDO2 authenticators are now available: security keys, native apps on mobile devices, platform authenticators built into the OS, and authenticators implemented by browsers and password managers.

## 10.4 Passkeys

The term “passkey” is a clever portmanteau that for the first time draws attention to the fact that FIDO credentials are cryptographic credentials.

It was introduced by Apple in [8] without definition and has been used with a variety of connotations. The FIDO Alliance currently defines it in [9] as a synonym with FIDO credential for which no trademark is claimed. But it has also been used with a more specific connotation as referring to a *resident*, or *discoverable*, credential.

We saw in Section 10.2.2 that U2F credentials are not stored in the authenticator: they are instead encrypted under the KEK and the ciphertext is exported as the credential ID. FIDO 2 credentials are of two types. Ordinary ones are encrypted and exported as the credential ID, like U2F credentials, while discoverable credentials are stored in the clear.

Discoverable credentials provide two benefits but raise two issues.

One benefit is that authentication does not require the credential ID. When calling the function navigator.credentials.get(), the allowCredentials input is not required if credentials are discoverable. In response to the call, the function returns a signature on the challenge computed with the private key of a credential found in the authenticator. It also returns a userHandle, whose value is the user.id input that the frontend of the relying party provided to the function navigator.credentials.create() when the credential was created. The RP uses the userHandle to identify the user and autofill the username in what is sometimes called “conditional mediation”.

Another benefit is that discoverable credentials enable syncing. Ordinary credentials can only be used in the authenticator where they are created, because the KEK is needed to decrypt the credential ID, and the KEK never leaves the authenticator. Perhaps not coincidentally, Apple introduced the term “passkey” when they announced that they were going to sync credentials through iCloud.

One issue raised by discoverable credentials is that they take up space in the authenticator and may make it impossible for security keys to support substantial numbers of relying parties. In a call to navigator.credentials.get(), the RP can set the value of an input authenticatorSelection.requireResidentKey to true. If RPs do that in order to take advantage of the UI benefit provided by discoverable credentials, security keys will become useless.

The second issue raised by discoverable credentials is coupled with the benefit they enable syncing. While syncing passkeys seems equivalent to syncing passwords, there is an essential difference. The user can copy passwords from one password manager to another but cannot copy passkeys. If security keys become useless and passkeys cannot be transferred between password managers, FIDO technology may become a monopoly exploited by a few large browser vendors.

To address the second issue, the FIDO Alliance has initiated the specification of a credential exchange protocol that will allow passkeys to be securely transferred between authenticators [10].

## References

[1] Fido Alliance. Changing the Nature of Authentication. Retrieved from https://fidoalliance.org/overview/

[2] Fido Alliance. White Paper: Multi-Device FIDO Credentials. Retrieved from https://fidoalliance.org/white-paper-multi-device-fido-credentials/

[3] FIDO Alliance. FIDO ECDAA Algorithm. Retrieved from https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-ecdaa-algorithm-v2.0-id-20180227.html

[4] Yubico. What is FIDO Universal 2nd Factor? Retrieved from https://www.yubico.com/resources/glossary/fido-u2f/

[5] Kukreja, C. L. (2024, July 18). Deprecation of U2F API & backward compatibility offered by WebAuthn API. Retrieved from https://medium.com/@chunilalkukreja/depreciation-of-u2f-api-backward-compatibility-offered-by-webauthn-api-a802d7d8ceb7

[6] W3C. (2019, March 4). Web Authentication: An API for accessing Public Key Credentials. Retrieved from https://www.w3.org/TR/webauthn-1/

[7] IDmelon. (n.d.). IDmelon Reader. Retrieved from https://idmelon.com/products/reader

[8] Mike Peterson. (2022, June 7). Apple passkey feature will be our first taste of a truly password-less future. Retrieved from https://appleinsider.com/articles/22/06/07/apple-passkey-feature-will-be-our-first-taste-of-a-truly-password-less-future

[9] FIDO Alliance. *Passkeys.* Retrieved from https://fidoalliance.org/passkeys/#faq.

[10] Nick Steele. (2024, October 14). Coming soon: Securely import and export passkeys. Retrieved from https://blog.1password.com/fido-alliance-import-export-passkeys-draft-specs/