Future-Proof Web Authentication: Bring Your Own FIDO2 Extensions

Florentin Putz, Steffen Schön, and Matthias Hollick

TU Darmstadt, Germany
{fputz,sschoen,mhollick}@seemoo.de

Abstract. The FIDO2 standards for strong authentication on the Internet define an extension interface, which allows them to flexibly adapt to future use cases. The domain of establishing new FIDO2 extensions, however, is currently limited to web browser developers and members of the FIDO alliance. We show how researchers and developers can design and implement their own extensions for using FIDO2 as a well-established and secure foundation to demonstrate innovative authentication concepts or to support custom deployments. Our open-source implementation targets the full FIDO2 stack, such as the Chromium web browser and hardware tokens, to enable tailor-made authentication based on the power of the existing FIDO2 ecosystem. To give an overview of existing extensions, we survey all published FIDO2 extensions by manually inspecting the source code of major web browsers and authenticators. Their current design, however, hinders the implementation of custom extensions, and they only support a limited number of extensions out of the box. We discuss weaknesses of current implementations and identify the lack of extension pass-through as a major limitation in current FIDO2 clients.

Keywords: Security · Authentication · Key Management · Hardware Token · Passwordless · WebAuthn

1 Introduction

Stronger forms of authentication than passwords can protect online accounts from phishing attacks and mitigate the impact of data breaches. The FIDO2 standards [8] are now implemented in all major web browsers and allow users to securely log in to websites without passwords, using either a hardware security key (e.g., a YubiKey [28]) or a built-in authenticator in modern smartphones or laptops. Website operators can easily implement support for FIDO2 using the WebAuthn JavaScript API [25], which also supports custom extensions to implement special use cases.

Extensible web standards such as X.509 [2] or TLS [3] have played a key role during the advancement of the Internet so far. A standards organization such as the IETF cannot predict all possible future use cases of the standard beforehand, but custom extensions provide flexibility to the otherwise fixed specification and prevent fragmentation via the creation of new competing standards to satisfy new use cases. The behavior of a protocol can then be modified without the effort to get the modifications accepted into the main standard.

Why extend FIDO2? At its core, FIDO2 is a well-established and secure platform for accessing public-key credentials, with little restrictions imposed on the authenticator. In the future, however, innovative authentication approaches might require modifications to the standard, e.g., by transmitting or receiving additional information to the authenticator. FIDO2 extensions can be used to support new uses cases, implement additional features, and mitigate shortcomings of the standard.

The potential of FIDO2 extensions becomes clear by looking at the extensions that have been proposed so far: A major weakness of FIDO2 is the lack of efficient recovery options, which has been criticized in previous user studies [15,5,1]. Yubico proposed a new FIDO2 extension to address this problem, by implementing an efficient way to automatically register backup credentials, which can be used in case the authentication token gets lost [30].

Developers and researchers can also use FIDO2 extensions to prototype and demonstrate new authentication designs within the existing FIDO2 ecosystem on real-world web browsers. Companies can use FIDO2 extensions to adapt the protocol for tailor-made authentication in internal deployments.

Web browsers, however, currently do not support any third-party FIDO2 extensions. Furthermore, there is no clear development path to implement custom extensions in current web browsers. Even FIDO2 extensions which do not require any special processing by the web browser need explicit browser support instead of automatically being forwarded to the authenticator. Although extensions are an important part of any standard, the extensibility of the FIDO2 standards unfortunately has not received much attention as of today. This calls for a detailed analysis of FIDO2 extensions:

– As our core contribution, we show how to design and implement tailor-made authentication based on the power of the existing FIDO2 ecosystem (Section 5). Our source code and additional documentation is available at https://seemoo.de/s/fido2ext.
– We survey all publicly known FIDO2 extensions by manually inspecting the source code of major web browsers and authenticators. (Section 3 and Section 4).
– We describe limitations of current FIDO2 extension implementations and identify the lack of extension pass-through as a major weakness that inhibits the development of innovative FIDO2 extensions (Section 6).

2 Background

2.1 FIDO2

Strong and passwordless authentication has been standardized by the FIDO industry alliance and the World Wide Web Consortium (W3C) in form of the

FIDO2 standards [8]. The top part of Figure 1 shows the FIDO2 system model, consisting of relying party (RP), client, and authenticator. In essence, FIDO2 is a challenge-response protocol, which RPs such as websites can use to access public key credentials, managed for the user by an authenticator. The authenticator can be embedded into the client device or attached as an external hardware token via USB, NFC, or Bluetooth. It mainly supports the following two operations:

1. `authenticatorMakeCredential`: The authenticator generates a new key pair, binds it to the requesting RP, and returns the public key and optionally an attestation signature. Websites use this operation to implement account registration.
2. `authenticatorGetAssertion`: The authenticator returns an assertion signature over a challenge from the RP. Websites use this operation to implement account login.

The client (e.g., a web browser) forwards the RP's requests to the authenticator, which stores all key material. FIDO2 consists of two specifications, WebAuthn (Web Authentication) [25] and CTAP (Client to Authenticator Protocol) [6], which succeed the older Universal 2nd Factor (U2F) standard [7]. All major web browsers implement FIDO2 [17] and there exists a large ecosystem of commercial hardware tokens, e.g., Yubico's YubiKey [28].

2.2 FIDO2 Extensions

The FIDO2 authentication protocols WebAuthn and CTAP2 can be extended to support special use cases. Figure 1 shows the general protocol flow of a FIDO2 extension, as specified by §9 of the WebAuthn specification [25]. The RP initiates an extension request with the client, which then processes the request and responds to the RP. WebAuthn distinguishes between different types of extensions. An extension is always a *client extension*, as it involves communication with the client. The extension specification defines the form of the JSON *client extension input/output* and what kind of *client extension processing* the client needs to perform to transform the inputs to the outputs.

If the extension also involves processing on the authenticator, it is additionally an *authenticator extension* and defines the form of the concise binary object representation (CBOR) *authenticator extension input/output* and what kind of *authenticator extension processing* the authenticator needs to perform to transform the inputs to the outputs. In this case, the client also needs to know how to convert the JSON *client extension input* to the CBOR *authenticator extension input*, and how to convert the CBOR *authenticator extension output* back to the JSON *client extension output*.

FIDO2 extensions also are *registration extensions* and/or *authentication extensions*, depending on which part of the protocol they affect. We describe further implementation details in Section 5.

Each FIDO2 extension has an extension identifier string, which can be registered in the IANA "WebAuthn Extension Identifiers" registry [12]. This registry

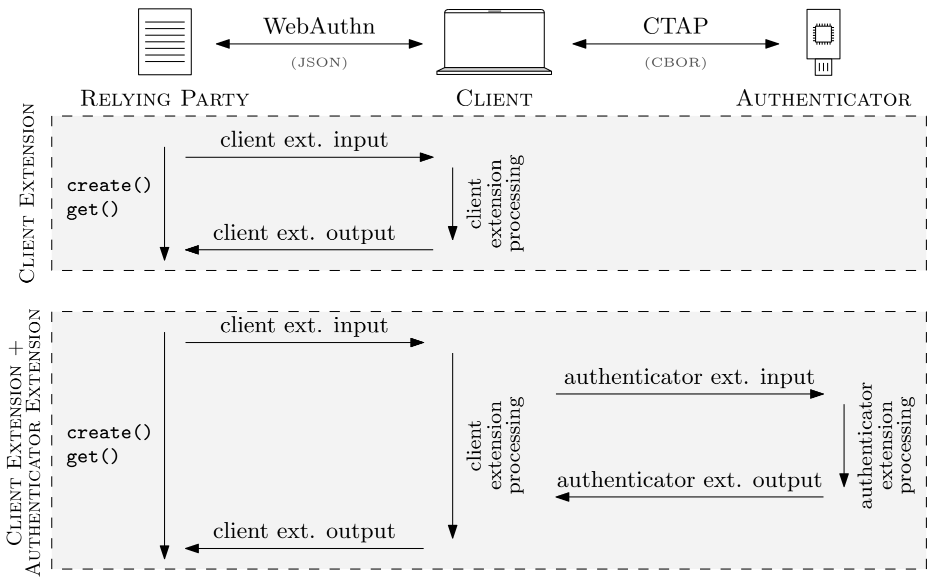

Fig. 1. Protocol flow of FIDO2 extensions. Extensions can communicate between RP, client, and authenticator *(top)*. An extension is always a client extension *(middle)* but can also additionally be an authenticator extension *(bottom)*.

has been defined and established by RFC 8809 [11] and contains an up-to-date list of all currently registered WebAuthn extension. We describe all currently available FIDO2 extensions in Section 3.

Authenticators supporting CTAP 2.1 must implement the `hmac-secret` extension and the `credProtect` extension if they support some form of user verification (§9 in CTAP2 [6]). Clients can use the CTAP `authenticatorGetInfo` method to detect which extensions an authenticator supports. We give an overview of which FIDO2 extensions are supported by popular browsers and authenticators in Section 4.

2.3 Extension Pass-Through

In general, clients and authenticators do not need to support any WebAuthn extensions and can simply ignore them as they are all optional. RPs must therefore be prepared to deal with any subset of requested extensions being ignored, as this must never fail the WebAuthn transaction. To increase compatibility with unknown extensions, clients can choose to directly pass through extension inputs to the authenticator and outputs to the RP, without any additional client processing. The WebAuthn standard defines conversion rules between JSON and CBOR to facilitate this [26]. Authenticators must be prepared to ignore such inputs in case such a direct pass-through results in invalid inputs.

To increase compatibility, extensions that do not require special client processing should define the authenticator input and outputs in such a way that direct pass-through results in semantically correct values. Clients that implement pass-through facilitate innovation as they allow extensions to work when only the authenticator explicitly supports them.

3 Survey: Existing FIDO2 Extensions

Our survey on FIDO2 extensions consists of two parts: (1) This section is the first part, where we identify all extensions that have been published so far. (2) Section 4 is the second part, where we determine which clients and authenticators support these extensions. Table 1 shows the results of our survey.

Most extensions are specified in the FIDO2 standards themselves ("standard extensions"), such as the `hmac-secret` extension for using a symmetric key with a FIDO2 credential [6]. Additionally, there are three non-standard extensions which are not specified in the official FIDO2 standards. First, there is Yubico's `recovery` extension [30], which has a separate specification but no web browser implementation yet. Second, there is Google's `caBLE` extension [19], which is part of the Google Chrome browser but has no public specification. Third, there is the `googleLegacyAppIdSupport` extension [13], which is implemented in some browsers but has no formal specification yet. We briefly introduce the standard extensions and then go over the non-standard extensions in more detail, as they are currently less well-known.

3.1 Standard Extensions

We refer to the extensions defined in the FIDO2 standards as *standard extensions*. At the time of writing, the IANA registry [12] contains eight registered extension identifiers, which are all specified in WebAuthn Level 1 [24]. WebAuthn Level 2 defines three additional extensions and also plans to register them (§12.4 in [25]). Furthermore, CTAP 2.1 defines five extensions and plans to register them as well (§14.1 in [6]). One of them is the `hmac-secret` extension, which allows requesting a symmetric secret from an authenticator to encrypt and decrypt user data. This extension can support new use cases where a static secret is necessary, such as disk encryption or unlocking password managers like KeePass. The other standard extensions enable features such as compatibility with legacy Universal 2nd Factor (U2F) credentials, specifying a credential protection policy, or storing arbitrary data associated with a credential. The FIDO2 standards contain further information on each extension as well as a formal specification.

3.2 Recovery Extension (Yubico)

Recovery and backups still are open problems within the FIDO2 standards, as the current best practice of manually registering multiple tokens on each RP has low usability. Yubico aims to solve this problem using the `recovery` extension, which currently has a draft specification available online [30]. This extension allows automatically registering a backup authenticator for recovery purposes without needing to have the backup authenticator physically available at the time of registration, so that it can be permanently stored at a secure location. Their proposal is based on a key agreement scheme where the primary authenticator generates nondeterministic public keys, but only the backup authenticator can derive the corresponding private keys. This enables a usable recovery process while maintaining the FIDO2 privacy protections of unlinkable public keys. Frymann et al. generalized the procedure to Asynchronous Remote Key Generation (ARKG) and proved the cryptographic security of such protocols [9].

3.3 CaBLE Extension (Google)

Google proposed caBLE (Cloud-Assisted Bluetooth Low Energy), which allows using a smartphone as a FIDO2 roaming authenticator via BLE [19,20]. The caBLE protocol consists of two phases: A pairing phase using a QR code, which encodes a nonce to derive key material. Afterwards, the smartphone broadcasts advertisements via BLE, which the web browser recognizes to initiate a handshake phase to establish a CTAP2 channel. There is no public specification available for caBLE and it is currently implemented only in the Chrome web browser and in Android smartphones via the proprietary Google Play Services. The Chromium implementation hints at a WebAuthn registration extension with client input `cableRegistration` and an authentication extension with client input `cableAuthentication`. The communication with the smartphone is not implemented using the standard FIDO2 extension API, but using a custom CTAP2 transport instead.

3.4 GoogleLegacyAppIdSupport Extension (Google)

Google proposed the `googleLegacyAppIdSupport` extension [13] to provide compatibility with some Android factory images, which only support the U2F JavaScript API and cannot be patched with WebAuthn support. Google Chrome plans to deprecate the U2F API, but credentials created using WebAuthn are normally not backwards compatible with the U2F JavaScript API. This extension allows creating WebAuthn credentials which work with both WebAuthn and U2F, by using a hard-coded U2F `AppID` specific to Google Accounts, limited to `*.google.com` domains.

4 Survey: Compatibility of FIDO2 Extensions

While the first part of our survey identified all currently known FIDO2 extensions, this section contains the second part of our survey, where we determine the client and authenticator support for all FIDO2 extensions. Since it is standard compliant for a client to ignore any extensions, the currently available feature trackers for web browser WebAuthn implementations do not indicate support

Table 1. FIDO2 extension compatibility as of July 2021.

Identifier	Chrome	Edge	Firefox	Safari	libfido2	python-fido2	Windows Hello	YubiKey 5	SoloKey	OpenSK	Specification	Reference	IANA Registration?
`appid`	✓	✓	✓	✓	✗	✓	-	-	-	-	WebAuthn2 §10.1	[25]	✓
`appidExclude`	✓	✓	✗	✗	✗	✗	-	-	-	-	WebAuthn2 §10.2	[25]	✗
`authnSel`	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	WebAuthn1 §10.4	[24]	✓
`biometricPerfBounds`	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	WebAuthn1 §10.9	[24]	✗
`credBlob`	✓	✓	✗	✗	✓	✓	✓	✗	✗	✗	CTAP2 §12.2	[6]	✗
`credProps`	✓	✓	✗	✗	✗	✗	-	-	-	-	WebAuthn2 §10.4	[25]	✗
`credProtect`	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓	CTAP2 §12.1	[6]	✗
`exts`	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	WebAuthn1 §10.5	[24]	✓
`hmac-secret`	✓	✓	✓	✗	✓	✓	✗	✓	✓	✓	CTAP2 §12.5	[6]	✗
`largeBlob`	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	WebAuthn1 §10.5	[24]	✗
`largeBlobKey`	✓	✓	✗	✗	✓	✓	✗	✗	✗	✗	CTAP2 §12.3	[6]	✗
`loc`	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	WebAuthn1 §10.7	[24]	✓
`minPinLength`	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	CTAP2 §12.4	[6]	✗
`txAuthSimple`	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	WebAuthn1 §10.2	[24]	✓
`txAuthGeneric`	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	WebAuthn1 §10.3	[24]	✓
`uvi`	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	WebAuthn1 §10.6	[24]	✓
`uvm`	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	WebAuthn2 §10.3	[25]	✓
`caBLE`	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗		[19]	✗
`googleLegacyAppIdSupport`	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗		[13]	✗
`recovery`	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗		[30]	✗

for individual extensions. Table 1 gives an overview of FIDO2 extension support for popular web browsers, client libraries, and authenticators. In the following, we explain our methodology and results in more detail.

4.1 Web Browsers

There is no automatic way to query all extensions supported by a web browser, so we need to manually inspect each browser's source code to see which FIDO2 extensions they implement. We checked the Chromium source code[1] (which forms the base of Google's Chrome browser and Microsoft's Edge browser), the Gecko source code[2] (which corresponds to Mozilla's Firefox browser), and the WebKit source code[3] (which forms the base of Apple's Safari browser). Table 1 shows

[1] Chromium version *canary 93.0.4570.0* → Google Chrome 93, Microsoft Edge 93.
[2] Gecko version version *nightly 91.0a1* → Mozilla Firefox 91.
[3] WebKit version *611.2.7.1* → Apple Safari 14.1.1.

all WebAuthn and CTAP2 extensions currently supported by these browsers. Chrome supports 11 FIDO2 extensions, including the non-standard caBLE extension, while Firefox and Safari only support two extensions each. All browsers support the `appid` extensions for compatibility with legacy U2F credentials.

Extension pass-through is an important feature that allows FIDO2 clients to transparently forward extension inputs and outputs between RP and authenticator, even for unknown extensions. The WebAuthn specification notes that extension pass-through can "facilitate innovation, allowing authenticators to experiment with new extensions and RPs to use them before there is explicit support for them in clients." [26]. While inspecting the web browser source codes, however, we noticed that no web browser currently supports extension pass-through. This makes it very difficult to establish custom extensions, as users will not be able to use them.

4.2 Client Libraries

Yubico's `python-fido2` library [27] can be used to implement custom FIDO2 clients that interface with CTAP authenticators. It currently supports six FIDO2 extensions as of version 0.9.1. This library is especially interesting for researchers and developers as it provides an easy interface for implementing custom CTAP2 extensions. Yubico also provides the C library `libfido2` [29] for interacting with FIDO2 authenticators, with bindings available to many other languages. This library supports four different FIDO2 extensions as of version 1.7.0.

4.3 Authenticators

To query the extensions supported by different FIDO2 authenticators, we use the `authenticatorGetInfo` CTAP2 request to retrieve a list of supported extensions in the `extensions` response field. As not all extensions are specified to announce themselves this way, we also manually inspect the authenticator's source code if available. We looked at Yubico's YubiKey 5 NFC, YubiKey 5C NFC, the open source Solo[4] C implementation (which powers the SoloKey, the NitroKey, and the OnlyKey), and Google's open source OpenSK[5] authenticator implementation written in Rust. All tokens only support the `credProtect` and the `hmac-secret` extensions.

In addition to these roaming authenticators, it is also possible to use platform authenticators. Microsoft provides Win32 headers for communicating with the Windows Hello platform authenticator and with roaming authenticators in general [18]. The actual implementation of these APIs is proprietary, but developers can use these headers to support FIDO2 authentication on Windows. As of `WEBAUTHN_API_VERSION_3`, Windows implements three FIDO2 extensions. It also partially implements `hmac-secret`, but obtaining the secret is not supported yet.

[4] Solo version 4.1.2.
[5] OpenSK version 1.0.0.

4.4 Summary

The survey in the last two sections shows that FIDO2 extensions have the potential to solve major weaknesses of the FIDO2 standards, such as the lack of efficient recovery options. Support of these extensions by web browsers, clients, and authenticators is currently weak, however, as shown in Table 1. Some extensions defined in the FIDO2 standards have no support at all. Extension pass-through would improve the compatibility of FIDO2 clients with current and future FIDO2 extensions, which would facilitate researchers and developers to create their own extensions, to add new features or to solve shortcomings of FIDO2.

5 Design and Implementation of Custom Extensions

In order to showcase the possibilities of FIDO2 extensions and the required steps to implement one, we define a proof-of-concept extension that we implement on all parts of the FIDO2 stack: RPs (web applications), clients (browser and non-browser), and authenticators. Implementing a custom FIDO2 extension is not straightforward and often undocumented, so we try to facilitate the process for other researchers and developers. We start by defining our extension and then describe the process of implementing this extension for each affected component. Our source code as well as supplemental material is available online[6].

There are two primary requirements for our proof-of-concept extension. First, it should use all capabilities that FIDO2 extensions offer, namely inputs, outputs, and usage in both registration (`makeCredential`) and authentication (`getAssertion`). Second, it should be simple in order to allow readers to focus on the implementation of an extension in general instead of the functionality of our example. Thus we choose to define and implement the `greeter` extension, which implements a concept that is well known in communications.

Definition 1 (Greeter Extension). *The `greeter` extension is an authenticator extension, which allows the authenticator to respond to the RP with a greeting message. We define an input string, which is the name of the sender (e.g., "John"). We also define an output string, which is the greeting message (e.g., "Hello John"). This extension requires no special client extension processing, as the client can simply pass through all inputs/outputs to/from the authenticator. The authenticator extension processing consists of constructing the greeting message from the input string and returning it. Our greeter extension uses the identifier `greeter` in both CTAP and WebAuthn.*

5.1 Relying Party

The RP initiates the FIDO2 authentication protocol and can include extensions in its requests. To demonstrate this, we implement our FIDO2 extension in a proof-of-concept web application, which uses the native browser

[6] Online repository with our source code and additional documentation on how to implement your own FIDO2 extensions: https://seemoo.de/s/fido2ext

JavaScript WebAuthn APIs supported by all modern browsers [16]. The process of using extensions for registration (`makeCredential`) and authentication (`getAssertion`) is very similar. To register a credential using the browser API, we use `navigator.credentials.create()`. This function takes a parameter object with the `publicKey` member in case of WebAuthn.

The `publicKey` object accepts an `extensions` attribute that contains the extension identifier including inputs: `extensions: {"greeter": "John"}`. In this case we do not use a backend to generate the challenge and various other values. In practice, those values should not be computed on the client side. After the extension was successfully processed by browser and authenticator, the outputs of the extension can be accessed using `credential.getClientExtensionResults()`, which returns an object with the extension identifiers as keys and the respective extension outputs as values.

5.2 Web Browser

Web browsers are the most common FIDO2 client and are next in the FIDO2 stack. They receive the authentication request from the RP and communicate with the authenticator. As web browsers do not currently support FIDO2 extension pass-through, we need to modify the browser to support our custom extension. We have implemented the `greeter` extension in the Chromium browser, as it is the foundation of the most popular web browser on the market, Google Chrome. Unfortunately, Chromium does not implement FIDO2 extension support in a modular way, which means that we need to modify different components of the browser stack in order to implement support for our `greeter` extension. Chromium does not provide any documentation for this, so we had to find out the correct approach by inspecting the implementation of other FIDO2 extensions in the source code.

Figure 2 shows how Chromium processes and sends FIDO2 extensions to authenticators. Let us assume a web application with the JavaScript code described in the previous section. Chromium's rendering engine Blink contains the V8 JavaScript engine, which runs this code and dispatches the command `navigator.credentials.create()` to the corresponding Blink module implementing this Web Platform API. This command is part of the general Credential Management API[7], which gets called first. This gets treated as a WebAuthn request in `CredentialsContainer::create()`, because we specified `publicKey` in the code above.

The JavaScript request contains a parameter object following the `PublicKeyCredentialCreationOptions` WebIDL specification. This object also contains the client extension inputs according to `AuthenticatorExtensionsClientInputs`, which we need to modify to include our own extension input. In Chromium, these parameters from JavaScript get converted into Mojo structures in `authenticator.mojom`[8],

[7] Chromium Credential Management API *(Blink)*: `third_party/blink/renderer/modules/credentialmanager/`

[8] Chromium WebAuthn Mojo *(Blink)*: `third_party/blink/public/mojom/webauthn/`

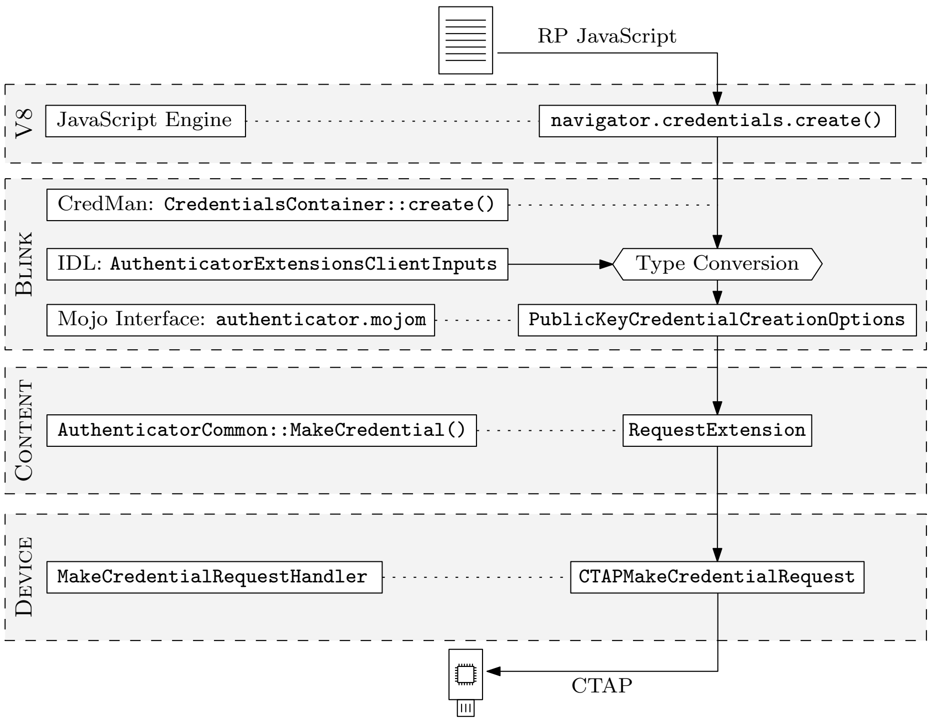
RP JavaScript
V8
JavaScript Engine
navigator.credentials.create()
Blink
CredMan: CredentialsContainer::create()
IDL: AuthenticatorExtensionsClientInputs
Type Conversion
Mojo Interface: authenticator.mojom
PublicKeyCredentialCreationOptions
Content
AuthenticatorCommon::MakeCredential()
RequestExtension
Device
MakeCredentialRequestHandler
CTAPMakeCredentialRequest
CTAP

Fig. 2. Chromium's FIDO2 client implementation, showing the data and processing flow for sending extensions to an authenticator.

which are used for IPC between different parts of Chromium. In our case, the relevant Mojo structure is `PublicKeyCredentialCreationOptions`. We need to change both the Mojo structure and the converter from IDL to Mojo. Afterwards, the call arrives at `AuthenticatorCommon::MakeCredential()` inside Chromium's `content` layer[9]. We need to modify this function to include our extension as a `RequestExtension`. This function then calls the `MakeCredentialRequestHandler` inside Chromium's `device` layer[10]. We modify the `CTAPMakeCredentialRequest`, which converts our request into a CTAP message. Afterwards, this gets sent to the authenticator, which responds back to us.

The response path is similar and traverses back to the `content` layer and finally to the `Blink` layer, where the client extension output gets converted into the WebIDL `AuthenticationExtensionsClientOutputs` and passed back to the V8 JavaScript engine.

[9] Chromium Web Authentication *(content)*: `content/browser/webauth/`
[10] Chromium CTAP *(device)*: `device/fido/`

5.3 Client Library

Besides browsers, it is also possible to implement a FIDO2 client in desktop applications. The `python-fido2` library [27] by Yubico provides the client functionality to communicate with an authenticator using CTAP2 in Python. Although not officially documented, the library can be extended with custom FIDO2 extensions due to its modular design. We have implemented our custom extension by inheriting from the `Ctap2Extension` class, which is located in `fido2/ctap2/extensions.py`. There are two abstract methods, `process_create_input` and `process_create_output`, which need to be implemented in order to use extension inputs and outputs. The subclass does not have to be registered anywhere, as `Fido2Client` considers all subclasses of `Ctap2Extension` when parsing a request or response.

5.4 Authenticator

The final component involved in FIDO2 is the authenticator. When implementing a FIDO2 extension, there are a number of open source hardware security token implementations available that we can build on:

- SoloKeys Solo 1 [21]
- SoloKeys Solo 2 [23]
- Google OpenSK [10]

While Solo 2 is still in early development and thus might not be suitable yet, Solo 1 and OpenSK are in a state that can be used to implement custom extensions. We have implemented the `greeter` extension in Solo 1, but describe the process for both Solo 1 and OpenSK. Neither project has a modular software architecture for implementing FIDO2 extensions, as all extensions are hard-coded at different code locations across the CTAP stack. Solo 1, however, provides documentation on how to implement custom extensions [22]. For OpenSK, we figured out the correct approach by inspecting the implementation of other FIDO2 extensions in the source code. In general, when implementing an extension for an authenticator, three modifications are required independent of the CTAP implementation:

- Announcing the support for the extension in the CTAP 2 `getInfo` function.
  - In Solo: `ctap_get_info()`.
  - In OpenSK: `process_get_info()`.
- Parsing and processing the extension inputs.
  - In Solo: `struct CTAP_extensions`, `ctap_parse_extensions()`.
  - In OpenSK: `struct MakeCredentialExtensions`, `MakeCredentialExtensions::try_from()`, `create_make_credential_parameters_with_cred_protect_policy()`.
- Building the extension outputs.
  - In Solo: `ctap_make_extensions()`.
  - In OpenSK: `process_make_credential()`.

6 Discussion

6.1 Extension Pass-Through

In the previous sections, we showed that FIDO2 extensions can address shortcomings of the current FIDO2 standards and allow them to easily adapt to new use cases. In practice, the applicability of FIDO2 extensions is limited due to browser design decisions. In order to facilitate the development and usage of innovative FIDO2 extensions, we argue that web browsers should pass-through unknown extensions to authenticators, as the WebAuthn specification suggests [26].

Some custom FIDO2 extensions might pose the risk of exposing too much fine-grained data to RPs, which would allow them to discriminate between authenticators. This is the reason why Chromium does not support the official `uvi` and `uvm` extensions [14]. This potential problem can be approached by letting the user opt-in to pass-through custom FIDO2 extensions, e.g., via a configuration option in `chrome://settings/securityKeys` or via the permission API that is already used for camera or microphone access.

6.2 Supporting FIDO2 Extensions via Browser Extensions

Apart from supporting pass-through, web browsers can also facilitate the development of innovative FIDO2 extensions by making it easier to implement them. As we have shown before, Chromium's support for FIDO2 extensions is currently hard-coded in many different parts of the browser stack. This could be improved using a modular architecture. Another interesting approach would be to allow implementing custom FIDO2 extensions as regular web browser extensions. Chromium already has an extensive web extensions API, which could be extended with an API for FIDO2 extensions. This would also facilitate the use case of internal company deployments, as it is already possible in an enterprise setting to provision web browsers with a set of extensions.

6.3 Outlook

Without support for custom FIDO2 extensions in web browsers, it is unlikely that FIDO2 extensions will be utilized to a similar extent than X.509 or TLS extensions (see Section 7). The domain of developing new FIDO2 extensions currently is rather limited to developers of web browsers or members of the FIDO Alliance, like Google is already doing with their caBLE extension. Third-party developers have difficulties of implementing and testing their custom FIDO2 extensions in web browsers. In practice, the primary use case of custom extensions is probably not the public Internet but rather internal deployments with a limited target audience. In this case, the argument of possible authenticator discrimination is not applicable anymore as these internal deployments already have custom authentication policies in place.

The previously mentioned restriction on the usage of custom extension might also explain the limited support for implementing custom extensions in FIDO2 authenticators and clients. Documentation is often limited, if available at all. The software architectures are not designed to be modular with regards to FIDO2 extensions. One instance of a well designed modular extension architecture is the `python-fido2` client library, although it still requires modification of the internal source code and lacks public documentation.

7 Related Work

FIDO2 is not the first protocol that allows the definition of extensions in order to extend the capabilities of the specified protocol. There are a number of other protocols or data formats that support extensions and successfully integrate them in real world usage, two of which we describe in more detail:

- **X.509 Certificates:** In the context of public key infrastructures, the use of X.509 [2] certificates is very common. X.509 allows the inclusion of additional fields in certificates, revocation lists, etc. through extensions. In practice, these extensions are heavily used to address issues that cannot be solved using the specification alone. An example is the subject alternative name extension, which allows the declaration of additional identities of the certificate subject, as the basic certificate only allows a common name.
- **TLS:** The transport layer security protocol provides a secure communication layer for otherwise insecure protocols such as HTTP. Since TLS 1.2 [3], additional data can be communicated in the handshake of the protocol using extensions [4]. These can, for example, be used to let clients specify the domain of the requested resources in addition to the IP address using the server name indication extension. In the modern web, servers often do not only serve a single domain but a number of domains. If different certificates are used for the different domains, the server cannot know which certificate to include in the response without the server name indication extension.

In summary, extensible protocols are easier to adapt to new use cases, which prevents fragmentation. This is especially important for authentication protocols such as FIDO2, which so far have not managed to get widespread adoption to replace passwords.

8 Conclusion

In this paper, we showed that the extensibility of the FIDO2 standards is an important feature to remain flexible and future-proof. We demonstrated how to design and implement new FIDO2 extensions in web browsers, client libraries, and authenticators. This allows researchers and developers to use the existing FIDO2 infrastructure as a secure foundation for custom authentication deployments or for demonstrating new authentication approaches with real-world browsers and hardware. We surveyed all existing FIDO2 extensions and determined their respective support by web browsers and authenticators. A small number of extensions have widespread support, but most extensions have only minimal support or no support at all.

Current FIDO2 client implementations limit FIDO2's extensibility due to missing features and complex software architectures. We also showed that web browsers should (1) implement a configuration option that enables FIDO2 extension pass-through, and (2) consider extending the regular web browser extensions API to allow implementing custom FIDO2 extensions as, e.g., Chrome extensions. Both steps would empower researchers and developers to create their own extensions.

Acknowledgements. This work has been funded by the LOEWE initiative (Hesse, Germany) within the emergenCITY center. We thank the anonymous reviewers for reviewing this paper and for their helpful comments.

References

1. Ciolino, S., Parkin, S., Dunphy, P.: Of two minds about two-factor: Understanding everyday FIDO U2F usability through device comparison and experience sampling. In: Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019). pp. 339–356. USENIX Association, Santa Clara, CA (Aug 2019)
2. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W.: Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. RFC 5280, RFC Editor (May 2008), http://www.rfc-editor.org/rfc/rfc5280.txt
3. Dierks, T., Rescorla, E.: The transport layer security (TLS) protocol version 1.2. RFC 5246, RFC Editor (Aug 2008), http://www.rfc-editor.org/rfc/rfc5246.txt
4. Eastlake, D.: Transport layer security (TLS) extensions: Extension definitions. RFC 6066, RFC Editor (Jan 2011), http://www.rfc-editor.org/rfc/rfc6066.txt
5. Farke, F.M., Lorenz, L., Schnitzler, T., Markert, P., Dürmuth, M.: "you still use the password after all" – exploring FIDO2 security keys in a small company. In: Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020). pp. 19–35. USENIX Association (Aug 2020)
6. FIDO Alliace: Client to authenticator protocol (CTAP) (Jun 2021), https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-20210615.html
7. FIDO Alliance: FIDO U2F raw message formats (Apr 2017), https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-raw-message-formats-v1.2-ps-20170411.html
8. FIDO Alliance: FIDO2: Moving the world beyond passwords using WebAuthn & CTAP (Jun 2020), https://fidoalliance.org/fido2
9. Frymann, N., Gardham, D., Kiefer, F., Lundberg, E., Manulis, M., Nilsson, D.: Asynchronous remote key generation: An analysis of yubico's proposal for W3C WebAuthn. Association for Computing Machinery, New York, NY, USA (Oct 2020). https://doi.org/10.1145/3372297.3417292
10. Google: OpenSK. https://github.com/google/OpenSK (2020)

11. Hodges, J., Mandyam, G., Jones, M.B.: RFC 8809: Registries for web authentication (WebAuthn). RFC 8809, RFC Editor (Aug 2020), https://www.rfc-editor.org/rfc/rfc8809.txt
12. IANA: Web authentication (WebAuthn) registries (Aug 2020), https://www.iana.org/assignments/webauthn/webauthn.xhtml
13. Kreichgauer, M.: Intent to deprecate and remove: U2F API (cryptotoken). https://groups.google.com/a/chromium.org/g/blink-dev/c/xHC3AtU_65A (2021)
14. Langley, A.: Re: Issue 1097972: Support WebAuthn uvi & uvm extension (Jun 2020), https://bugs.chromium.org/p/chromium/issues/detail?id=1097972#c3
15. Lyastani, S.G., Schilling, M., Neumayr, M., Backes, M., Bugiel, S.: Is FIDO2 the kingslayer of user authentication? a comparative usability study of FIDO2 passwordless authentication. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 842–859. IEEE Computer Society (May 2020). https://doi.org/10.1109/SP40000.2020.00047
16. MDN: Web authentication API. https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API (2021)
17. MDN: Web authentication API: Browser compatibility (Mar 2021), https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API#Browser_compatibility
18. Microsoft: Win32 headers for WebAuthn. https://github.com/microsoft/webauthn (2021)
19. Mooney, N.: Addition of a network transport. https://github.com/w3c/webauthn/issues/1381 (2020)
20. Protalinski, E.: You can now use your android phone as a 2FA security key for google accounts. VentureBeat (Apr 2019), https://venturebeat.com/2019/04/10/you-can-now-use-your-android-phone-as-a-2fa-security-key-for-google-accounts
21. SoloKeys: Solo 1: open security key supporting FIDO2 & U2F over USB + NFC. https://github.com/solokeys/solo (2018)
22. SoloKeys: Tutorial: Writing an extension for the solo stick. https://github.com/solokeys/solo/blob/b86f0ee4e563f0b5ceb69770a6d6f64e42a688b6/docs/tutorial-getting-started.md (2020)
23. SoloKeys: Solo 2 monorepo. https://github.com/solokeys/solo2 (2021)
24. W3C: Web authentication: An API for accessing public key credentials - level 1. W3C recommendation (Mar 2019), https://www.w3.org/TR/2019/REC-webauthn-1-20190304/
25. W3C: Web authentication: An API for accessing public key credentials - level 2. W3C recommendation (Apr 2021), https://www.w3.org/TR/2021/REC-webauthn-2-20210408/
26. W3C: Web authentication: An API for accessing public key credentials - level 3. W3C first public working draft (Apr 2021), https://www.w3.org/TR/2021/WD-webauthn-3-20210427/
27. Yubico: python-fido2 (Mar 2018), https://github.com/Yubico/python-fido2
28. Yubico: Discover YubiKey 5. strong authentication for secure login. https://www.yubico.com/products/yubikey-5-overview (Jul 2021)
29. Yubico: libfido2 (Jul 2021), https://developers.yubico.com/libfido2
30. Yubico: webauthn-recovery-extension. https://github.com/Yubico/webauthn-recovery-extension (2021)