Overcoming the UX Challenges Faced by FIDO Credentials in the Consumer Space*

Francisco Corella

Pomcor
`https://pomcor.com`
`fcorella@pomcor.com`

Abstract. Cryptographic authentication using FIDO credentials promises to improve cybersecurity by preventing man-in-the-middle phishing attacks against traditional two-factor authentication. But the FIDO Alliance reported in a March 2022 white paper that FIDO authentication had not yet attained large-scale adoption in the consumer space, citing user experience challenges such as the burden of enrolling a new device to replace a lost or stolen device. Passkey syncing is now being implemented to eliminate the need to enroll a new device with the relying party, but it requires password-based, phishing-vulnerable enrollment with the platform provider. This paper proposes and shows how to implement two alternative user experiences that overcome these challenges. The first proposed UX lets the user log in on any browser, in any device, with on-the-fly device enrollment using an email verification link for authentication. The second UX frees the user from having to set up device locking, by using as a second factor a password submitted to the relying party, instead of a device-unlocking PIN or biometric. The password is protected against reuse at malicious sites and backend database breaches by being used together with an enhanced cryptographic credential in a joint authentication procedure. The same enhanced credential is replicated in all devices, without syncing, by regenerating it from a seed derived in an HSM from a master secret and the email address.

Keywords: User experience · Cryptographic authentication · Two-factor authentication · FIDO · WebAuthn.

Patent disclosure. Pomcor owns US patent 9,887,989, which is related to the joint authentication procedure described in Section 5.1.

1 Introduction

1.1 The need for cryptogaphic authentication

Over the last decade, two-factor authentication (2FA) with a password and an authentication code has become the standard method for mitigating the well-known vulnerabilities of passwords. Revision 1 of the Electronic Authentication

* Preprint of paper to be presented at HCI International 2023.

Guidelines, published by NIST in December 2011, included 2FA as an option for authentication at Assurance Level 3, citing a code sent to the user's phone in a text message as an example of a second factor.

But traditional 2FA methods have been found to have their own vulnerabilities. Sending a code to a phone was "restricted" in Revision 3 of the Guidelines [4, §5.2.10] for being vulnerable to attacks such as "device swap, SIM change or number porting" [4, §5.1.3.3]. A one-time password (OTP) generated by an app was not restricted, but OTPs can be phished [19] just like passwords. Furthermore, a phishing attack against any authentication method based on sending secrets over the wire can be turned into a man-in-the-middle attack, allowing the attacker to log in by relaying the secrets, then observe and modify the traffic, capture the session cookie, and continue the session by importing the cookie "into a different browser, on a different computer, in a different country" [13]. Some of these vulnerabilities were reported to NIST in a response by the FIDO Alliance to a pre-draft call for comments on the forthcoming Revision 4 of the Guidelines [8].

The vulnerabilities of traditional 2FA can be avoided by using instead *cryptographic authentication.* In cryptographic authentication of a web user to a web site or web application (the "relying party" or RP), the JavaScript frontend of the RP running in the user's browser registers the public key with the backend, and later authenticates by proving possession of the private key. The private key cannot be phished because it is not sent to the backend, and a man-in-the middle phishing attack is prevented by restricting the use of the private key to JavaScript code of same web origin as the RP. FIDO authentication [15] is a method of cryptographic authentication.

1.2 FIDO2 and WebAuthn

Since its launch in 2012, the FIDO Alliance [10] has published a series of cryptographic authentication standards where a key pair called a *FIDO credential* is generated by a cryptographic module called a *FIDO authenticator*, and proof of possession of the private key is provided by a signature computed within the authenticator.

The World Wide Web Consortium (W3C) has endorsed FIDO authentication by specifying the *Web Authentication API (WebAuthn)* [20], which defines the interface that the RP frontend uses to ask the authenticator to generate a credential or compute a signature, while the *Client to Authenticator protocol (CTAP)* of the FIDO Alliance defines the communication protocol between the browser and the authenticator. Together, WebAuthn and CTAP comprise the *FIDO2 specifications* [9].

FIDO2 authenticators include *roaming authenticators*, implemented as *security keys* that communicate with the user's computing device over NFC, Bluetooth or USB, and *platform authenticators*, implemented within the device in secure storage, and made available by the operating system to the browsers running on the device. Original authenticators had a limited amount of storage and saved space by exporting the private key after encrypting it under a key-wrapping key. The wrapped private key serves as the *credential ID*, which is passed as an argument to the authenticator when requesting a signature, and is decrypted in the authenticator before signing. Platform authenticators have more space than security keys and may use *resident credentials*, a.k.a. *discoverable credentials*, which are not exported and are referenced by a randomly generated credential ID.

All OSes now provide platform authenticators, and all browsers support them. That makes FIDO2 a generally available available web technology with the potential to greatly improve the security of web applications by providing phishing resistant authentication.

But as is the case for any new technology, adoption of FIDO credentials will require a favorable user experience (UX), and FIDO credentials face UX challenges, some of which were recognized in a FIDO Alliance white paper [11, 12]. While some of these UX challenges will no doubt be ironed out as the W3C publishes incremental revisions of the WebAuthn specification, two of them are major challenges that will require rethinking of the FIDO UX. This paper proposes two alternative user experiences that overcome those challenges, and two protocols that provide those experiences.

Complexity issues in WebAuthn Besides UX challenges, adoption of FIDO authentication is no doubt also impeded by a very complex and confusing specification [12].

A particularly complex aspect of WebAuthn is *attestation* [21]. Attestation is omitted by default [20, §5.4.7] and not recommended in consumer cases [17]. It is omitted in the protocols proposed here.

Another complexity issue is how signatures are computed and verified.

All WebAuthn signatures, including the "assertion signatures", called "authentication signatures" here for clarity, are computed on a signature base derived from a challenge, rather than on the challenge itself. As shown in [20, Figure 4], the signature base is the concatenation of authenticator data and a hash of client data comprising the challenge. The process used by the RP to verify such a signature takes as inputs the authenticator data and the client data. The RP verifies that the correct challenge is found in the client data, then it verifies the signature after reconstructing the signature base by concatenating the authenticator data and the hash of the client data.

In the figures, each signature should be understood as being supplemented by the authenticator and client data that the RP backend needs to reconstruct the signature base and verify the signature.

2 First challenge: the private key is bound to the authenticator

The first challenge is not specific to FIDO credentials: it is faced by any key pair credential that is generated and used within a cryptographic module. It is a tenet of cryptography that the private key component of such a credential never leaves the module in the clear. This means that a FIDO credential can only be used in the authenticator where it was generated, and may be irrecoverably lost if the authenticator is lost.

A FIDO platform authenticator is accessible to every browser in the device and a FIDO credential can be used in any such browser. But it is not accessible to browsers in other devices, and this has been blamed for lack of adoption. The above-cited FIDO Alliance white paper [11] reported in March 2022 that FIDO2/WebAuthn “has not attained large-scale adoption in the consumer space”, and attributed this to difficulties that users face with platform authenticators: “having to re-enroll each new device”, and having “no easy ways to recover from a lost or stolen device”.

As anticipated in the white paper, Apple, Google and Microsoft are addressing this challenge by syncing FIDO credentials across platform authenticators located in devices with operating systems from the same OS vendor [7]. A synced credential is called a “passkey”, presumably because, like a password, it can be used on multiple devices.

2.1 Challenges faced by passkeys

But passkeys face their own challenges, with respect to both usability and security:

1. They weaken security by violating the cryptographic principle that a private key generated in a cryptographic module never leaves the module in the clear.
2. While a new device does not have to be enrolled with the RP, it must be enrolled with the platform provider for syncing, which may be just as onerous.
3. Enrollment with the platform provider requires authentication of the user to the platform provider with password-based, phishing-vulnerable, traditional 2FA,[1] which further weakens security and conflicts with the FIDO Alliance marketing message that FIDO authentication is passwordless and phishing resistant.
4. And credentials cannot be synced between devices with operating systems from different OS vendors.

3 First alternative user experience (UX 1): multi-device authentication without passkey syncing

The loss of credential problem is not unique to cryptographic authentication. It also occurs when a user forgets a password, and a standard solution is used to recover from that in the consumer space: an email message is sent to a registered address with a password reset link containing an email verification code. This

[1] As documented, for example, in the section on “Synchronization security” of Apple’s support article on the security of passkeys [2]

solution is phishing resistant: manually entering the code would be phishing-vulnerable, but clicking on the link is not. This solution can be adapted to construct a cryptographic authentication protocol, which we shall call *Protocol 1*, where the user can log in with any browser, on any device, without passkey syncing.

3.1 Summary of Protocol 1

To register with the RP, the user enters user data and an email address in the registration form of the relying party, shown on an initial browser. The email address is verified by a code contained in a link sent to the address, which the user opens, usually, in the initial browser.[2] The user unlocks the platform authenticator of the device where the browser is running with a biometric or a PIN, an initial FIDO credential is created by the authenticator, and the public key is registered with the RP backend. If the initial credential is a resident credential, the private key is stored in the authenticator; otherwise it is wrapped and exported as the credential ID; in either case we shall say that the browser *owns* the FIDO credential.

To log in on any browser, the user enters the email address in a login box. If the browser owns a FIDO credential, the user is authenticated by a signature computed by the platform authenticator using the private key. If not, the RP sends an email verification link to the address entered in the login box, and the user is authenticated by opening the link, usually in the same browser.[3] A new FIDO credential is created on the fly in the platform authenticator of the device where the browser is running, and a credential ID for the new credential is stored in the browser. Notice that different browsers in the same device create different credentials in the platform authenticator of the device, all with the same web origin as the RP, each referenced by a credential ID stored in the browser that owns the credential.

Fig. 1 shows the resulting user experience.

3.2 RP database schema

Fig. 2 illustrates the schema of the user database of the RP in Protocol 1.

The database comprises user records, credential records and session records. Each user record comprises the email address and the user data entered on the registration form. The email address is used as the unique identifier of the record. The user record is also used to record working data items such as the email verification code, the authentication challenge, and their issuance timestamps.

Different FIDO credentials are used to authenticate the user on different browsers, and there is a record for each of them, comprising the credential ID,

[2] If the link is opened is another browser, possibly on another device, the registration process continues on that other browser.

[3] If the link is opened in another browser, the user is logged in on that other browser, using an existing FIDO credential or a new one created on the fly.

```
REGISTRATION

A. User registers user data and email address
B. Email verification link is sent to address
C. User opens link in browser
D. User unlocks authenticator with biometric or PIN
E. User is now registered and logged in on browser,
   and browser owns a FIDO credential

LOGIN ON BROWSER THAT OWNS A FIDO CREDENTIAL

A. User submits email address
B. User unlocks authenticator with biometric or PIN
C. User is now logged in on browser

LOGIN ON BROWSER THAT DOES NOT OWN A FIDO CREDENTIAL

A. User submits email address
B. Email verification link is sent to address
C. User opens link in browser
D. User unlocks authenticator with biometric or PIN
E. User is now logged in on browser, and browser owns a FIDO credential
```

Fig. 1. UX 1

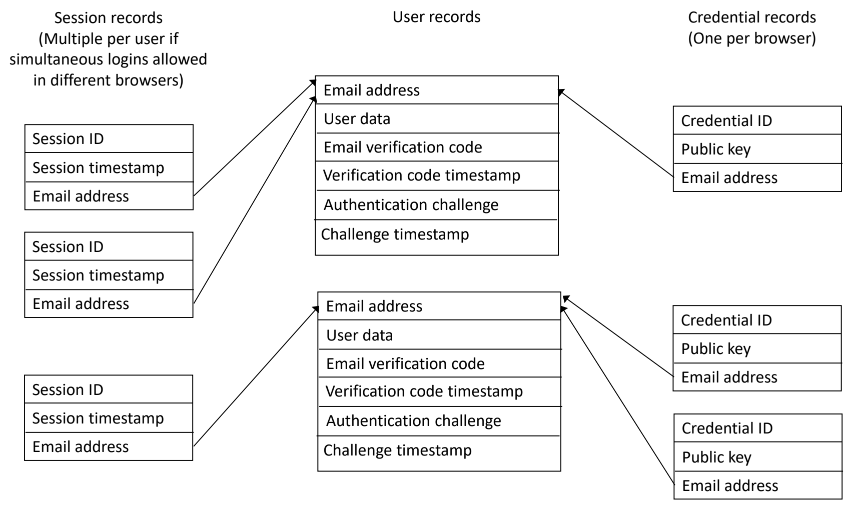

Fig. 2. Protocol 1: User database

the public key, and the user's email address, which is used as a reference to the user's record.

There may be multiple session records for a given user if simultaneous login sessions are allowed on different browsers. Each session record comprises the session ID used as the value of the session cookie, the session creation timestamp that determines expiration, and the user's email address, used as reference to the user's record.

3.3 Registration phase

Fig. 3 shows the steps of the registration phase of Protocol 1.

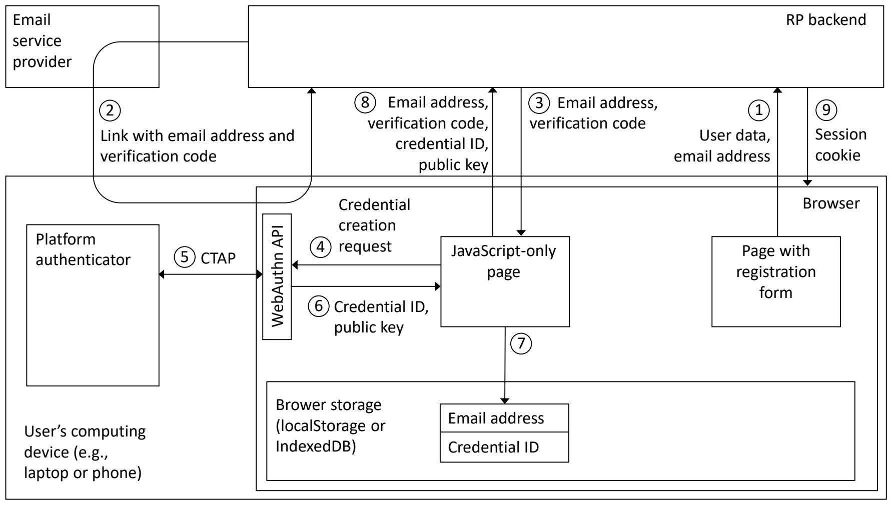
Email service provider
RP backend
2 Link with email address and verification code
8 Email address, verification code, credential ID, public key
3 Email address, verification code
1 User data, email address
9 Session cookie
Browser
Platform authenticator
5 CTAP
WebAuthn API
Credential creation
4 request
JavaScript-only page
Page with registration form
6 Credential ID, public key
7
Brower storage (localStorage or IndexedDB)
Email address
Credential ID
User's computing device (e.g., laptop or phone)

Fig. 3. Protocol 1: Registration

1. The user submits the RP's registration form with user data and the user's email address, and the RP backend creates a user record with the data and address, and a randomly generated email verification code.
2. The RP backend emails a link to the email address with the address and an email verification code. The user opens the link in the browser, causing the browser to send an HTTP request to the RP backend, containing the email address and the verification code. (All HTTP requests and responses should be understood as being sent over TLS.)
3. The RP backend verifies the code against the user record referenced by the email address, and sends an HTTP response to the browser with a JavaScript-only page containing the email address and the verification code. The verification code is included in the response so that it can be used in

step 8 to authenticate the browser to the backend; an alternative to using the verification code for this purpose would be to create a session (not yet a login session) and use the session ID.

4. The JavaScript code in the page calls the function `navigator.credentials.create` of the WebAuthn API to request the creation of a FIDO credential.
5. The browser communicates with the platform authenticator using the CTAP protocol, transmitting the request. The user is prompted to unlock the authenticator by supplying a biometric or a PIN. The authenticator creates a FIDO credential and returns the credential ID and the public key.
6. The browser asynchronously responds to the call to `navigator.credentials.create` with an object that contains the credential ID and the public key.
7. The code in the JavaScript-only page creates a record in browser storage (either LocalStorage or an IndexedDB database) containing the email address and the credential ID.
8. The code in the JavaScript-only page sends an HTTP POST request to the RP backend conveying the email address, the verification code, the credential ID and the public key. The RP backend verifies the code a second time and creates a credential record.
9. The RP backend creates a session record and sets the session cookie.

3.4 Authentication on a browser that owns a FIDO credential

Fig. 4 shows the steps of the authentication phase of Protocol 1 when the browser already owns a credential.

1. The user visits an RP page containing a form with a text input field for entering an email address, enters his/her email address in the field, and requests submission of the form. A form submission event listener finds a record in browser storage containing the email address and a credential ID, and copies the credential ID to a hidden input of the form.
2. The form submission event listener submits the form, sending an HTTP POST request to the RP backend that conveys the email address and the credential ID.
3. The RP backend uses the email address to find the user's record, generates an authentication challenge that it records in the user's record, and responds to the HTTP request with a JavaScript-only page containing the email address, the credential ID and the challenge.
4. The JavaScript code in the page calls the function `navigator.credentials.get` of the WebAuthn API, passing as an argument an object that contains the credential ID and the challenge.
5. The browser communicates with the platform authenticator using the CTAP protocol, forwarding the credential ID and the challenge. The user is prompted to unlock the authenticator by supplying a biometric or a PIN. The authenticator computes the authentication signature and returns it along with authenticator data

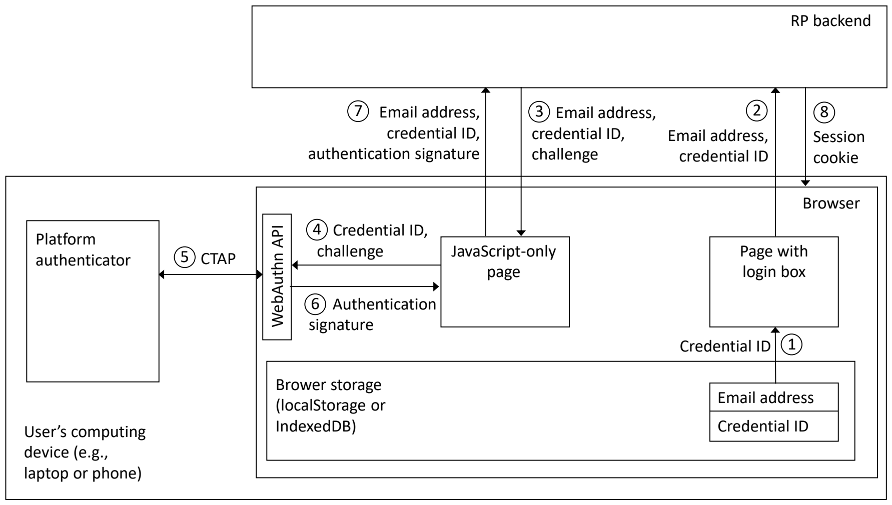
RP backend
⑦ Email address, credential ID, authentication signature
③ Email address, credential ID, challenge
② Email address, credential ID
⑧ Session cookie
Browser
Platform authenticator
⑤ CTAP
WebAuthn API
④ Credential ID, challenge
⑥ Authentication signature
JavaScript-only page
Page with login box
Credential ID ①
Brower storage (localStorage or IndexedDB)
Email address
Credential ID
User's computing device (e.g., laptop or phone)

Fig. 4. Protocol 1: Authentication on browser owning a credential

6. The browser asynchronously responds to the call to `navigator.credentials.get` with an object that contains the signature, supplemented with the authenticator and client data (not shown in the figure) that the RP backend needs to reconstruct the signature base as explained in Section 1.2.
7. The code in the JavaScript-only page sends an HTTP POST request to the RP backend conveying the email address, the credential ID, and the supplemented authentication signature.
8. The RP backend uses the email address to locate the user record, and the credential ID along with the user record to locate the credential record for the credential owned by the browser. It verifies the challenge found in the client data against the user record and authenticates the user by verifying the signature. Then it logs the user in by creating a session record and setting the session cookie.

3.5 Authentication on a browser that does not own a FIDO credential

Fig. 5 shows the steps of the authentication phase of Protocol 1 when the browser does not yet own a credential.

1. The user visits an RP page containing a form with a text input field for entering an email address, enters his/her email address in the field, and requests submission of the form. A form submission event listener cannot find a record in browser storage containing the email address.

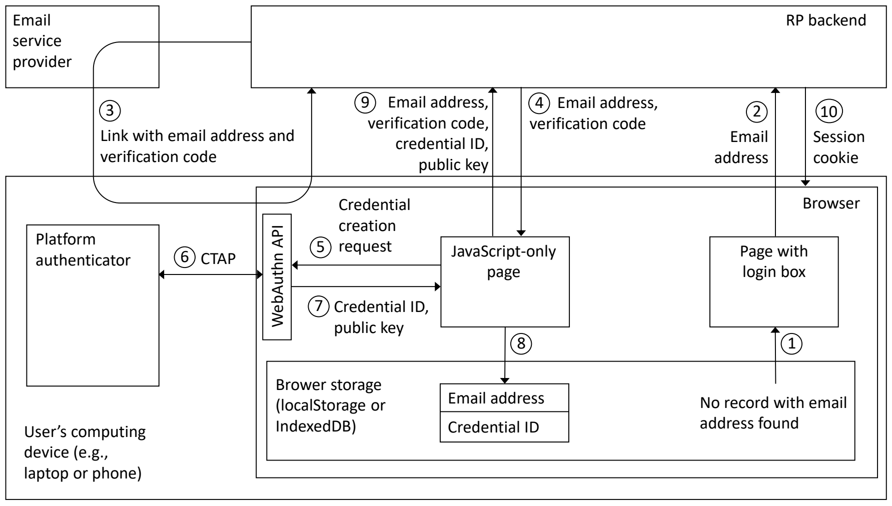
Email service provider
RP backend
3 Link with email address and verification code
9 Email address, verification code, credential ID, public key
4 Email address, verification code
2 Email address
10 Session cookie
Browser
Credential creation request
5
Platform authenticator
6 CTAP
WebAuthn API
JavaScript-only page
Page with login box
7 Credential ID, public key
8
1
Brower storage (localStorage or IndexedDB)
Email address
Credential ID
No record with email address found
User's computing device (e.g., laptop or phone)

Fig. 5. Protocol 1: Authentication on browser not owning a credential

2. The form submission event listener submits the form with the email address as-is, sending an HTTP POST request to the RP backend that conveys the email address.
3. The RP backend verifies that there is a user record with the submitted email address, then sends a link to the email address with the address and an email verification code, which the user opens as in step 2 of the registration phase.
4. Steps 4–10 are then as steps 3–9 of the registration phase.

4 Second challenge: reliance on the device unlocking mechanism

The second major UX challenge is specific to FIDO2 and WebAuthn. Cryptographic authentication needs a second factor for protection against theft of the hardware where the private key is stored, and FIDO2 uses as second factor the same biometric or PIN used to unlock the user's device; but many users do not set up the device unlocking mechanism. There is evidence, for example, that only about 30% of Windows users set up Windows Hello [1].

Large scale adoption of FIDO authentication would require convincing most users of setting up device unlocking, and that is going to be difficult. Recent user research [6] has shown that depending on biometrics as the sole authentication method for unlocking a device raises anxieties about being locked out of the device; therefore a PIN would have to be used instead of, or as backup for a biometric. But a PIN is a very weak and much reused password; and asking users to use a PIN for authentication conflicts with the "passwordless authentication" marketing campaign of the FIDO Alliance, and the decades of cybersecurity user education arguing against weak passwords and password reuse.

5 Second alternative user experience (UX 2): cryptographically protected password as second factor

The second alternative UX uses a full-fledged password as a second factor instead of a PIN or a biometric. But the password is not submitted to the RP backend separately and independently from the cryptographic credential, which would make it vulnerable to reuse at malicious sites and backend database breaches. The password and the cryptographic credential protect each other by being used together in a joint authentication procedure.

5.1 Joint authentication with an enhanced credential and a password

While the UX of Section 1 can be implemented using ordinary FIDO authenticators and credentials, UX 2 requires several extensions to WebAuthn, as recapitulated below in Section 5.3.

One of those extensions is the option to generate an enhanced credential that can be combined with a password in a joint authentication procedure. An enhanced credential differs from an ordinary credential in two ways: (i) it has an additional component that is hashed with the password as a *secret salt*; and (ii) its public key is retained in the authenticator and is not stored in the backend database. In the joint authentication procedure, the RP frontend submits the authentication signature, the salted password and the public key to the backend. The backend uses the public key to verify the signature, then computes a *joint hash* of the salted password and the public key that it compares with a registered joint hash.

This protects the password against reuse at a malicious site, because different sites use different secret salts; and it protects both the password and the key pair against database breaches. In case of a database breach, the password is protected against a dictionary attack by being hashed first with the secret salt and then with the public key; and, by being hashed with the salted password, the key pair is protected against any weakness of its underlying cryptosystem that might be discovered by an adversary, and against a postquantum brute force attack that would attempt to derive the private key from the public key.

5.2 Replicating the enhanced credential without syncing

At first glance UX 1 and UX 2 are mutually exclusive. It would seem that an enhanced credential used to implement UX 2 could only be used on one device, because the same password must be used on all devices, and the public key of the credential generated in one device would not be available for computing the joint hash and verifying the password when creating a new credential in a different device.

This difficulty could be solved by syncing the enhanced credential across devices, as passkeys are now being synced. But that would negate the benefits of UX 1.

The protocol proposed here to implement UX 2, which we shall call *Protocol 2*, solves the difficulty instead by generating the enhanced credential using the same pseudo-random bit generation seed in all devices. The seed is computed by the RP backend in a hardware security module (HSM) from the email address and a master secret randomly generated in the module from a noise source. The seed can be derived, for example, as the PRK output of HKDF-Extract [16, §2.2], using the address as the HKDF salt, and the master secret as the IKM input. The seed is then included in the email verification link along with the email verification code at registration, and when creating a credential for a new device.

Using the same seed will result in having the same credential in all devices without syncing, provided that the same method is consistently used to generate the credential from the seed. This means that the credential creation options passed as input to `navigator.credentials.create` will have to determine not only the type of credential to be used (such as an ECDSA key pair suitable for use with COSE algorithm ES256 [14], or an RSA key pair usable with algorithm RS256) but also the procedure to be used for computing the key pair from the seed.

That can be done by specifying: (i) how to derive a stream of random bits from the seed; and (ii) how to compute one or more components of the credential from one or more portions of the stream. The bit stream can be derived, for example, using any of the DRBG algorithms of [3, §10] with specific parameters, or HKDF-Expand [16, §2.2] using the seed as the PRK input. The component computation is easy to specify for an ES256 credential: the private key d is a random number in the range $0 \ldots n$ where n is the order of the NIST P256 curve [5], which can be computed using, for example, the extra random bits method of [18, §B.1.1] by reducing modulo n the integer having as its binary representation the first $256 + 64 = 320$ bits of the stream. The public key is then the scalar product dG of the private key with the base point G of the curve. It would be more complicated to specify the computation of the p and q primes for an RS256 credential but, although RS256 is used by Windows Hello, it is not a recommended COSE algorithm [14] and may be replaced with ES256 in the future.

5.3 Required extensions to WebAuthn

To recapitulate, the following modifications to WebAuthn are needed to implement Protocol 2:

– The function `navigator.credentials.create` used at registration must provide the option to create an *enhanced credential* that comprises a secret salt as an additional component and whose public key is retained by the authenticator.

- When an enhanced credential is requested, `navigator.credentials.create` must take a DRBG seed as an additional input and use it to generate the pseudo-random bits used to construct the credential.
- When creating an enhanced credential, `navigator.credentials.create` must take a password as an additional input, hash the password with the secret salt, and outputs the joint of hash the public key and the salted password instead of the public key.
- When authenticating with an enhanced credential, the function `navigator.credentials.get` must take the password as an additional input, compute the hash of the password and secret salt, and ouput the retained public key and the salted password in addition to the authentication signature.
- When creating or using an enhanced credential, the user must not be asked for a biometric or PIN to unlock the authenticator.

5.4 Summary of Protocol 2

To register, the user enters the email address and user data in a registration box. The RP backend creates a user record with the address and the data, then it derives a seed from the address and emails a link with the seed and an email verification code. The user opens the link and is prompted to register a password. The RP frontend inputs the seed and the password to the authenticator, which creates the enhanced credential and returns the credential ID and the joint hash of the public key and the salted password. The RP frontend creates a record with the email address and the credential ID in browser storage and sends the email address, the credential ID, and the joint hash to the RP backend. The backend adds the joint hash to the user record and logs the user in by creating a session record and setting a session cookie.

To log in, the user enters the email address in a login box. The RP frontend looks for a record containing the email address and a credential ID in browser storage.

If such a record is found, the email address and the credential ID are submitted to the backend, which generates a challenge and responds with a page containing a password submission box and JavaScript code containing the email address, the credential ID and the challenge. The user supplies the password and the RP frontend inputs the challenge, the credential ID and the password to the authenticator, which returns a signature, the salted password, and the public key. The RP frontend submits the email address, the salted password, the public key and the signature to the backend, which verifies the signature, computes the joint hash of the public key and the password, and verifies the joint hash against the user record referenced by the email address. The user is thus authenticated by possession of the private key and knowledge of the password. Then the backend logs the user in by creating a session record and setting a cookie with the session ID.

If no such record is found, the RP backend derives the seed from the email address and sends the seed to the address along with an email verification code in a link. The user opens the link and is prompted for the password. The authenticator creates an enhanced credential identical to the one that was created at registration. (The authenticator may have multiple replicas of the credential for multiple browsers installed in the device, each with a different credential ID.) The authenticator computes the hash of the password and the secret salt and outputs the joint hash of the public key and the salted password, which is sent to the backend and verified against the registered joint hash. The user is thus authenticated by having received the email verification link and knowing the password. Then the backend logs the user in by creating a session record and setting a cookie with the session ID.

Fig. 6 shows the resulting user experience.

```
REGISTRATION

A. User registers user data and email address
B. Link with email verification code and DRBG seed is sent to address
C. User opens link in browser and registers password
D. User is now registered and logged in on browser,
   and browser owns a credential

LOGIN ON BROWSER THAT OWNS A CREDENTIAL

A. User submits email address and is prompted for password
B. User submits password
C. User is now logged in on browser

LOGIN ON BROWSER THAT DOES NOT OWN A CREDENTIAL

A. User submits email address
B. Link with email verification code and DRBG seed is sent to address
C. User opens link in browser and is prompted for password
D. User submits password
E. User is now logged in on browser and browser owns a credential
```

Fig. 6. UX 2

5.5 RP database schema

Fig. 7 illustrates the schema of the user database of the RP in Protocol 2.

Since the same enhanced credential is used for all browsers in all devices, there are no credential records. Instead, each user record stores the joint hash of the public key and the salted password. Notice how the public key is not stored in the database, as it is retained by the enhanced authenticator and submitted along with the salted password for authentication.

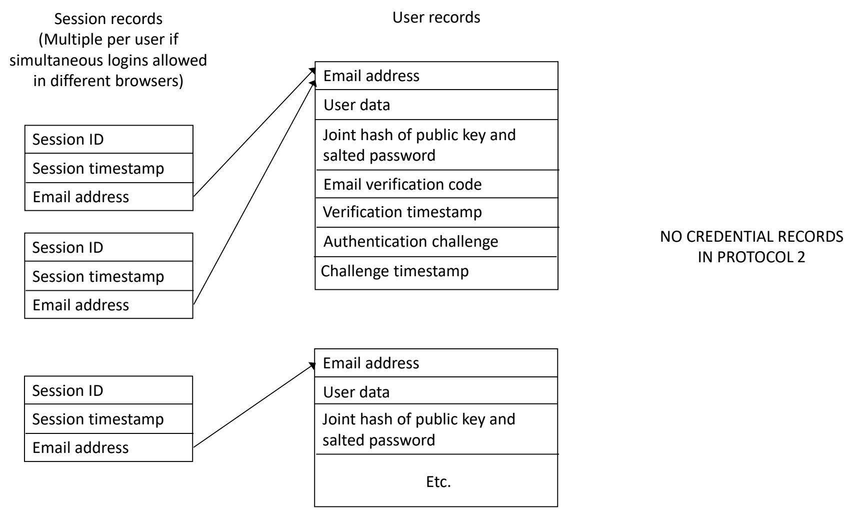

Fig. 7. User database of the RP

5.6 Registration phase

Fig. 8 shows the steps of the registration phase of Protocol 2.

1. The user submits the RP's registration form with user data and the user's email address; the RP backend creates a user record comprising the data, the address, and a randomly generated email verification code.
2. The RP backend inputs the email address to a hardware security module (HSM) containing a master secret. The HSM outputs a bit string to be used as a DRBG seed, computed as the PRK output of HKDF-Extract [16, §2.2] with the master secret as the IKM input and the email address as the salt input.
3. The RP backend sends a link to the email address with the email address, the verification code and the seed. The user opens the link in the browser, causing the browser to send an HTTP request to the RP with the contents of the link.
4. The RP backend verifies the code against the user record referenced by the email address, and sends an HTTP response to the browser with a password registration form. JavaScript code in the page contains the email address and the seed, as well as the verification code, which will be used in step 9 to authenticate the browser to the RP backend; an alternative to using the verification code for this purpose would be to create a session (not yet a login session) and use the session ID.
5. The user supplies a password. The JavaScript code in the page calls the function `navigator.credentials.create` of the extended WebAuthn API to request the creation of an enhanced credential, passing the seed and the password.

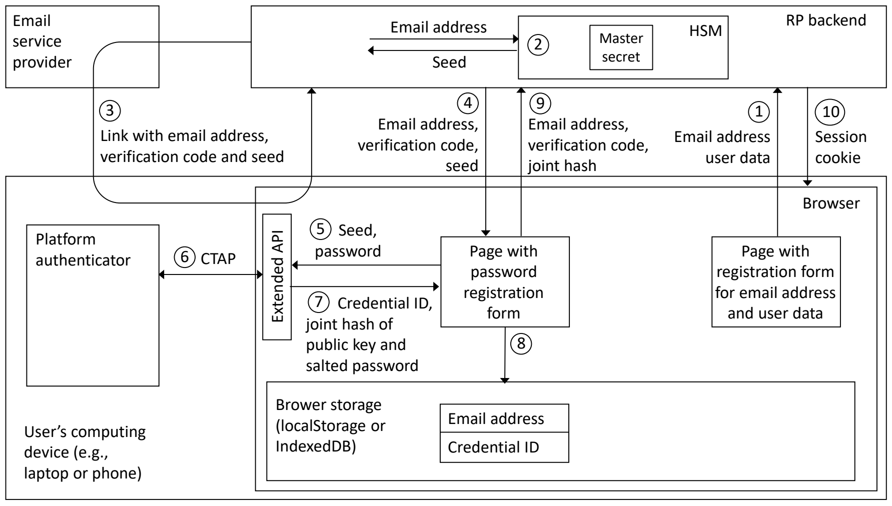
Email service provider
RP backend
Email address
Seed
2
Master secret
HSM
3
Link with email address, verification code and seed
4
Email address, verification code, seed
9
Email address, verification code, joint hash
1
Email address user data
10
Session cookie
Browser
Platform authenticator
6 CTAP
Extended API
5 Seed, password
7 Credential ID, joint hash of public key and salted password
Page with password registration form
Page with registration form for email address and user data
8
Brower storage (localStorage or IndexedDB)
Email address
Credential ID
User's computing device (e.g., laptop or phone)

Fig. 8. Protocol 2: Registration

6. The browser communicates with the platform authenticator using an enhanced version of the CTAP protocol, and transmits the request. The user is NOT prompted to unlock the authenticator by supplying a biometric or a PIN. The authenticator creates an enhanced credential, computes the hash of the password and the secret salt, and returns the credential ID and the joint hash of the public key and the salted password.
7. The browser asynchronously responds to the call to `navigator.credentials.create` with an object that contains the credential ID and the joint hash.
8. The JavaScript code in the password registration page creates a record in browser storage (either LocalStorage or an IndexedDB database) containing the email address and the credential ID.
9. The JavaScript code in the password registration page sends an HTTP POST request to the RP backend conveying the email address, the verification code and the joint hash. The RP backend verifies the code again and adds the joint hash to the user record referenced by the email address.
10. The RP backend creates a session record and responds to the POST request with an HTTP response that sets a cookie with the session ID in the browser.

5.7 Authentication on a browser that owns a FIDO credential

Fig. 9 shows the steps of the authentication phase of Protocol 2, when the browser already owns a credential.

1. The user visits an RP page containing a form with a text input field for entering an email address, enters his/her email address in the field, and

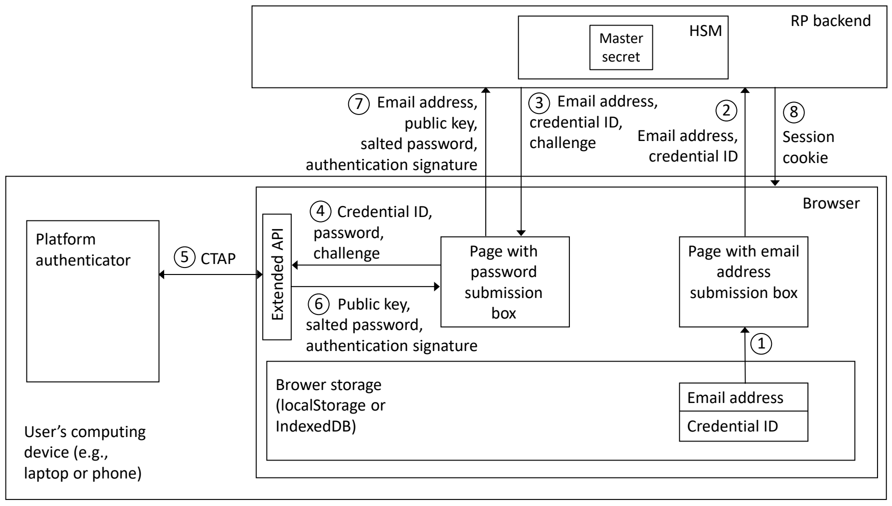
RP backend
HSM
Master secret
7 Email address, public key, salted password, authentication signature
3 Email address, credential ID, challenge
2 Email address, credential ID
8 Session cookie
Browser
Platform authenticator
5 CTAP
Extended API
4 Credential ID, password, challenge
6 Public key, salted password, authentication signature
Page with password submission box
Page with email address submission box
1
Brower storage (localStorage or IndexedDB)
Email address
Credential ID
User's computing device (e.g., laptop or phone)

Fig. 9. Protocol 2: Authentication on browser owning a credential

submits the form. A form submission event listener finds a record in browser storage containing the email address and a credential ID, and copies the credential ID to a hidden input of the form.

2. The form submission event listener submits the form, sending an HTTP POST request to the RP backend that conveys the email address and the credential ID. The credential ID is not stored in the backend. It is sent in this step so that it can be returned in the next step and used in step 4.
3. The RP backend generates an authentication challenge, records it in the user's record along with a challenge creation timestamp, and responds to the HTTP request with a page for completing the login by entering a password. JavaScript code in the page contains the email address, the credential ID and the challenge.
4. The JavaScript code in the password submission page calls the function `navigator.credentials.get` of the extended WebAuthn API, passing as an argument an object that contains the challenge, the credential ID and the password.
5. The browser communicates with the platform authenticator using an enhanced version of the CTAP protocol, forwarding the challenge, the credential ID and the password. The user is NOT prompted to unlock the authenticator by supplying a biometric or a PIN. The authenticator computes the hash of the password with the secret salt, derives a signature base from the challenge as explained above in Section 1.2, signs it with the private key of the credential, and sends the signature, the salted password and the public key to the browser along with the authenticator data and the client data that the backend will need to reconstruct the signature base.

6. The browser asynchronously responds to the call to `navigator.credentials.get` with an object that contains the the salted password, the public key, and the signature supplemented with the authenticator and client data.
7. The JavaScript code in the password submission page sends an HTTP POST request to the RP backend conveying the email address, the salted password, the public key, and the signature supplemented with the authenticator and client data.
8. The RP backend uses the email address to locate the user record and verifies that the challenge recorded in the user record is recent, and is the one found in the client data. It hashes the client data and reconstructs the signature base by concatenating the authenticator data and the hash of the client data. It uses the public key to verify the signature on the signature base. It computes the joint hash of the public key and the salted password and verifies it against the user record. Then it logs the user in by creating a session record and responding to the POST request with an HTTP response that sets a cookie with the session ID in the browser.

5.8 Authentication on a browser that does not own a FIDO credential

Fig. 10 shows the steps of the authentication phase of Protocol 2, when the browser does not yet own a credential.

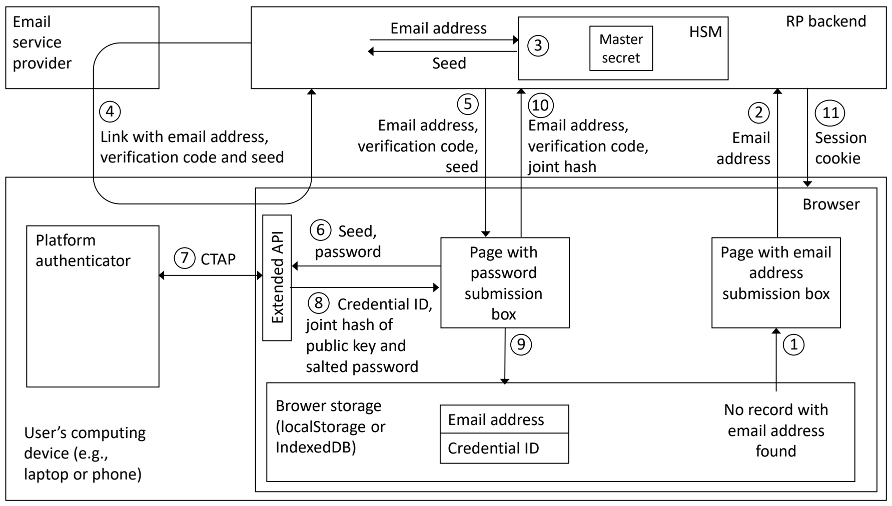
Email service provider
RP backend
Email address
Seed
3
Master secret
HSM
4
Link with email address, verification code and seed
5
Email address, verification code, seed
10
Email address, verification code, joint hash
2
Email address
11
Session cookie
Browser
Platform authenticator
7 CTAP
Extended API
6 Seed, password
8 Credential ID, joint hash of public key and salted password
Page with password submission box
9
Page with email address submission box
1
Brower storage (localStorage or IndexedDB)
Email address
Credential ID
No record with email address found
User's computing device (e.g., laptop or phone)

Fig. 10. Protocol 2: Authentication on browser lacking a credential

1. The user visits an RP page containing a form with a text input field for entering an email address, enters his/her email address in the field, and submits the form. A form submission event listener cannot find a record in browser storage containing the email address.
2. The form submission event listener submits the form with the email address as-is.
3. The RP backend verifies that there is a user record with the submitted email address, then inputs the email address to the HSM and obtains the seed as in Fig. 8.
4. Steps 4–11 are then as steps 3–10 of the registration phase, except that, at step 10, the RP backend verifies the joint hash against the user record instead of adding it to the user record.

6 Conclusion

This paper has proposed two protocols for two-factor cryptographic authentication to a web site or web application that overcome the user experience challenges of FIDO credentials. Both protocols allow the user to log in on any browser, in any device, with authentication by email verification and on-the-fly browser enrollment.

The first protocol uses ordinary FIDO credentials and authenticators. When a browser is enrolled, a key pair is generated for the browser in the platform authenticator of the device, and a record containing the public key, the credential ID and a reference to the user record is added to the backend database. The second authentication factor is provided by the biometric or PIN supplied by the user to unlock the authenticator.

The second protocol uses a cryptographically protected password as a second factor instead of a PIN or biometric, but requires an enhanced credential that comprises a secret salt in addition to the key pair. A two-factor joint authentication procedure protects the password against reuse at malicious sites and database breaches. If the database is compromised, it also protects the public key against exploitation of any cryptographic weakness that may be discovered by an adversary in the underlying cryptosystem, and against any postquantum brute force attempt to compute the private key from the public key. The use of the password and the credential in combination requires the same enhanced credential to be used in all devices. This is achieved, without passkey syncing, by using a pseudo-random bit generation seed derived in an HSM from the email address and a master secret to generate the credential.

References

1. Informal interview with a Geek Squad Agent on February 22, 2023
2. Apple Support: About the security of passkeys, `https://support.apple.com/en-us/HT213305`

3. Barker, E., Kelsey, J.: Recommendation for Random Number Generation Using Deterministic Random Bit Generators (June 2015), NIST Special Publication 800-90A Revision 1. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf
4. Burr, W.E., Dodson, D.F., Polk, W.T.: NIST SP 800-63-1 Electronic Authentication Guideline (December 2011), http://csrc.nist.gov/publications/nistpubs/800-63-1/SP-800-63-1.pdf
5. Center for Research on Cryptography and Security: Standard curve database, P-256, https://neuromancer.sk/std/nist/P-256
6. Chuhan, S., Wojnas, V.: Designing and evaluating a resident-centric digital wallet experience, To appear in the proceedings of HCI International 2023
7. FIDO Alliance: Apple, Google and Microsoft Commit to Expanded Support for FIDO Standard to Accelerate Availability of Passwordless Sign-Ins, May 5, 2022. https://fidoalliance.org/apple-google-and-microsoft-commit-to-expanded-support-for-fido-standard-to-accelerate-availability-of-passwordless-sign-ins/
8. FIDO Alliance: FIDO Alliance Input to the National Institute of Standards and Technology (NIST), August 2020. https://www.nist.gov/system/files/documents/2020/09/08/Comments-800-63-009.pdf
9. FIDO Alliance: FIDO2 Specifications, https://fidoalliance.org/fido2/
10. FIDO Alliance: Open Authentication Standards More Secure than Passwords, fidoalliance.org
11. FIDO Alliance: White Paper: Multi-Device FIDO Credentials, March 2023. https://fidoalliance.org/white-paper-multi-device-fido-credentials/
12. Firstyear (anonymous blogger): Exploring Webauthn Use Cases, June 13, 2022. https://fy.blackhats.net.au/blog/html/2022/06/13/exploring_webauthn_use_cases.html
13. Gretzky, K.: Evilginx 2 – Next Generation of Phishing 2FA Tokens, July 26, 2018. https://breakdev.org/evilginx-2-next-generation-of-phishing-2fa-tokens/
14. IANA: COSE algorithms, https://www.iana.org/assignments/cose/cose.xhtml#algorithms
15. Jen Easterly, Director, CISA: NEXT LEVEL MFA: FIDO AUTHENTICATION, October 18, 2022. https://www.cisa.gov/blog/2022/10/18/next-level-mfa-fido-authentication
16. Krawczyk, H., Eronen, P.: HMAC-based Extract-and-Expand Key Derivation Function (HKDF), RFC 5869, May 2010. http://tools.ietf.org/html/rfc5869
17. Langley, A.: Attestation not recommended in consumer cases. https://groups.google.com/a/chromium.org/g/security-dev/c/BGWA1d7a6rI/m/nwOt22fDBAAJ?pli=1
18. NIST: Digital Signature Standard (DSS) (July 2013), FIPS PUB 186-4, http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
19. Sachs, E., Dingle, P.: The Cutting-Edge: Standards at Work In Google's Mobile-Focused Future, Presentation at Cloud Identity Summit 2015. https://www.youtube.com/watch?v=UBjEfpfZ8w0
20. W3C: Web Authentication: An API for accessing Public Key Credentials Level 3, W3C First Public Working Draft, 27 April 2021. https://www.w3.org/TR/webauthn-3/
21. Yubico: Attestation, WebAuthn Developer Guide. https://developers.yubico.com/WebAuthn/WebAuthn_Developer_Guide/Attestation.html