27-11-2023

# White Paper: Passkeys Use and Deployment for R&E Services

|  |  |
|---|---|
| Grant Agreement No.: | 101100680 |
| Work Package: | WP5 |
| Task Item: | Task 5 |
| Nature of Document: | White Paper |
| Dissemination Level: | PU (Public) |
| Lead Partner: | UoB/AMRES |
| Document ID: | GN5-1-23-FB1864 |
| Authors: | B. Marović (UoB/AMRES); P. Břoušek (CESNET); M. van Es (SURF); N. van Dijk (SURF); M. Schmidt (LRZ/DFN) |

## Abstract

Passkeys are a more secure and convenient alternative to passwords, which they aim to replace as the primary method for account authentication. This white paper describes how they work; their usage; key aspects of their implementation, in particular for R&E services; and next steps with regard to development, transition and adoption.

# Table of Contents

## Table of Figures

## Table of Tables

# 1 Overview

Passkeys are a more secure and user-friendly alternative to passwords, aiming to replace passwords as the primary method for account authentication. The goal is to provide an effortless login experience while preventing anyone from stealing or guessing user credentials. Unlike passwords, passkeys do not require the website or service (referred to as the Relying Party) to store secret data, reducing the risk of secret theft.

In step-up authentication or transaction confirmation, passkeys can be used after the user has already signed in and wants to perform critical operations or access particularly sensitive data. The user is asked to complete an extra step to confirm their identity before accessing the service or completing the transaction. The validity of the step-up may be limited to a specific operation or time-bound. Passkeys can also be used in enhanced security scenarios where one or more passkeys are used as additional factors to ensure user verification and presence.

According to common understanding, passkeys are FIDO2/WebAuthn public key credentials that can be discovered on devices during sign-in. The user chooses them from among those applicable to the Relying Party. They require user verification and can usually be copied to the user's other devices and backed up. One way to describe them is 'A passkey is a FIDO2 Discoverable Credential that is protected against device loss' [PkDef]. Protection against device loss implies user verification and the possibility of sharing and backup through credential management platforms. Additionally, single-device passkeys are specific to a particular device and cannot be copied.

For users, using passkeys is similar to using stored passwords but with less inconvenience [PkStats] and more security. Passkeys reduce the need for additional secondary authentication methods like SMS, app-based one-time codes or confirmation-key-based approvals. The use of passkeys is secured by existing device-unlocking mechanisms, including biometric features such as facial or fingerprint recognition, PIN codes or locally managed passwords that are securely stored and verified by matching them with hardware-secured data. Modern smartphones and laptops typically have the necessary hardware to support these features. The improved usability and security increase user trust and enhance the overall user experience, covering typical usage scenarios and real-life situations such as device loss recovery and seamless service integration for developers, which is device- and hardware-independent unless otherwise specified.

Passkeys authenticate users against credentials stored locally on their devices. While a passkey can be specific to one device, it is typically shared across multiple devices owned by the user. The user can also sign in on one device (their own or someone else's computer) using a passkey stored on another device (such as their smartphone or security key).

Even if user verification is based on a PIN, two factors are involved: the authenticator device and the credential possessed by the user, with the PIN known by the user as the second factor. In recent years, additional protection mechanisms leveraging secure device hardware and sensors have become common on mobile devices and laptops. Passkeys provide a standardised way to utilise this hardware. However, due to the need to support different devices with varying levels of protection and user verification methods, passkeys may not always be accepted by services that require a higher level of assurance. Therefore, the need for multi-factor authentication (MFA) is expected to persist, as passkeys may still need to be combined with additional factors in certain scenarios.

Passkeys are the result of an industry-wide effort and are supported by leading client operating systems and platforms. They combine secure authentication standards, WebAuthn [WebAuthn] and CTAP2 [CTAP2], developed by the W3C Web Authentication Working Group and the FIDO Alliance, respectively. These standards ensure consistent behaviour, user experience and terminology across platforms.

Passwords are still the most commonly used authentication method today, but passkeys have the potential to render them obsolete. They are already supported by major service providers and websites, such as Google, Apple, Microsoft, Twitter, Facebook, GitHub, Dropbox, Cloudflare, Yahoo, AOL, Zoho, Okta, ID.me, Login.gov, PayPal, GoDaddy, WordPress, Kayak, eBay, Best Buy, Shopify, DocuSign, etc. A partial list of supporting websites, applications and services is available at [PkDir]. However, a complete transition requires significant time, effort and resources for service providers to modify their services and websites. Therefore, the widespread adoption of passkeys may take some time. Consequently, solutions that can be easily integrated and support the implementation of passkeys, along with related guidelines, could help expedite this transition.

This white paper describes how passkeys work (Section 2) and their usage (Section 3). It considers key aspects of their implementation (Section 4) and their integration in popular identity provider products (Section 5), outlines additional implementation factors for research and education services, including related transition and adoption challenges (Section 6), and examines the role of passkeys in multi-factor authentication and level of assurance (Section 7). It also elaborates findings from testing on various platforms (Section 8), before presenting conclusions and next steps (Section 9). Two passkey web integration prototypes are presented in Appendix A, and the relationship between the REFEDS Single-Factor Authentication profile and passkeys is discussed in Appendix B.

# 2 How Do Passkeys Work?

Passkeys seamlessly work across different operating systems and browsers and can be used on websites and in native applications. They are suitable for signing in in common service scenarios and for more sensitive accounts, such as those providing banking and financial services. Passkeys offer several mechanisms that enhance and integrate the user experience, which is crucial for their wider adoption as a passwordless authentication method. This section outlines passkeys' security features, architecture and governing standards.

## 2.1 Security Features

Unlike passwords, which are prone to human errors, passkeys cannot be seen, leaked or reused with multiple websites or accounts. Here are some key security-related features of passkeys:

- Users do not directly generate passkeys and they are not available or used in plain text. Their secret private key information is not visible, even to the user. They cannot be weak, reused, read from the screen, written down on paper, forgotten or guessed.
- It is much more challenging to trick users into sharing their passkeys.
- Passkeys are not transmitted during the sign-in process, making them immune to interception by attackers.
- Passkeys cannot be stolen from the service in a service database breach. Even if a service is compromised, the passkey's public key is useless to the attacker, unlike the hashed password of an account, which can be cracked.
- Users cannot use passkeys on the wrong website or casually share them with malicious individuals, making them less susceptible to simple mistakes, phishing and social engineering attacks. Unlike SMS codes or application-based one-time passwords, passkeys protect against phishing attempts.
- Passkeys do not enable tracking of users or devices between websites. None of the information shared with websites can be used as a tracking vector. Passkeys are unique for each website and cannot be used across multiple websites.
- The user's biometric data is never exposed to the website or application. Biometric material never leaves the user's device, while the remote server only receives confirmation that the biometric verification was successful.
- Passkeys cannot be used from arbitrary locations or devices, unlike passwords, which can be used from anywhere. With passkeys, the user, the device accessing the service and the authentication device must be in proximity to each other.

## 2.2 Architecture

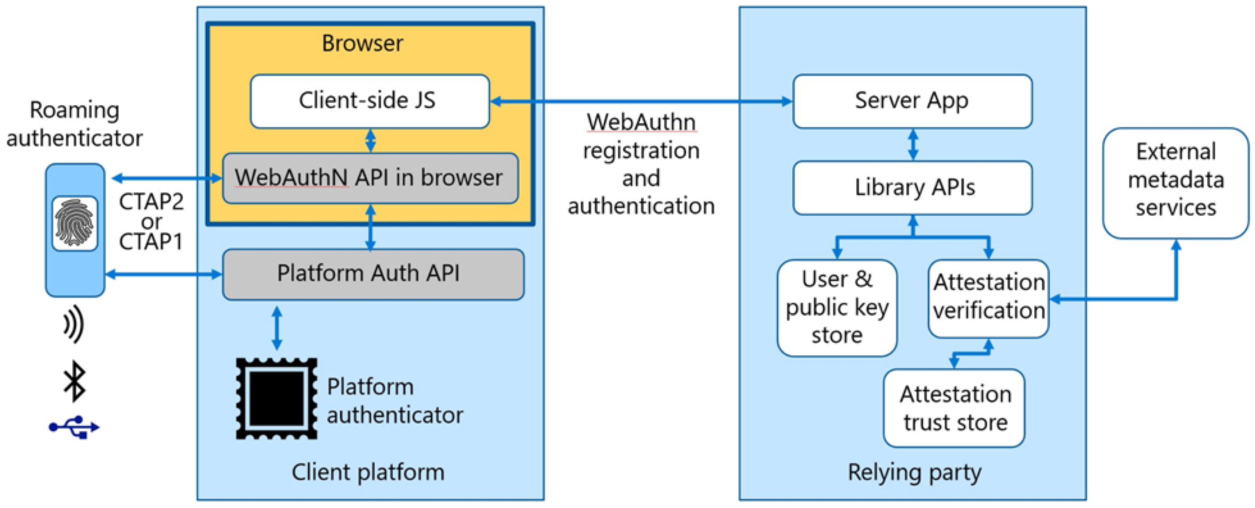


Figure 2.1: Passkey architecture: FIDO2 (CTAP2) and WebAuthn [FIDO2Comps]

Several components work together to provide a secure and user-friendly authentication mechanism using passkeys for online services and applications:

- **Roaming authenticator** (or external authenticator) – A FIDO2 authenticator that is not physically attached to the user's device. It is a separate device that can be connected to the user's device via Bluetooth, USB or other methods. With passkeys, a smartphone can also act as a roaming authenticator.
- **Client platform** – The user's device, such as a smartphone or laptop, where the authentication process is initiated. The client side of the application runs within the browser or native application.
- **Browser** – Enables users to access websites and web applications and provides the WebAuthn API for web applications to communicate with authenticators.
- **Client-side JS** – JavaScript code that runs within the browser and interacts with the WebAuthn API, authenticators and the Relying Party to implement registration and sign-in passkey flows. A native application can be written in any supported language.
- **WebAuthn API** – A browser-based software interface used to generate and use passkeys during registration and sign-in. If a native application is used, a platform-specific API, such as Apple WebKit or Google Credential Manager, can be used instead of WebAuthn.
- **Platform authentication API** – A platform- and hardware-specific software interface that enables the use of built-in authentication methods on the user's device, such as fingerprint or facial recognition, as well as biometric sensors and secure elements.
- **Platform authenticator** – A FIDO2 authenticator built into the operating system or firmware of the user's device. It typically includes a Trusted Platform Module (TPM) and biometric sensors.
- **Relying Party** – The online service that requires authentication to access its resources. It interacts with the browser using HTTPS.
- **Server application** – Runs on the Relying Party's server and receives authentication requests from the client platform.
- **Library APIs** – Software interfaces that allow developers to integrate authentication into their applications.

- **User and public key store** – A secure storage location for the user's private keys and the public keys associated with their passkeys.
- **Attestation verification** – The process of verifying the authenticity of a FIDO2 authenticator. It involves checking the attestation statement provided by the authenticator to ensure it was signed by a trusted party.
- **Attestation trust store** – A database of trusted attestation roots and intermediate certificates used to verify the authenticity of FIDO2 authenticators.
- **External metadata services** – Servers that provide additional information about FIDO2 authenticators, such as their capabilities and supported algorithms, to Relying Parties during the authentication process.

## 2.3 Standards

To implement passkeys, websites and remote applications use the W3C Web Authentication standard (WebAuthn), while client operating systems and passkey-sharing platforms use the FIDO Alliance FIDO2 Client to Authenticator Protocol 2 (CTAP2) API to access locally stored credentials. Each of these is described below.

### 2.3.1 WebAuthn

On the client side, support for WebAuthn [WebAuthn] can be implemented in several ways. The underlying cryptographic operations are performed by an authenticator, an abstract functional model that mostly does not depend on how the key material is managed. This enables implementation support for WebAuthn purely in software but also allows the use of hardware such as a Trusted Platform Module (TPM), the processor's Trusted Execution Environment, Windows Hello or a phone's authentication methods. Sensitive cryptographic operations can be offloaded to a roaming hardware authenticator, which in turn can be accessed via USB, Bluetooth or near-field communication (NFC). A fixed or roaming hardware authenticator is resistant to malware because the secret information (private key and biometric data) is not accessible to software running on the host machine. If biometric data or a secret such as a PIN is used, the corresponding data is not transmitted over the network, like service passwords. However, passkeys are not completely malware-proof; if the device does not have a physical display, malware on the client machine can change the origin of the authentication request and trick the user into confirming a challenge on a phishing site instead of a legitimate one.

WebAuthn comprises three elements:

- The authenticator is a FIDO2 authenticator.
- The web user agent browser or application with a compliant WebAuthn client is an intermediary between the authenticator and the WebAuthn Relying Party.
- The website, service, identity provider or authentication proxy is a compliant Relying Party.

Points to note with regard to certain functions and extensions include the following:

- The Enterprise Attestation (EA) function is not available to regular users and hardware authenticators because it involves the identification of authenticators that must be preloaded with traceable Enterprise Attestations.
- Cross-origin iframe support is used for combining web authentication and web payments.
- There are also a few extensions. Large blob storage allows the Relying Party to store opaque data associated with a credential on authenticators such as certificates rather than running a centralised

authentication service. Discoverable credentials can be created with 'discouraged', 'preferred' or 'required' user verification. FIDO AppID exclusion is a migration bridge for legacy U2F implementations where developers can also exclude FIDO U2F credentials that are otherwise largely compatible with WebAuthn. It also allows verification that a particular Android, iOS or web application is authorised for the target service of the passkey.

The main problem with basic WebAuthn is that the private keys remain on the secure hardware of the enrolled user's device. The frictionless sign-in provided by it is, therefore, tied to the device and the user has to re-enrol with the service after replacing or switching to a new device. If the user has multiple devices, each of them has to be registered separately with each Relying Party.

### 2.3.2 FIDO2 CTAP2

The FIDO Alliance and the W3C published the FIDO2 Web Authentication standard specification in 2018 [FIDO2]. FIDO2 enables users to use common devices to easily authenticate their identity for online services in both mobile and desktop environments. By adding Client to Authenticator Protocol 2 (CTAP2) to WebAuthn, FIDO2 enables a compliant cryptographic authenticator to interoperate with a WebAuthn client.

CTAP2 supports both single and multi-factor modes. In single-factor mode, the user activates the authenticator through a test of user presence, such as a simple button push on a device associated with the user and protected with a screen lock. In multi-factor mode, the user verifies their identity with a PIN, password or biometric data on the authenticator. Verification data is not shared with the website and a single biometric or PIN can work with all registered websites, as the authenticator selects the correct cryptographic key material for the service requesting authentication after the user approves its use.

The adoption of FIDO2 single-device authenticators by major browsers and operating systems was lagging after the FIDO2/WebAuthn specification became available. The reasons for this are the limitations of its authenticators, which are divided into two categories:

- **Roaming authenticators** are devices such as Yubico or Feitian keys that can be moved between different devices. The user must carry their roaming authenticator with them if they want to authenticate on other devices. If the physical device is lost, damaged or stolen, account recovery is time-consuming and tedious.
- **Platform authenticators** are hardwired to the device performing the authentication. Examples of platform authenticators include Windows Hello authentication via PIN, fingerprint or face on a Windows computer, fingerprint on an Android device, or Face ID and Touch ID on an iOS device. Authentication is tied to one device and is not shared across the user's PCs and smartphones. If the device is lost, account recovery restrictions apply, as with roaming authenticators.

A secret and a biometric on the authenticator can be used together, similar to how they would be used on a smartphone. For example, a fingerprint is used to provide convenient access to the smartphone; when fingerprint access occasionally fails, a PIN can be used instead.

# 3 Passkey Usage

A passkey is used to identify a user account with a particular service. It can be stored on a smartphone, tablet, PC or a password manager. The passkey may be shared among a set of devices or bound to a specific device. A passkey ensures a strong and private relationship between a person and a website or application. Typically, users have different passkeys for different services, unless these services are provided by the same Relying Party. However, it is possible to have multiple passkeys for the same site or account and the browser will display them for users to choose from. This enables users to assume different identities/roles, create backup passkeys and use passkeys on different devices that are later synchronised.

The passkey hides the mechanism used to verify the user, which is typically tied to biometrics or a PIN and a device or physical key. The mechanism on the device includes private user verification data, with the private key usually stored in a secure module of the user's device or a roaming hardware authenticator. Additionally, there is software tied to the operating system and ecosystem supporting passkeys, which provides usage and synchronisation across devices. This greatly increases convenience and ease of use, providing a higher level of reliability compared with using passwords. However, recovering a passkey if the user's last device is lost requires the passkey's private key to be stored in the cloud of the operating system or platform vendor. This, along with sharing among several devices, introduces security risks and may affect the level of assurance (LoA) of the passkey.

Even browsers that do not support passkeys by default can use credential management extensions that form their passkey-sharing ecosystems. For example, Dashlane integrates passkeys across Safari, Firefox, Chrome, Edge and (soon) mobile devices. OS-integrated or standalone password managers, also known as credential management applications, are evolving rapidly to manage passkeys in a user-friendly way, linking web browsers, devices, user identities and services with passkeys.

This section discusses the use of passkeys during registration, authentication and sharing across devices, and support by password managers.

## 3.1 Registration

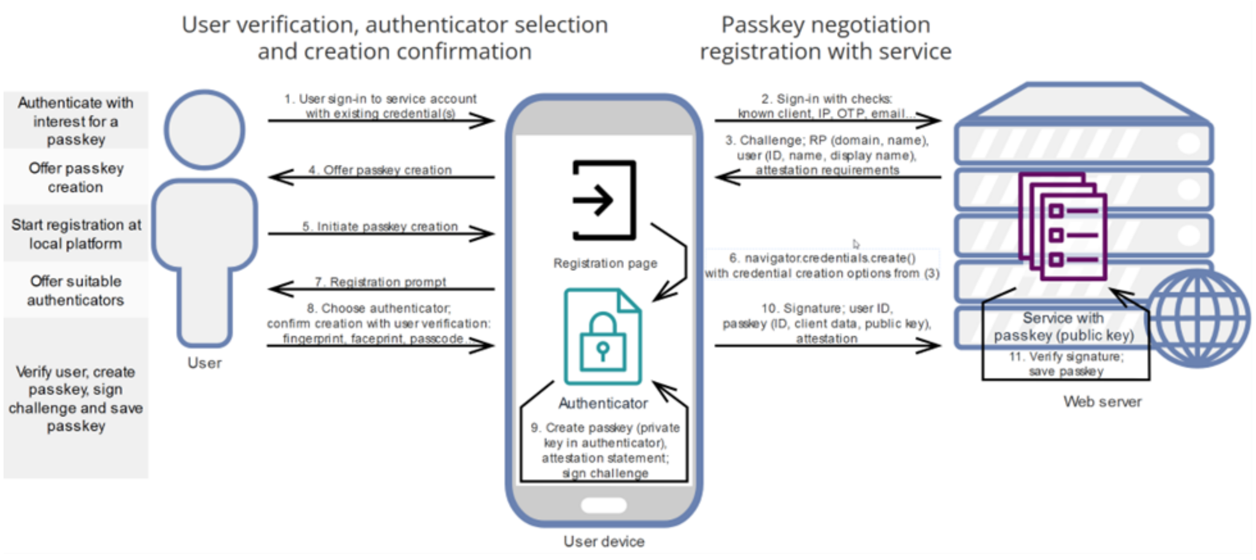

User verification, authenticator selection and creation confirmation
Passkey negotiation registration with service
Authenticate with interest for a passkey
Offer passkey creation
Start registration at local platform
Offer suitable authenticators
Verify user, create passkey, sign challenge and save passkey
User
1. User sign-in to service account with existing credential(s)
4. Offer passkey creation
5. Initiate passkey creation
7. Registration prompt
8. Choose authenticator; confirm creation with user verification: fingerprint, faceprint, passcode...
Registration page
Authenticator
9. Create passkey (private key in authenticator), attestation statement; sign challenge
User device
2. Sign-in with checks: known client, IP, OTP, email...
3. Challenge; RP (domain, name), user (ID, name, display name), attestation requirements
6. navigator.credentials.create() with credential creation options from (3)
10. Signature; user ID, passkey (ID, client data, public key), attestation
Service with passkey (public key)
11. Verify signature; save passkey
Web server

Figure 3.1: Registration flow

When creating an account for a website or other resource provider that supports passkeys, the user undergoes a registration procedure and is offered the option to create a passkey or sign in with a security device. A passkey is generated on the user's device specifically for that website. The public key is sent to the website's server for storage, while the private key remains securely stored in the device's authenticator.

The private cryptographic key of a passkey stays on the user's device (such as a mobile phone or laptop) or in the credential store of a detachable secure key. It is safeguarded by biometrics and hardware-based security, if present and used. The associated public key is stored by the accessed service on the application server or by the application itself, while the private key is never sent to the service.

## 3.2 Authentication

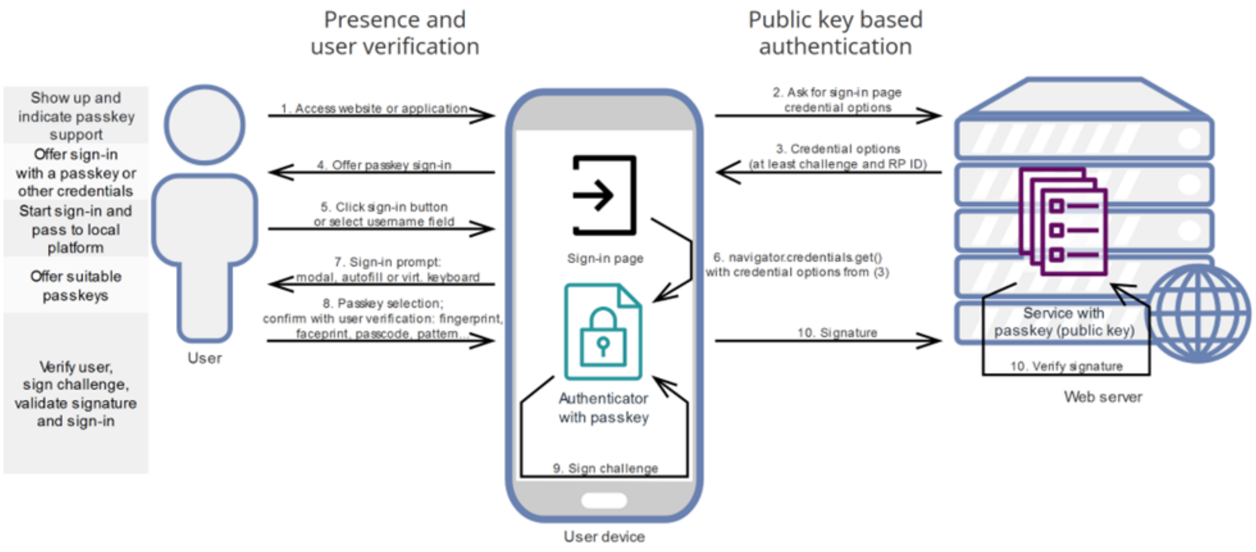

Presence and user verification
Public key based authentication
Show up and indicate passkey support
Offer sign-in with a passkey or other credentials
Start sign-in and pass to local platform
Offer suitable passkeys
Verify user, sign challenge, validate signature and sign-in
User
1. Access website or application
4. Offer passkey sign-in
5. Click sign-in button or select username field
7. Sign-in prompt: modal, autofill or virt. keyboard
8. Passkey selection; confirm with user verification: fingerprint, faceprint, passcode, pattern...
Sign-in page
Authenticator with passkey
9. Sign challenge
User device
2. Ask for sign-in page credential options
3. Credential options (at least challenge and RP ID)
6. navigator.credentials.get() with credential options from (3)
10. Signature
Service with passkey (public key)
10. Verify signature
Web server

Figure 3.2: Sign-in flow

When the registered user visits the service (1), the service shares some information about the website or remote application with their browser or client application (2, 3). The server also creates and sends a randomly generated challenge. The browser, operating system or password manager offers the user the option to use passkeys (4, 5) and select one from among those that are applicable (6, 7, 8), similar to choosing a saved password. To ensure that only the rightful owner can use a passkey, the system prompts the user to unlock their device (8). By unlocking the device's screen, they confirm their identity, preventing login from a stolen device. This can be achieved through biometric sensors (e.g., fingerprint or facial recognition), PIN or pattern, typically using Apple Face ID and Touch ID, Android face and fingerprint unlock mechanisms or Windows Hello. The identity confirmation allows the use of the private key bound to the service to sign its challenge (9). Finally, the application or website uses the public key obtained during passkey registration to verify the signature and confirm the passkey and associated user identity (10). This signature can only come from one of the devices on which the user is able to confirm their identity, and with a corresponding private key in their secure storage. If the signature is valid, the authentication process is successful and the user is granted access to the service.

Registration and authentication flows share a high degree of similarity, adhering to identical sequences of steps that exhibit minor variations in content and meaning. This not only streamlines implementation but also underscores their fundamental similarity and indicates conceptual coherence.

## 3.3 Use Across Devices

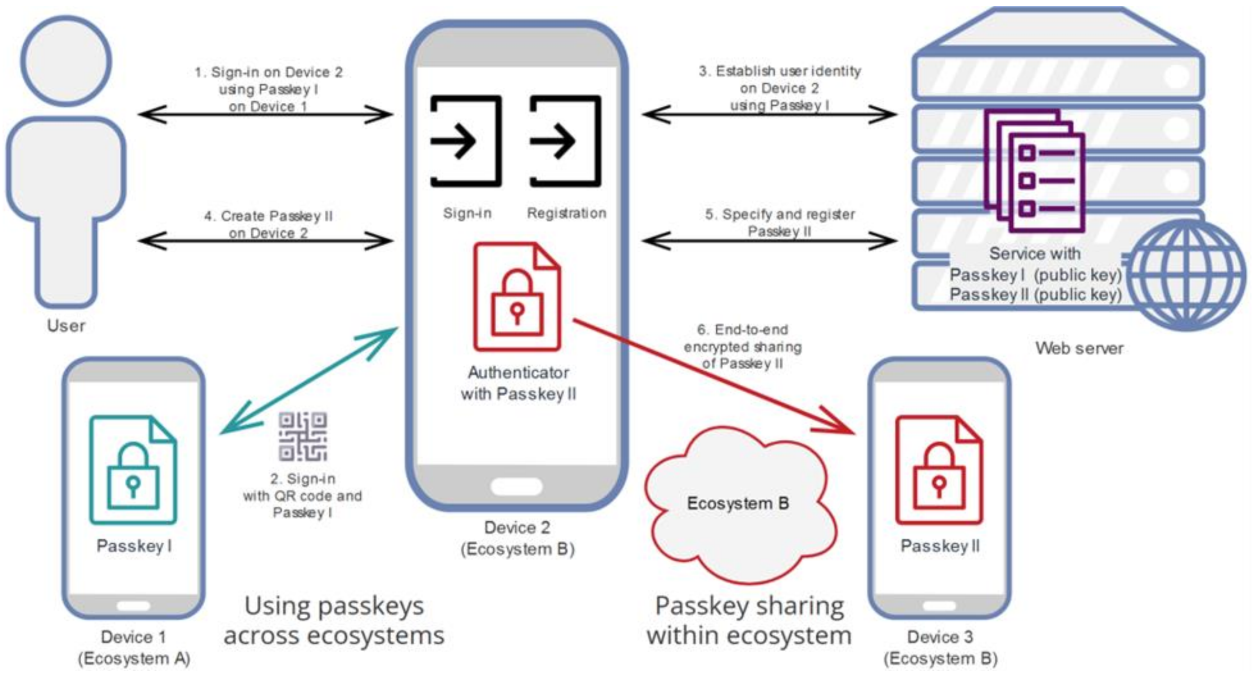

1. Sign-in on Device 2 using Passkey I on Device 1
3. Establish user identity on Device 2 using Passkey I
4. Create Passkey II on Device 2
5. Specify and register Passkey II
User
Sign-in
Registration
Authenticator with Passkey II
Service with Passkey I (public key) Passkey II (public key)
Web server
6. End-to-end encrypted sharing of Passkey II
2. Sign-in with QR code and Passkey I
Passkey I
Device 1 (Ecosystem A)
Device 2 (Ecosystem B)
Ecosystem B
Passkey II
Device 3 (Ecosystem B)
Using passkeys across ecosystems
Passkey sharing within ecosystem

Figure 3.3: Using an existing passkey to sign in on Device 2, followed by the creation of a passkey for its Ecosystem B

Alternatively, on a Linux, Windows or macOS computer (or potentially another smartphone), the user can sign in to a website or service using a passkey stored on their mobile device. The user has previously logged in to the website on the signing device and created a passkey on it, and this passkey can be used to sign in to a website on a computer, such as an Apple laptop (1). This cross-device and cross-platform passkey authentication is standardised by FIDO. On the sign-in prompt, the user chooses to use their passkey on the smartphone (2). For example, Safari on a Mac will display a QR code that the user can scan with an Android phone. The smartphone does not need to be online. Bluetooth, however, plays a key role in accessing the passkey from a device in a different ecosystem. The two devices use Bluetooth to ensure they are physically close to each other, preventing phishing. After a sign-in prompt is shown on the Android device, the user verifies themselves and confirms the sign-in by providing a secret or biometric data to unlock the phone's screen and then selecting an applicable passkey (2). The authenticator on the sign-in device signs the challenge with the passkey's private key. A one-time signature for the challenge is sent back to the laptop via an end-to-end encrypted tunnel, which the browser then forwards to the service (3). This form of sign-in is even suitable for signing in from another person's or a shared computer.

After a successful login, if the devices used to sign in and access the service are different, the service will usually offer to create a new passkey for the device accessing the service (4). This way, the smartphone is not needed the next time the user wants to sign in. In the example above, a new passkey would be created for the laptop. Since it is an Apple device, the created passkey will be shared with all other devices the user has in the Apple ecosystem (5). However, they should not create a new passkey if they do not own that laptop or if others know its PIN or password and can access it along with the new passkey.

Additionally, the computer on which the QR code was scanned can be remembered on the sign-in device, so the user can avoid scanning the QR code in the future. If this is done, their mobile device will be offered as an option the next time they need to use passkeys on their laptop. When selected, their smartphone receives a notification and displays a sign-in prompt for the passkeys applicable to the accessed website. In this case, their phone is used similarly to a pluggable security key, with the difference that they verify themselves and use their passkey on their smartphone.

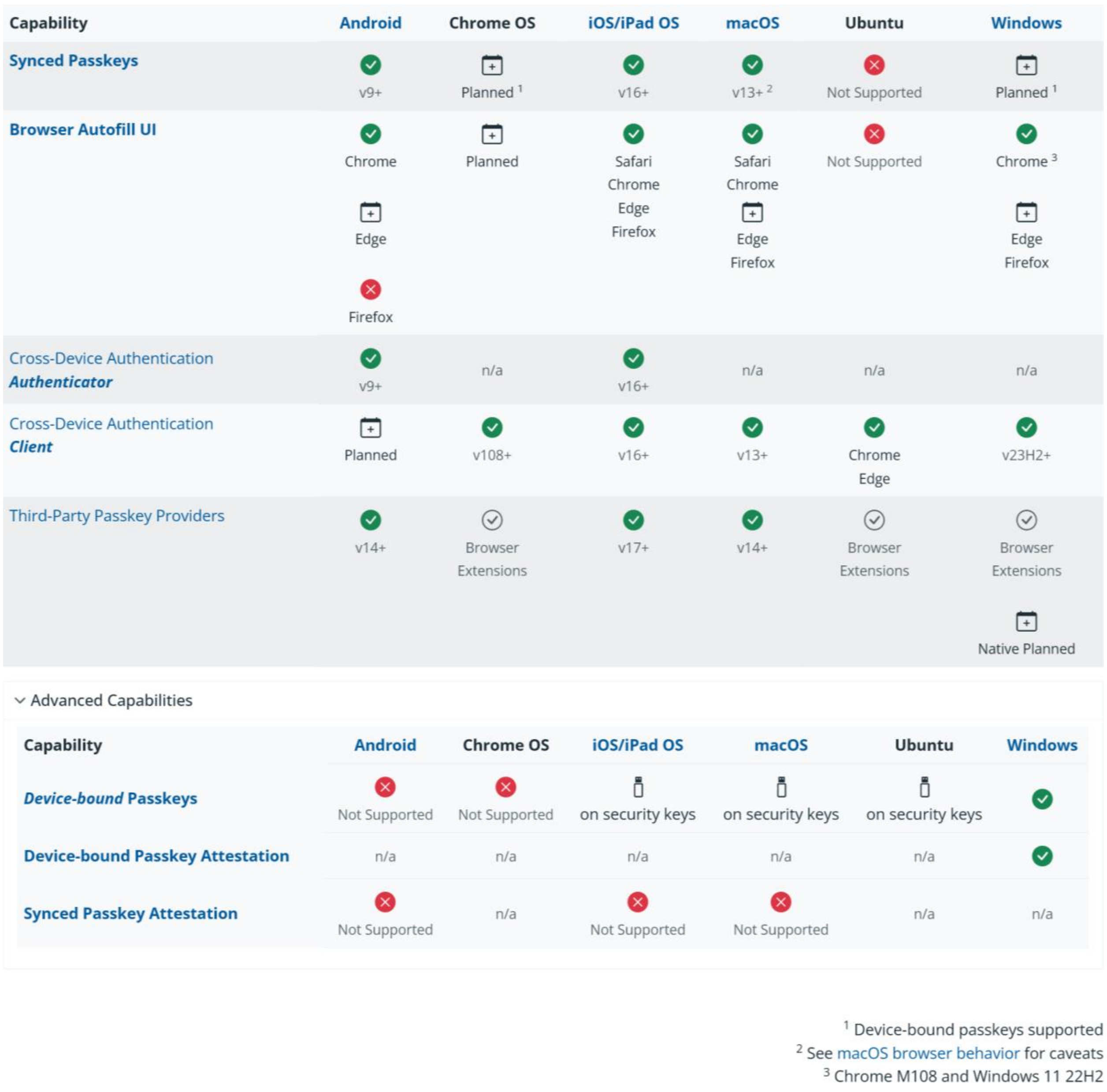
It is important to note that when using a QR code and Bluetooth, the passkey is not transmitted to the laptop. The FIDO2 Cross-Device Authentication (CDA) flow, which leverages CTAP 2.2, uses Bluetooth Low Energy (BLE) to verify physical proximity but does not depend on Bluetooth security to secure the sign-in. It uses its own security but also does not exchange the passkey or its private key between the devices. Furthermore, the QR code contains only information that sets up a secure tunnel between the device displaying the QR code and the device used for signing in. It does not include the challenge or passkey information from the Relying Party or authenticator.

The user can also use a device with a security key that has been registered with the service in a similar flow. The user visits the website on their computer and sees a 'Sign in with a passkey' button or autofill UI. They select or use their security key and when the browser prompts them for authentication, the user can plug in their security key device and unlock it with biometric data or a PIN.

All the flows described above are mediated by the browser and OS platform, so service developers do not need to know about QR codes, Bluetooth or CTAP. These interactions are demonstrated and described in a demonstration video [PkInAction].

Passkeys are synchronised across all devices that share an ecosystem account. Passkeys created in Safari on Apple devices and in Chrome on iOS 16 and iPadOS are stored and shared via iCloud Keychain [PkIphone]. On Apple devices, users can share passkeys with others via AirDrop. When receiving passkeys in this way, recipients must verify themselves with the receiving authenticator. Passkeys created in Chrome on Android or Chromebook are stored in Google Password Manager. Shared passkeys are also available in other browsers and apps within these ecosystems. When shared, passkeys are encrypted end-to-end and their private keys are secured locally in device vaults. However, Chrome on Windows stores passkeys in Windows Hello, while on macOS and Linux, it stores them in a local profile that is not synchronised with other devices. Passkey platform providers are currently unclear about whether they track user sign-ins with passkeys or track site registrations when passkeys are created.

For more detailed information about support on different platforms and with different browsers, go to the passkeys.dev website [PkDevSuprt]. The following is a summary table from passkeys.dev, as of 9 November 2023:



| Capability | Android | Chrome OS | iOS/iPad OS | macOS | Ubuntu | Windows |
|---|---|---|---|---|---|---|
| **Synced Passkeys** | ✓ v9+ | ⊞ Planned [1] | ✓ v16+ | ✓ v13+ [2] | ✗ Not Supported | ⊞ Planned [1] |
| **Browser Autofill UI** | ✓ Chrome<br>⊞ Edge<br>✗ Firefox | ⊞ Planned | ✓ Safari<br>Chrome<br>Edge<br>Firefox | ✓ Safari<br>Chrome<br>⊞ Edge<br>Firefox | ✗ Not Supported | ✓ Chrome [3]<br>⊞ Edge<br>Firefox |
| Cross-Device Authentication ***Authenticator*** | ✓ v9+ | n/a | ✓ v16+ | n/a | n/a | n/a |
| Cross-Device Authentication ***Client*** | ⊞ Planned | ✓ v108+ | ✓ v16+ | ✓ v13+ | ✓ Chrome<br>Edge | ✓ v23H2+ |
| Third-Party Passkey Providers | ✓ v14+ | ✓ Browser Extensions | ✓ v17+ | ✓ v14+ | ✓ Browser Extensions | ✓ Browser Extensions<br>⊞ Native Planned |

Advanced Capabilities

| Capability | Android | Chrome OS | iOS/iPad OS | macOS | Ubuntu | Windows |
|---|---|---|---|---|---|---|
| ***Device-bound* Passkeys** | ✗ Not Supported | ✗ Not Supported | 🔑 on security keys | 🔑 on security keys | 🔑 on security keys | ✓ |
| **Device-bound Passkey Attestation** | n/a | n/a | n/a | n/a | n/a | ✓ |
| **Synced Passkey Attestation** | ✗ Not Supported | n/a | ✗ Not Supported | ✗ Not Supported | n/a | n/a |

[1] Device-bound passkeys supported
[2] See macOS browser behavior for caveats
[3] Chrome M108 and Windows 11 22H2

Table 3.1: Passkey support on different platforms

For more information on Chrome and Android passkey support, visit [PkCASuprt]. It should also be noted that Firefox currently supports operating system-managed passkeys on Windows 11, Android and iOS, but with autofill only on iOS. On macOS, it only works with passkeys on security keys. Edge supports autofill on iOS only, while on macOS, it generates device-bound passkeys exclusively accessible on that platform.

## 3.4 Support by Password Managers

To avoid platform lock-in, it is essential to have independent passkey solutions widely adopted in the market and seamlessly integrated with various operating systems and browsers. Password managers such as Dashlane [PkDash], 1Password [1pPk], Keeper [KpPk], Bitwarden [BwPk] or LastPass [LPPk] plan to offer passkey integration as a standard feature in 2023.

For Linux users, passkey usage in standalone client applications is inconsistent due to the lack of a unifying passkey management platform and fragmented browser use. However, password managers and browsers could provide consistent and reliable passkey solutions that would fully embrace Linux.

# 4 Implementation

There are two ways to implement passkey sign-in: using the ‘Sign in with passkey’ button, and by autofill. Each of these is described below, together with other important aspects of passkey implementation: reauthentication; passkey selection, accounts and user data; decommissioning; backup, loss and recovery; and deployment considerations.

## 4.1 Authentication with the ‘Sign in with a passkey’ Button

Passkeys enable sign-in without forms, but with a few screen unlock taps. When a website or application has only passkey users and does not have to handle the legacy of passwords, it can place a ‘Sign in with a passkey’ button or UI element to allow the user to initiate the sign-in. When the user clicks on it, the browser or operating system displays a modal selector with the names of the applicable passkeys. Consequently, this is also known as a ‘modal UI’. The user chooses an account and unlocks the device screen for verification.

## 4.2 Form-Based Autofill Authentication

If a service has users without a passkey, it still has to let the user sign in using other methods, such as a username and password. Passkey autofill, also referred to as ‘conditional UI’, is suitable for the transition from password-based or multi-factor authentication to authentication with passkeys. If there are some users with passwords, the website or application must still offer a sign-in form with username and password fields. With autofill, the user is automatically shown suggestions for suitable passwords and passkeys. This way, the user does not have to remember whether they use a passkey or a password.

With this approach, the user sees an account selector upon starting an interaction with the login form. If the account is password-based, the autofill provides a username and password. If the account is passkey-based, the user is immediately prompted to unlock the device to sign in.

To use a passkey with security keys, autofill should be implemented in conjunction with ‘Sign in with a passkey’, as autofill is not allowed to query a security key for available passkeys.

For a good guide on implementing sign-in with passkeys in web applications, refer to [PkAutofill].

## 4.3 Reauthentication

Reauthentication is a common situation where the user is already logged in but needs additional authentication because a session has expired or because the user wants to perform a sensitive operation, such as adding a shipping address or finalising a transaction. With password-based authentication, the user is prompted to enter the password to reauthenticate. With passkeys, the application can simply prompt the user to unlock the device again.

## 4.4 Passkey Selection, Accounts and User Data

It is critical to indicate which website a passkey is for so that the user knows where they are signing in, regardless of what the website states during registration and sign-in. If there is only one appropriate passkey to sign in with and the website address is visible to the user, it can simply be rendered by the browser as 'Passkey for this site'. In implementations, the website information is enriched with the domain string and the name of the Relying Party from the passkey data, and even the website's favicon. The passkey selection display also includes additional details such as the name of the account otherwise associated with the website (e.g., the one associated with the password used during passkey registration) and the user account with the credential management platform.

When the user is in the username field, the smartphone virtual keyboard may offer applicable passkeys or a switch to a richer selection interface. In the autofill interface, clicking on the username displays a list of credentials, including passkeys, passwords and other remembered devices and secure keys. This list appears as soon as the username field receives input focus. After a passkey is selected, the username field remains empty.

With both interfaces, all applicable credentials are listed and the options may include using a passkey on another device by scanning a QR code or using a security key.

The offered passkeys may be associated with different user accounts. A user account may be accessed by one or several users and one natural person may have access to one or more user accounts, depending on the user's actions, the policy of the Relying Party and its enforcement during registration. The user may not always be the same natural person if multiple people share access to the same authenticator. The ability to distinguish between natural persons depends on the biometric capabilities of the platform and the authenticator. For example, even devices intended for a single individual's use may be used by several natural persons by enrolling several faces or fingerprints, or sharing the PIN.

A Relying Party must support associating multiple passkeys with one account, as different user devices may use different passkeys. If these passkeys are shared or copied, they may end up in a single authenticator.

The user's name in a passkey is typically the name of their account, while the display name represents the person's name. Both are specified during registration and can contain any information provided by the user and the Relying Party. These names need to be clear to the user so that they can select the desired one for signing in to the website. The Relying Party is responsible for setting the name and display name, and the authenticators must store them with the passkey. The name and display name are used for local display and search by the user and are not utilised for account matching on the server side or returned to the Relying Party during sign-in. Instead, the passkey's user handle is employed to identify the associated user account. The user handle is generated during registration and is not intended to be presented to the user. Some platforms may also offer the option for users to add private notes about the account or website, but these notes are not an integral part of a passkey.

By limiting the user data included in the passkey, only necessary information is communicated to display and select a passkey on the client side. Even if the passkey's username or display name contains personally identifying information, such as a name or email address, this data is transmitted in encrypted form, stored locally, and displayed securely and only when necessary. The data used to provide service representation, context and user preferences, as well as the content of their accounts, remains with the service and is not exposed through the passkey's content.

While browsers and platforms facilitate the use of passkeys by providing a user interface for autofill and modal passkey selection, WebAuthn does not allow services to silently discover whether the user possesses WebAuthn credentials. Instead, services must explicitly prompt the user to determine if they have WebAuthn credentials.

To ensure user privacy and prevent tracking, the WebAuthn specification stipulates the minimum number and distribution of user devices that can be recognised as different makes or series of authenticators and by their attestation certificates. The registration and authentication flows must be implemented in a way that minimises the possibility of malicious Relying Parties or third parties identifying users without their consent and actual sign-in, such as through the characteristics or capabilities of the authenticator and available passkeys. Relying Parties must also prevent leakage of information about existing accounts and passkeys through the details revealed in and/or timing of their responses, and related use of other communication channels, such as email verification.

## 4.5 Decommissioning

There may be situations that require a passkey to be decommissioned:

- If a passkey is lost or compromised, the user should inform the service by signing in using alternative means and accessing a page that lists their registered credentials, including creation dates and names set during registration. The user can then select the passkey to be deleted, which will subsequently be removed from the service's database. This action must be taken if the passkey is secured with a shared or leaked PIN or if the user suspects that the authenticator or passkey may have been compromised in some other way.
- The service may also remove an unused passkey from its database during maintenance and cleanup. If the user wishes to resume using the service, they will need to re-register, create a new passkey and update their account information accordingly.
- If a user wishes to delete a passkey from all their devices and passkey-sharing platforms, they can use platform-specific methods. This action removes the passkey from future sign-in prompts for the service. Since the user's private key is lost in this process, there is no need for the user or the platform to inform the service that a passkey has been deleted unless the user suspects that it may still be available to someone else. The service may later deregister the passkey deleted this way due to inactivity.

## 4.6 Backup, Loss and Recovery

A common scenario involving a lost, damaged, stolen, reset or upgraded device is addressed by sharing the same private key across multiple user devices. This sharing is enabled by the passkey platform's synchronisation and backup capabilities. It is particularly useful when users own multiple devices, lose a device or transition to a new one. However, the PIN or biometric data used to access the passkey is stored only on each device's secure storage along with the passkey's private key. The passkey's private key is shared between the user's devices through end-to-end encryption. Users should treat devices that share passkeys as mutual backups.

As end-to-end encrypted synchronisation of passkeys is handled by the passkey-sharing platform, some ecosystems may store the passkeys not only on users' devices but also in encrypted form within their passkey management service. Such backups typically involve separate storage of passkey recovery keys and an elaborate recovery procedure, but this still raises questions regarding the privacy of shared passkeys. This backup simplifies recovery, but it is an option that users should be allowed not to use as it introduces additional risks. Users should pay attention to whether this option is available in the ecosystem they are using and whether it is activated by default.

If the user previously declined to synchronise a passkey with other devices and has lost the device where it resides, or has lost all devices with the shared passkey and the platform backup is not available, then they must register a new passkey with the service. The service should treat the subsequent access from a new device (which may be in a different ecosystem) as a standard account recovery and passkey creation situation and take appropriate additional steps to validate the user. However, if a passkey for the service is still available in another ecosystem or device, then the user can sign in on a new device using that device and then recreate a passkey without going through the full registration procedure.

In the case of the user's only Android device being lost, damaged or stolen, the user can recover the keys on a new replacement device by recovering the end-to-end encryption keys from a secure online backup by the platform. If these keys were not already transferred when the new device was set up, the recovery process will happen automatically when a passkey is created or used on that device for the first time [AndrRecovery]. During this recovery, the user is prompted for the old device's screen lock PIN to restore the end-to-end encryption keys and then for the screen lock of the current device to use the passkey. This is possible since the data used to verify the old device's screen lock PIN input is 'stored on Google's servers in secure hardware enclaves and cannot be read by Google or any other entity. The secure hardware also enforces the limits on maximum guesses, which cannot exceed 10 attempts'. So, although the data used to unlock devices and passkeys is not shared with Relying Parties, it is shared with Google and then protected by it. However, if a passkey on Android uses a device-bound Public Key WebAuthn extension (devicePubKey), only that device can use its corresponding private key. This passkey will have a device-bound private key and the corresponding public key at the Relying Party. It cannot be backed up in any way. A new device-bound key pair will have to be recreated after resetting or restoring the device from backup.

In Apple's ecosystem [AppleRecovery], if all devices are lost, users can recover their passkeys via the iCloud Keychain escrow. The escrow records are protected against brute-force attacks, accidental exposure or misuse with Hardware Security Modules (HSMs) and an elaborate interactive multi-channel user verification and keychain decryption process. Passkeys in iCloud Keychain are encrypted end-to-end, ensuring that even Apple cannot read them.

In scenarios where the user has lost access to all their synchronised passkey devices but still possesses a security key, that key can serve as a recovery credential.

If a user wants to switch between ecosystems or simply create a passkey for another ecosystem and still has their old device, then similarly to the scenario described at the top of the page they can use the passkey on the old device (e.g., an Android device) to sign in to their account on the new device (e.g., an iOS device). Once signed in, the user can create a passkey associated with their account on the new platform.

## 4.7 Deployment Considerations

A final implementation aspect, for site administrators, concerns deployment considerations. Passkeys are, by default, associated with the domain name of the website where they were created. Therefore, both registration and authentication must occur under the same domain name. For instance, the registration page could be at **example.org/registration**, and the authentication page at **example.org/authentication**, both under the **example.org** domain. Alternatively, the registration page can reside on a subdomain of the authentication domain. For instance, the registration page could be at **registration.example.org/page**, while authentication might take place at **example.com/auth**. In this case, the registration page must explicitly set the passkey's origin to **example.com** to prevent assuming **registration.example.org**. This ensures that the created passkey can be discovered during authentication.

# 5 Supporting IdP Products

This section provides a summary of selected resources and tutorials available for implementing passkeys in web and mobile applications, together with an overview of the passkey support provided in existing software identity provider (IdP) products.

For a comprehensive guide on implementing passkey-based sign-in in web applications, refer to the Yubico Passkey Autofill example at [PkAutofill]. This example provides both a comprehensive guide and a concise sign-in illustration [PkAfFlow].

Apple's tutorial covers both Apple WebKit and web HTML/JS programming [MeetPk].

Google offers resources at [GooDevPkLogin] with implementation guides for both web and Android applications. The Android Credential Manager API (androidx.credentials) [AndrJpCred] enables users to create and store passkeys within Google Password Manager. This library manages various credentials, including passkeys, passwords, federated identities, multi-factor authentication tokens and OAuth tokens. It leverages the features of the Android Keystore system for credential storage and provides access to biometric authentication. However, using passkeys is relatively straightforward and does not necessitate direct access to these features. Another Google tutorial [GooglmplAfTut] presents a comprehensive example covering different aspects of passkey-related web UI, including listing, registration, deletion and usage.

The WebAuthn.me debugger is a tool for developers to experiment with passkey registration and authentication parameters [WADebugger]. It is useful for testing various passkey registration and authentication parameters.

Server-side passkey solutions can be implemented at both the application platform's authentication and authorisation infrastructure (AAI) level and the application level.

Apart from passwordless solutions available within IdP software listed in the following subsections, libraries and standalone solutions exist for directly integrating passkeys into applications. Open-source libraries from Yubico [YubDido2WALib], Duo [DLPyWA] and others [WAIO] are available.

When developing a mobile application, several options exist for managing passwordless login. The simplest approach is to use web-based authentication, such as an existing OIDC IdP with passkey support. In such cases, minimal adjustments are needed within the mobile application, as passkeys are managed externally.

Alternatively, proprietary APIs provided by operating system vendors such as Microsoft, Apple and Google can be employed to handle passwordless authentication within the application.

Another approach to web and native application development is to utilise an existing software IdP product that supports passkeys. This approach shields developers from the direct use of HTML/JS or Apple WebKit and enables them to leverage one of the standard identity and access management protocols, the product's built-in components or a custom API. Some of the available solutions also include their own supporting backend components. Hanko [Hanko] consists of a backend with an authentication API, user management, JWT issuance, and support for passkeys, passcodes and passwords. It also provides web components for customisable onboarding and login to the backend, along with a client package for interacting with the API. The Quiq WebAuthn Proxy [QqWAProxy] is a WebAuthn module designed for integration into service deployments using DevOps tools such as Docker and Ansible. It operates behind common reverse proxies such as NGINX or OpenResty and possibly in conjunction with OAuth2 proxies. Passkey support across various popular software IdP products is described below, while the experience of using SimpleSAMLphp and privacyIDEA products is detailed in Appendix A.

## 5.1 Microsoft Entra ID (formerly Azure Active Directory)

Microsoft's IdP, now Entra ID, enables passwordless login using generic FIDO2 security keys [MSFIDOKeys]. This functionality requires pairing with Azure AD Multi-Factor Authentication. Users must register at least one Azure AD Multi-Factor Authentication method before adding FIDO2 security keys to their Microsoft profile page [MSProfile].

## 5.2 Keycloak

The passwordless sign-in can be enabled within the Keycloak authentication flow [KcWATut], alongside password login during the migration to these methods. An unresolved issue exists concerning passwordless sign-in as the default method [KcCredOrd]. Consequently, a password login form is displayed, prompting users to click 'Try another way' to switch to passkeys.

## 5.3 Shibboleth

Shibboleth IdP v4 does not inherently support passkeys at present, although it is likely to be on the roadmap [ShWAPlug]. Some third-party options are available solely for using WebAuthn as a second factor. Duke University has incorporated WebAuthn support into its Shibboleth build [DUShWA], and a plugin for connecting to privacyIDEA is also available [ShIDEA2FA].

## 5.4 SimpleSAMLphp

Passkeys are supported in a SimpleSAMLphp-based IdP through its WebAuthn as Second Factor module [SSphpWAMod]. This module extends SSP by providing a registration page where users can manage their passkeys and an authentication source that replaces password-based login with a passkey sign-in page. If enabled, users can also manage their passkeys during each sign-in.

Additionally, there is a plugin for connecting to privacyIDEA. At the time of writing, that plug-in for SimpleSAMLphp does not support passkey login, only using WebAuthn as the second factor.

## 5.5 privacyIDEA

privacyIDEA supports WebAuthn alongside MFA and various other authentication methods. However, sign-in with passkeys without prior user identification is not yet possible [pIDEAWAUser]. This feature is planned to be included in the March 2024 privacyIDEA milestone as part of generalised support for discoverable credentials and usernameless authentication.

# 6 Implementation for R&E

In addition to the passkey implementation factors described in Section 4, several operational considerations need to be taken into account when deploying passkeys for research and education (R&E) services.

Apart from configuring and implementing passkeys in software, service providers and IdPs should also be prepared for an increased demand for user support. This may result in a temporary surge in the workload for help desk personnel, added complexity in the support process, and potential user dissatisfaction. Passkey authentication may initially lead to a higher number of failed login attempts and requests for credential recovery. While it is relatively straightforward to verify whether a user is entering the correct password or to send a password reset link, guiding users through various passkey interfaces, explaining the registration with a new authenticator and advising them on the appropriate passkey, authenticator and user verification method can prove significantly more challenging.

For passkeys to be accepted in a federated environment, they may need to be recognised and implemented as a reliable user authentication method following the applicable trust guidelines across all participating organisations. One such framework is provided by the REFEDS Single-Factor Authentication (SFA) profile, and its relationship to passkeys is discussed in Appendix B.

The following subsections discuss potential methods for implementing passkeys in software, and transition and adoption challenges.

## 6.1 Implementing Passkeys from Scratch

Individual services that do not rely on an external identity provider or federated login, and those that require additional local authentication alongside the one provided by the external identity provider or federation, could implement passkeys as any passkey-aware web application.

Similarly, federation operators implementing passkeys from scratch need to implement passkeys using WebAuthn API or by integrating one of the supporting open-source technologies such as Shibboleth IdP and SimpleSAMLphp, as described in Appendix A.

## 6.2 With WebAuthn API Already Implemented

For organisations already implementing the WebAuthn API, a small change is needed to implement passkeys, particularly in the sign-in user interface. Applications should be updated to initiate sign-ins using WebAuthn browser support. Additionally, a separate passkey registration UI needs to be provided.

When integrating passkeys into the default password-based login page, there are a few key design decisions to consider. One option is to use autofill to incorporate passkeys into the existing login page. Another option is to add a 'Sign in with a passkey' button and use it to activate a modal UI (see Section 4 for further information). During the transition period, the first option would be the preferred choice. However, not all browsers currently fully support it. Multiple sign-in pages could also be offered, allowing users to switch between them.

Initially, the default page should include both password-based login and passkey creation and use. Once the majority of users have created their passkeys, signing in with passkeys should be the primary login method offered. Trying to predict on the server side whether a user has a passkey on their current device and is likely to use it based on passkey registration and sign-in history can be challenging and confusing for users. It is better to rely on an autofill form, as it allows the device to offer an applicable passkey if one is available.

## 6.3 Federated Implementation and Usage

In a decentralised identity federation, the IdP should offer passkey-based sign-in in its login form. Its web UI conducts passkey registration and management flows. This sign-in frontend not only manages the aforementioned flows but can also accommodate MFA, step-up scenarios, or other custom needs of the IdP. IdPs can independently introduce passkeys, irrespective of whether the federation recognises passkeys or not. Nevertheless, regardless of federation recognition, IdPs must adhere to established federation rules, such as those defined in a dedicated authentication profile like the REFEDS Single-Factor Authentication (SFA) profile, and handle passkey registration and management accordingly.

For a centralised identity federation, the implementation of passkey registration, sign-in and passkey management workflows must take place at its single point of authentication. During passkey use and management, the central IdP can consult local IdPs using protocols supported by their databases, while keeping information about created passkeys and associated IdPs and accounts. As it needs to keep information for all actively used passkeys and associated accounts from all its IdPs, this database of passkeys may grow substantially. The federation operator may need to establish a process to periodically review and remove unnecessary entries by querying IdPs and purging passkeys for deleted identities, or by deleting passkey information for accounts rejected by their IdPs during authentication, as well as for passkeys that have not been used recently. However, the removal of unused passkeys may inconvenience occasional users and increase the user support workload as users would need to recreate their passkeys after failing to log in.

In both cases, relying services continue to use their existing SAML or OIDC interfaces. The addition of passkey support to an IdP or their identity federation does not impact services, as their authentication process remains unchanged. Once the IdP or federation begins supporting passkeys, the connected service will seamlessly incorporate this authentication method without necessitating any internal adjustments. Nevertheless, the federation or IdP may opt to include additional passkey-related attributes to convey information about their use, user verification or other WebAuthn flags. If a service needs to utilise these attributes provided by the IdP or the central point of authentication, it must be adapted to read them as it would for any other available attributes. Furthermore, the intervening SP proxies or IdP discovery services placed between SPs and web interfaces for authentication have no impact on implementation and use of passkeys.

## 6.4 Transition and Adoption Challenges

The primary challenges and questions facing R&E organisations transitioning to and adopting passkeys may be summarised as follows:

- Passwords will have to remain available as an option despite their weaknesses for the short to mid term because not all user devices have passkeys. This leads to postponing the decision of when to delete them, which could result in a prolonged transition.
- Unlike passwords, where individuals can, at least in theory, manage their credentials manually or use independent password managers, tighter integration of passkeys with hardware by a few operating system and browser providers could accelerate their rise into controllers of the infrastructure for accessing virtually all digital services. However, if third-party providers become more present, the major ecosystems will become more user-centric and equitable.

- Aggressive promotion or enforcement of passkeys by large providers could lead to resistance in some communities, yet at the same time it may also significantly improve user uptake of passkeys.
- All relevant identity providers need to update registration and authentication to support passkeys, which will be a daunting task.
- There will be a long tail of users and services that are slow to adopt passkeys. Linux users, in particular, have no single vendor offering and integrating passkeys for their varied hardware and many Linux distributions.
- Identity providers and federations must shift from directly managing user credentials to ensuring the integrity and validity of user-stored and managed passkeys. This involves proper registration and authorisation through new workflows that start with identity verification and self-registration, or involve authentication without initial user identification but require the user to choose from the available and applicable discoverable passkeys. The subsequent validation of a user's status as a student or employee remains unchanged.
- Passkeys are not shared across different operating system, vendor or browser ecosystems, so users need to create a new passkey for each ecosystem and service they use. This fragmentation diminishes the user experience and reduces the user’s control over their passkeys.
- Should identity providers and federations be allowed to use one passkey for all services that rely on them?
- Should R&E identity providers and federations establish their passkey ecosystems?
- Some special but not rare use cases are already essentially supported but are not yet clearly accessible with passkeys or are incompatible with existing practices. These are related to the separation of user identities in services from those in passkey sharing platforms, such as spouses using one account, co-workers sharing access to resources, anonymised users, separate private and business accounts, users with multiple personas for one service, and high-security scenarios. For example, when a user is using a shared account on a passkey-sharing platform (such as Google), they cannot create a private passkey for their bank account because all passkeys are automatically shared with all people who have access to devices linked to that account.
- Passkey platforms should increase transparency about what they track, record or do in the name of availability and usability.
- There is a possibility of fragmentation between platforms through extensions and platform-specific features and conventions if their development and adoption are not accompanied by additional harmonisation and standardisation.
- There are a number of perception and trust issues, including:
    - Considering passkeys only as a matter for big companies.
    - Fear that passkey-sharing platforms could misuse users’ passkeys or share them with other entities.
    - Fear that someone else’s access to the device could compromise all user passkeys and accounts with services.
    - Fear about recoverability, especially if a device is lost.
    - Fear of not being able to use a service if the user does not have their phone with them or if it is not charged.
    - Fear of biometric data being shared with passkey platforms or even services that use passkeys.
    - Fear of passkey platforms tracking access to third-party services.
    - The idea that Bluetooth is always necessary or that Bluetooth can be used to compromise passkeys.
    - Fear that any change is bad or a conspiracy against the privacy and agency of individuals.

# 7 Passkeys in MFA and LoA Applications

Passkeys embody the core principle of multi-factor security, as they already incorporate two factors. They are stored on the user's device, making them something the user possesses, and they can only be used when the user presents biometrics (something the user is) or enters a PIN (something the user knows). Technically, there are no limitations to using passkeys in MFA scenarios that involve other independently acquired and verified factors.

Relying Parties may have concerns that a passkey could be compromised if an attacker gains access to a single factor, typically a password, used to access the platform provider's account during passkey registration. However, this is usually not the case, as platform providers use multiple elements in addition to the user's password when authenticating users or recovering passkeys on their devices. These elements can include the devices and IP addresses used, location, phone number access, account recovery questions, etc. Some of these elements are visible to the user, while others are not. Therefore, the illusion is created that knowing only the password is sufficient to access the platform account.

Passkeys involve the use of locally stored credentials, but these credentials are concealed behind the passkey implementation of the WebAuthn interface. This concealment allows the use of various user verification methods, fixed or roaming authentication hardware, sign-in on other devices, and enforcement of other security-related options. It also allows passkeys to be easily incorporated into multi-factor authentication. To ensure a certain level of consistency and security, passkey standards and guidelines regulate the options for user verification used to access passkeys and facilitate interaction between devices. The available and allowed options may change in the future to support new and more secure authenticators and emerging security technologies.

There is no restriction on using passkeys as one of the factors in the implementation of multi-factor authentication, similar to the step-up reauthentication scenario with passkeys but with the instant invocation of additional factors. WebAuthn enables the service to create or request credentials with specific characteristics, verify the properties of the passkey used and determine how the user was verified. This can include device-bound passkeys, the use of other specific types of credentials supported by WebAuthn and FIDO2, or even completely independent methods.

On the other hand, information about the factor used to access the passkey is typically not disclosed to the Relying Party, and it should not be a regular practice to do so. The flexibility in using factors may be seen as a disadvantage in more rigid level of assurance (LoA) schemes, as the technical dependency on various mechanisms for user verification on different devices does not align well with the designation of an LoA. Passkey platforms also introduce a third party that is required for convenient sharing and sign-in with passkeys on other devices. The end-to-end encryption and backup of encrypted passkeys for recovery purposes present in some ecosystems add another element of uncertainty. These issues can be resolved over time, potentially through credible and auditable tracking and validation of the hardware and software stacks of passkey vaults involved in user verification, as well as the supporting processes and services provided by passkey platforms.

In certain environments with special compliance requirements, it may be necessary to ensure that only one copy of the cryptographic key exists. For such use cases, device-bound passkeys on FIDO2 security keys and the use of flags indicating whether a credential can be or has been backed up could be a suitable solution.

Regulatory regimes have yet to evolve and officially recognise passkeys as one of the sanctioned forms of multi-factor authentication. The FIDO Alliance is actively involved in this area.

It remains to be seen how the available passkey capabilities can be used in high-security scenarios and effectively integrated with new user verification methods to achieve a specific LoA or other recognised trust levels.

## 7.1 Managing Security

In the WebAuthn authenticator model, the underlying data structure for the authenticatorMakeCredential operation is credTypesAndPubKeyAlgs.

It indicates public key algorithms with pairs of ‘public-key’ PublicKeyCredentialType and COSEAlgorithmIdentifier and by using identifiers described in CBOR Object Signing and Encryption (COSE) [COSE]. The authenticatorGetAssertion operation can retrieve the credential characteristics by using the allowCredentialDescriptorList data structure, which contains a list of acceptable public key credentials as PublicKeyCredentialDescriptor objects. RFC 8812 [RFC8812COSEJOSE] specifies the use of several algorithms for web authentication with COSE and JSON Object Signing and Encryption (JOSE). These algorithms are used by implementations of the WebAuthn and FIDO2 Client to Authenticator Protocol (CTAP) specifications.

The authenticatorGetAssertion operation’s ‘extensions’ parameter is described in WebAuthn Extensions [IANAWA]. At the FIDO2 level, credentials are described in [FIDOReg] via the ALG_KEY constants, which indicate the specific public key algorithm and encoding format, while the ALG_SIGN constants indicate the specific signature algorithm and encoding.

## 7.2 FIDO2 Attestations

Passkey implementations can leverage FIDO2 attestations to secure high-assurance scenarios. Services can request, store and use attestation information to make risk decisions based on the type of credential being used.

Currently, attestations and metadata statements [PkAttestMS] provide the ability to obtain details about the actual security device on which a passkey was created and the manufacturer who produced that device. This allows the Relying Party to decide whether to allow the use of the passkey.

These attestations are requested by the Relying Party during user registration. They provide a unique identifier for every FIDO2 authenticator and an attestation signature that can be verified with the manufacturer’s public key. The identifier indicates the make and model of the authenticator. These attestations and the manufacturer’s metadata statements are available in a public database.

During registration, the Relying Party can send its registration policy along with the challenge message. This enables clients to select or configure authenticators that match the requested characteristics for registration. Along with the signed challenge, the client can optionally send a signature created with the private key corresponding to the public key of the authenticator’s attestation certificate. These certificates are described in publicly available metadata statements.

The authenticator’s adherence to the registration policy (e.g., the use of a specific factor), its identifier, metadata (e.g., characteristics of the user verification method) and other information (e.g., about its attestation or application-specific regulatory requirements) can be used to assign a risk or trust score to the authenticator.

The extensibility of authenticator attestations and attestation statements allows the addition of further details, such as those describing the management of passkeys. Features provided by FIDO2 and WebAuthn can, for example, help applications determine whether the credentials used can be backed up or have already been backed up.

## 7.3 Capabilities Offered by FIDO2

FIDO2 provides several user verification methods, including presence, fingerprint, local-only passcode, voiceprint (speaker recognition), faceprint (face recognition), location (sensor or measurement), eyeprint (eye biometrics), drawn pattern and handprint (palm-print, hand or vein geometry) [FIDOReg]. Additionally, authenticators must specify:

- Whether the verification method includes elements outside the authenticator.
- The method used to protect the private key material.
- The method protecting the matcher that performs user verification.
- Whether they use display capability to interact with the user.
- The use of specific authentication algorithms, public key formats and other cryptography-related data.

The authenticator must indicate whether it supports transaction confirmation display and whether software operating in a privileged context produces this display, in order to point out the nature of user confirmation and its vulnerability to screen overlay attacks. It should also indicate whether a Trusted Execution Environment, hardware display capabilities or a device other than the authenticator or user device has been used.

Authenticators may indicate the method used for communication between the user device and the internal or external authenticator. External authenticator connections can be made through exclusive wired protocols such as USB or Firewire, wireless personal area networks or non-routed protocols such as Bluetooth, NFC or Wi-Fi Direct, and non-exclusive networks such as TCP/IP LAN or WAN.

For example, the JavaScript API used in sign-in pages allows the specification of user verification requirements, such as whether a biometric check is required.

All of these options enable the implementation of multi-factor authentication by requiring multiple interactions using different credentials and combining their trust score into a composite value, or by requiring the user to sign in with specific passkeys and possibly other credentials. However, these applications have not been widely tested with existing passkey implementations.

# 8 Findings from Testing

During the work on the prototypes and this report, the GN5-1 Trust and Identity (T&I) Incubator team identified some implementation-dependent aspects of passkeys and different interpretations of some definitions in implementations. Furthermore, most passkey implementations lack detailed public documentation that would comprehensively describe the underlying design decisions and supported WebAuthn features and flags.

To assess differences in common implementations of passkeys, the team designed a battery of tests and conducted a series of tests using the most prevalent passkey authenticators. The objective was to capture the behaviour of authenticators that regular users would likely use, to address questions not answered by reading the specifications, and potentially uncover deviations from the specification or its common interpretations that could impact a substantial portion of the user base. Therefore, the team did not aim to test all passkey devices available and all WebAuthn features and flags but to test the common cases and most relevant features that could affect deployments within eduGAIN.

Instead of developing a testing tool, the Incubator team compared several available testing tools, including the WebAuthn.me debugger [WADebugger] and WebAuthn.io [WAIO], and ultimately decided to conduct tests using the site WebAuthnTest.IdentityStandards.io [WATISIO]. This website offers a straightforward JavaScript application enabling the registration of security keys and passkeys. Before registration, users may modify numerous WebAuthn parameters such as user verification and attestation, making this application suitable for the team's test cases. After each registration, the application displays the result of the registration, including the generated public key and the corresponding WebAuthn flags. This enables verification of whether the resulting passkey is aligned with specified requirements.

The team created detailed instructions on conducting a passkey test using the selected website [PkCT]. This allowed support for various WebAuthn features to be assessed, including resident keys (passwordless authentication), conditional mediation (autofill UI), user verification (requiring a PIN, fingerprint or other biometrics for authentication), attestation (providing vendor data about the device, including FIDO certification), and cryptography (algorithms employed in the digital signatures).

Several members of the Incubator team performed tests using various devices and passkey providers. These encompassed both single-device passkeys such as YubiKey and Windows Hello, and multi-device passkeys provided by Google and Apple. Tests were conducted in various web browsers and on different operating systems, resulting in 15 successful test results. The tests spanned five operating systems, four web browsers, their nine combinations, and eight passkey authenticators, as summarised in the four tables below.

| Operating system | Number of tests |
|---|---|
| Windows | 3 |
| macOS | 2 |
| Ubuntu (Linux) | 5 |
| Android | 2 |
| iOS | 3 |

Table 8.1: Tested operating systems

| Web browser | Number of tests |
|---|---|
| Brave | 1 |
| Firefox | 7 |
| Google Chrome | 5 |
| Safari | 2 |

Table 8.2: Tested web browsers

| Operating system | Web browser | Notes |
|---|---|---|
| Android | Google Chrome | Backed-up passkeys are shared in the Google ecosystem.<br>Device-bound passkeys are supported only on security keys. |
| iOS | Safari, Google Chrome | Backed-up passkeys are shared in the Apple ecosystem.<br>Device-bound passkeys are supported only on security keys. |
| macOS | Safari | Backed-up passkeys are shared in the Apple ecosystem.<br>Does not support self-asserted attestations. |
| macOS | Firefox | Only device-bound passkeys on security keys are supported.<br>User verification setup is not supported, but PINs can be set in Chrome or Safari. |
| macOS | Google Chrome | Created passkeys are device-bound.<br>Does not support self-asserted attestations. |
| Ubuntu (Linux) | Google Chrome | Only device-bound passkeys on security keys are supported. |
| Ubuntu (Linux) | Safari | Only device-bound passkeys on security keys are supported. |
| Windows | Firefox | Only device-bound passkeys are supported. |
| Windows | Brave, Google Chrome | Only device-bound passkeys are supported. |

Table 8.3: OS and browser combinations with notes on behaviours

| Authenticator | User verification method | Notes |
|---|---|---|
| Windows Hello | Fingerprint, face recognition, PIN | Uses RSA, single-device only. |
| Apple TouchID/FaceID | Fingerprint or face recognition | Seems to require iCloud Keychain on iOS.<br>Does not work in Firefox. |
| Google/Android | Fingerprint | Chrome offers Google passkey over other options on all platforms. |
| SoloKey | PIN | The PIN has to be set up beforehand in Chrome or Safari, but can be used also in Firefox. |
| OnlyKey by CryptoTrust | No user verification or PIN | The PIN set up on the built-in keyboard is not used for passkey login. |
| YubiKey 5 by Yubico | PIN | PIN has to be set up beforehand in Chrome or Safari, but can be used also in Firefox. |
| ePass Plus K9D by FEITIAN | No user verification or PIN | Does not ask for a PIN even though the site requires it. |
| Dashlane (password manager) browser extension | Master password | The sign-in prompt is displayed as an iframe inside the page and therefore might be covered by other elements. |

Table 8.4: Used passkey authenticators with notes on their user verification and cryptographic capabilities

For a comprehensive view of the test results, please refer to the related set of reports in GÉANT Confluence [PkCTResults].

Key findings from the test results are summarised below:

- Passkey usage without user verification (i.e., without a password, PIN or biometric verification) is possible. Consequently, it cannot be assumed that all passkeys are safeguarded against use on a stolen or unprotected user device.
- User experience and available features differ significantly across operating systems, web browsers and devices. For example, a PIN for a physical security token may be used in Firefox and Chrome but cannot be created in Firefox. Furthermore, TouchID may not function in Firefox on macOS, despite working in Safari and Chrome.
- Most tested passkeys employ ES256 (ECDSA using P-256 and SHA-256) for digital signatures, while Windows Hello is the only one that exclusively uses RS256 (RSASSA-PKCS1-v1_5 using SHA-256; it used a 2048-bit key in the team's test).
- At least some passkeys can provide attestation data. Among the tested authenticators, only physical security keys provided a FIDO-certified verifiable attestation chain. Most other authenticators in the test provided self-asserted attestations (Windows Hello, TouchID and FaceID on iOS and Android). macOS supported only enterprise attestations, which are designed for controlled environments and include uniquely identifying information in attestations.
- It is necessary to always validate the outcome of registration and not rely solely on requested parameters. For example, even when user verification is mandated in the registration request, the authenticator may not always perform it. However, the flags in the registration result can typically be trusted.

# 9 Conclusions and Next Steps

This white paper has described what passkeys are, their technical background, user experience, interaction flows, benefits, limitations, their relationship to multi-factor authentication and their applicability within R&E federations.

It is evident that passkeys greatly enhance user authentication security and have the potential to improve the user experience. They offer a secure authentication method using devices that many users already possess. Passkeys guard against most known remote authentication threats such as credentials guessing, stuffing, spoofing or phishing, and against their use on stolen or unprotected devices. The user experience is greatly streamlined, and passkeys better protect users from compromising their credentials. Therefore, the T&I Incubator team believe that passkeys should be offered as an alternative user authentication method whenever feasible.

Although passkeys solve numerous problems that arise from the use of passwords, some of which cannot be solved even by password managers, there are a few new challenges specific to passkeys. Passwords are simple – users are used to them, they know how to use them, they work in the same way regardless of the device used, and they can be easily created, reset, sent, and shared between devices and users. Passkeys are relatively new, most users are not familiar with the required flow, they can only be created on a user's device, can currently only be shared within single-vendor ecosystems, and the user experience differs between operating systems and web browsers. The adoption of passkeys would benefit from a unified user interface (UI) for their end-user management. Ideally, a standardised set of features, options and behaviours should be established to simplify their management and enhance the user experience (UX) across various platforms and third-party passkey providers. However, such unification should not hinder innovation in the market. Currently, there are only UI implementation guidelines aiming to harmonise passkey registration and sign-in UI. Currently missing are standard APIs for accessing platform authenticators and integrating third-party passkey providers with browsers and operating systems. It would also be beneficial if end users could directly control the synchronisation of passkeys across ecosystems, possibly through standardised and regulated integration of passkey providers.

To introduce passkeys in an organisation or a research community, there is work to be done not only on the technical level but, more importantly, user documentation and support processes need to be significantly adapted to accommodate passkeys.

At the technical level, rolling out a passkey is relatively straightforward if the IdP product the organisation is already using supports the scenario and features it intends to implement. For instance, SimpleSAMLphp together with the SimpleSAMLphp WebAuthn module, maintained by SimpleSAMLphp developers, offer both first-factor authentication and a filter for second-factor authentication; Keycloak also supports passwordless authentication, although not as a default; Shibboleth IdP should provide support for passkeys in the foreseeable future. If an organisation decides to migrate from one IdP product to another that better supports passkeys, the primary work involved is the migration itself, with configuring passkeys being a relatively minor effort.

The specific issues to be resolved differ depending on the use case. In some instances, the transition might be quite easy, while in other cases it is far more challenging. For example, if an organisation mandates the use of a single-vendor ecosystem and existing support processes are compatible with passkeys and features of the used platform, the transition to passkeys is relatively easy. On the other hand, when users have the freedom to choose any passkey provider, platform or ecosystem, and current support procedures involve the distribution of passwords via postal mail or IT technicians, the shift to passkeys necessitates substantial planning, piloting and iterative improvements based on user feedback.

The recommended approach therefore is to add passkey support to the existing password-based authentication and let users try out passkeys by offering them both authentication methods. This should be implemented first in a limited set of applications and to limited groups of users, and should be followed by the gradual improvement of the system, user manuals and support procedures. Later, the users should be allowed to disable password authentication by themselves, and, eventually, forced to use only passwordless authentication.

Currently, only FIDO-certified single-device passkeys and devices with enterprise attestation should be used in level of assurance (LoA) scenarios unless a controlled set of passkey authenticators and platforms can be enforced.

The behaviour of currently available authenticators, as well as the advanced and optional features of existing passkey platforms and providers, is unlikely to undergo significant changes until they achieve widespread adoption among mainstream users. Consequently, the insights presented in this white paper are expected to remain relevant for the next few years. It is also anticipated that, at that point, vendors will exhibit more interest in accommodating the needs of niche communities, such as those focused on LoA scenarios and individuals seeking finer control over passkey sharing and backups. However, the T&I Incubator team expects that support for passkeys in IdP products, particularly Shibboleth and privacyIDEA, will undergo substantial improvements in the first half of 2024.

# Appendix A Passkeys Web Integration Examples

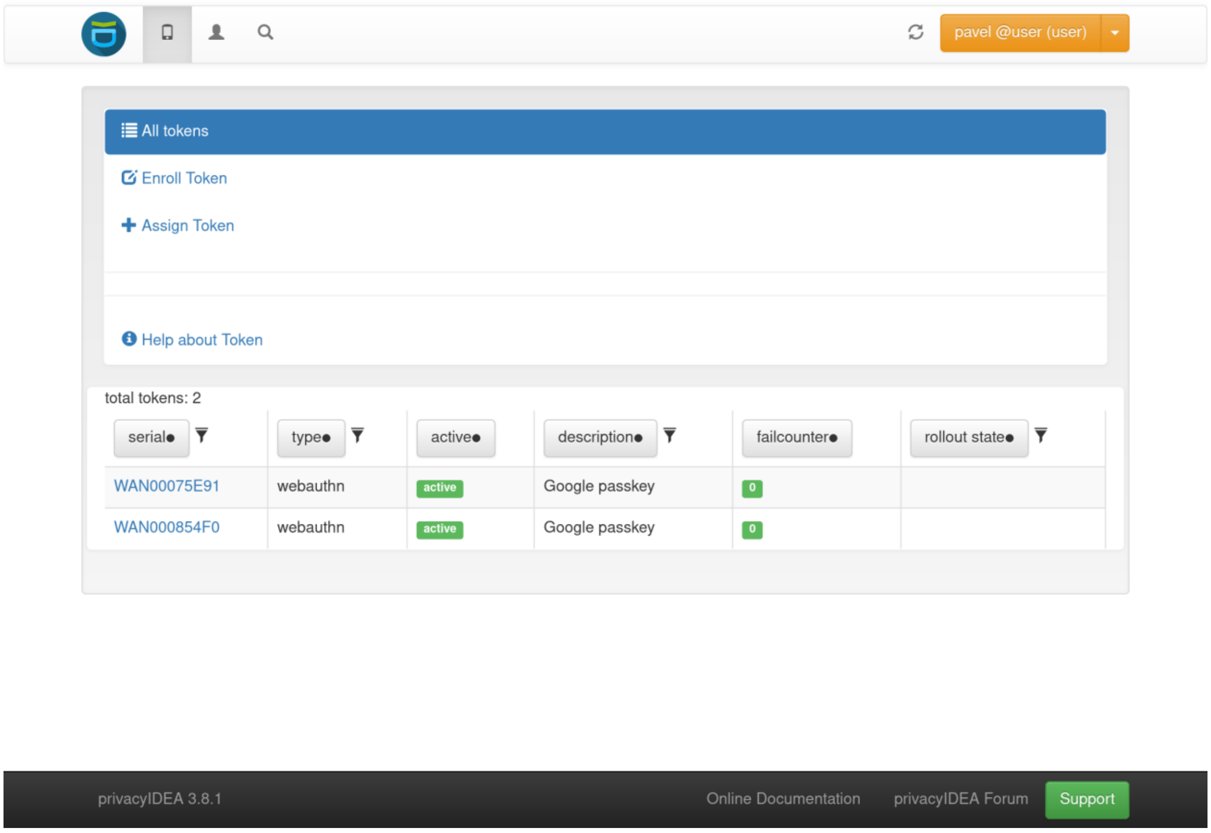
One of the objectives of the passkey activity in the T&I Incubator was to leverage available passkey-capable technologies to create a functional prototype of an IdP with a passkey-based login. The team's primary focus was on open-source technologies that could be integrated with the most widely used IdP software in the eduGAIN federation, namely Shibboleth IdP and SimpleSAMLphp.

Regarding Shibboleth, the team found solutions for second-factor authentication using WebAuthn security keys, but not a fully passwordless experience. As comprehensive passkey support is likely still on the Shibboleth IdP roadmap, the team decided to concentrate their efforts on SimpleSAMLphp-compatible software.

## A.1 privacyIDEA Prototype

Two potential candidates for the SimpleSAMLphp prototype were identified. The first was the privacyIDEA token management system, which not only allows the registration of WebAuthn security keys, including passkeys, but also supports TOTP tokens, SSH keys and more. A corresponding module for SimpleSAMLphp integrates privacyIDEA into the IdP and provides limited support for first-factor and for second-factor authentication. The team's initial attempt to build a prototype using this module was due to the versatility provided by privacyIDEA.

privacyIDEA features a web application for users and administrators, where users can manage a variety of tokens – add new tokens, revoke lost ones, remove tokens and more. Administrators may set up complex policies that determine the options available to different user groups. The system encompasses an API, furnishing the necessary methods for user verification. Integrations with privacyIDEA, including the SimpleSAMLphp module, make use of this API during user authentication.



total tokens: 2

| serial | type | active | description | failcounter | rollout state |
|---|---|---|---|---|---|
| WAN00075E91 | webauthn | active | Google passkey | 0 | |
| WAN000854F0 | webauthn | active | Google passkey | 0 | |

Figure A.1: Token management in privacyIDEA

Unfortunately, while working on the prototype, the Incubator team discovered that it is currently not possible to deliver passwordless authentication using this combination of software. The first limitation stems from the privacyIDEA API, which does not support usernameless login. Consequently, it is possible to sign in with a username and a TOTP code, but passkeys can only be used after prior authentication through another method. The second limitation arises from the authentication theme in the privacyIDEA module for SimpleSAMLphp – it only supports authentication methods based on two text input fields: one for the username and another for a password, TOTP code or similar. Consequently, this theme does not support passkey authentication, which does not start by providing a username.

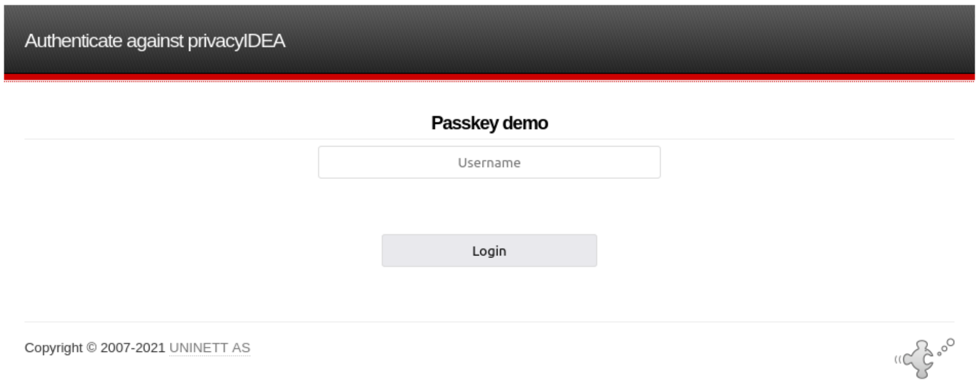

Authenticate against privacyIDEA
Passkey demo
Username
Login
Copyright © 2007-2021 UNINETT AS

Figure A.2: Username-bound authentication page in privacyIDEA

Although the theme could be modified to support WebAuthn credentials, the API limitation prevents a truly usernameless flow. A flow where the user would have to enter a username before using a passkey would bring the security benefits of passkey authentication, but most of the UX benefits would be lost. The autofill UI could not be used because the username would have to be manually entered. Furthermore, the privacyIDEA module only supports version 1 of SimpleSAMLphp, which is no longer actively maintained.

The presented prototype was produced by configuring the stock code. The components used include:

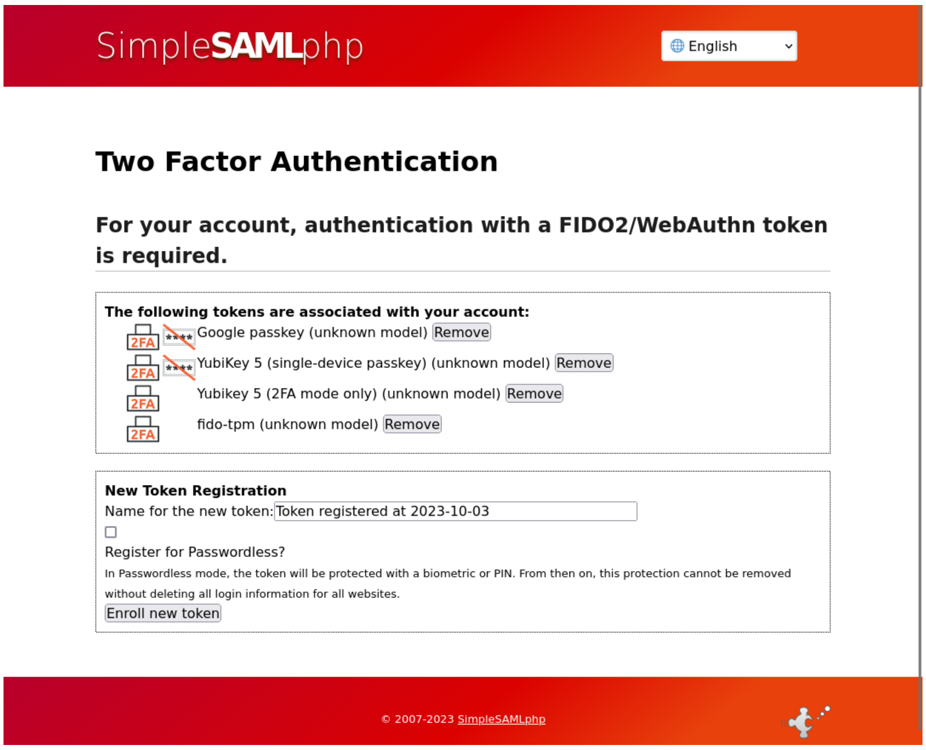
- SimpleSAMLphp 1.19.8 or later [SSphp].
- privacyIDEA 3.8.1 or later [pIDEA].
- privacyIDEA module for SimpleSAMLphp 3.0.0 or later [pIDEA_SSphp].

The prototype has a minimal configuration file, with most settings stored in a database as ‘policies’. A dump of the prototype’s database table with policies is available on the GÉANT wiki at [pIDEA_PC], but it primarily reflects the default setup.

## A.2 SimpleSAMLphp Prototype

The second prototype was constructed based on the WebAuthn module for SimpleSAMLphp, maintained by SimpleSAMLphp developers. This module provides an authentication source for first-factor authentication and an authentication processing filter for second-factor authentication, making it suitable for both passwordless login and second-factor authentication. It supports WebAuthn security keys and passkeys, which should not pose a limitation for most deployments where SimpleSAMLphp is used.

The module also features a registration page where users can manage their security keys and passkeys. The options are relatively limited, allowing users to add new keys (along with associated labels) and remove unwanted ones. Passkeys are distinguishable by an icon with a strike-through password field, differentiating them from security keys that can only be used for second-factor authentication.



SimpleSAMLphp

English

**Two Factor Authentication**

**For your account, authentication with a FIDO2/WebAuthn token is required.**

**The following tokens are associated with your account:**

- Google passkey (unknown model) Remove
- YubiKey 5 (single-device passkey) (unknown model) Remove
- Yubikey 5 (2FA mode only) (unknown model) Remove
- fido-tpm (unknown model) Remove

**New Token Registration**

Name for the new token: Token registered at 2023-10-03

☐

Register for Passwordless?

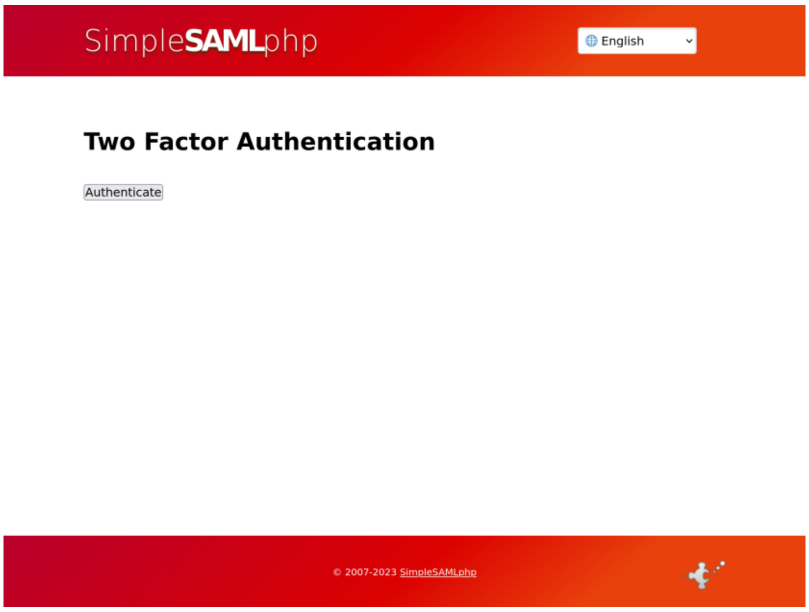
In Passwordless mode, the token will be protected with a biometric or PIN. From then on, this protection cannot be removed without deleting all login information for all websites.

Enroll new token

© 2007-2023 SimpleSAMLphp

Figure A.3: Token management in SimpleSAMLphp

The WebAuthn module is actively maintained and developed and is compatible with the current version 2 of SimpleSAMLphp. The team successfully tested both a 'stable' version 2.0.4 and the development version from the GitHub master branch, which contained several minor bugs. The team addressed those bugs and submitted the fixes upstream as pull requests, all of which were accepted and released as part of version 2.1.0. This resulted in a functional IdP with passkey support.



SimpleSAMLphp

English

**Two Factor Authentication**

Authenticate

© 2007-2023 SimpleSAMLphp

Figure A.4: Two-factor authentication page in SimpleSAMLphp

While passwordless authentication is feasible using this module, the user experience is not ideal. For example, passkey authentication is initiated automatically when the sign-in page loads, which, in some cases (e.g., Android), means that the user does not see the sign-in page and is shown a system prompt directly, without knowing the target of the authentication. The resulting user interface also depends on the theme used by the IdP, which typically differs from the SimpleSAMLphp default theme. The module's configuration determines whether users are allowed to manage their tokens on the standalone registration page, during each authentication, or both.

An organisation implementing passkeys with this module may have additional considerations. At the time of testing, there was no documented option to offer both password and passwordless login, which would be helpful during migration to passkeys. However, there is an authentication source in development that seems to be intended for this purpose, and further improvements might be forthcoming.

The presented prototype was produced by configuring the stock code which includes the team's bug fixes. The components used include:

- SimpleSAMLphp 2.0.4 or later [SSphp].
- WebAuthn module for SimpleSAMLphp 2.1.0 or later [SSphpWAMod].

The prototype has configuration files with choices made during installation, which are well-documented in SimpleSAMLphp. The provided installation instructions were followed closely. Users should also follow the installation instructions for the components, while the provided configuration [SSphp_PC] may be used for reference or comparison in case of trouble.

# Appendix B REFEDS Profiles and Passkeys

The REFEDS Assurance Framework (RAF) [REFEDS_AF] provides organisations with comprehensive guidelines for establishing and maintaining trust in their online identities and services. The RAF covers both technical and non-technical requirements, including identifier uniqueness, identity assurance and attribute assurance. Since passkeys are a credential technology, they are mostly unrelated to the RAF. However, during passkey registration, organisations must ensure that the RAF requirements are met at the beginning of the registration process when the user's identity is established through means other than the passkey. The RAF also determines when a credential needs to be renewed, replaced or revoked. For passkeys, this is achieved or triggered by the deletion of the Relying Party's record for the specific passkey. Therefore, organisations still need to understand and comply with the RAF requirements and apply them when managing passkeys, just like any other credentials they handle.

In addition to the RAF, REFEDS has developed profiles, such as the Single-Factor Authentication (SFA) profile [SFA] and the Multi-Factor Authentication (MFA) profile [MFA]. The SFA profile defines the requirements for authentication with a single factor, such as passwords, cryptographic tokens or biometric factors. However, it does not explicitly cover passkey authentication. A passkey is a form of authentication that combines local hardware, secret data, biometrics and cryptographic signing. Because these elements are practically inseparable, it is justified to treat passkeys as a single factor for SFA purposes. This appendix provides the necessary information for organisations to implement passkeys in a way that meets the REFEDS Single Factor Authentication profile or to develop a potential SFA extension that explicitly includes passkeys.

Passkey authentication can also count as one of the two required factors under the REFEDS MFA profile, which requires at least two different factors. However, an open question remains about whether passkeys stored on two authenticators could be treated as independent factors, especially if they are directly connected to a single client device.

Although passkey authentication is not currently explicitly included in REFEDS profiles, it relies on the WebAuthn specification, which defines various algorithms for authentication and credential management and protection. Developers can utilise the WebAuthn API and registered codes of signing algorithms and follow the guidance provided in the WebAuthn documentation to implement passkey-based solutions that meet the requirements set by REFEDS, as follows:

**1. The authentication factor fulfils the following requirements:**

**1.1. Authenticator secrets have at least the following minimum length:**

Most authenticators use a 256-bit key by applying ES256 (ECDSA using P-256 and SHA-256). Windows Hello uses RS256 (RSASSA-PKCS1-v1_5 using SHA-256) with a 2048-bit RSA key [WinHello].

During passkey registration, the allowed signing algorithm list should be specified only with algorithms that comply with SFA, which means that they use EC with 256+ bits or RSA with 2048+ bits. Specifically, as described in the W3C WebAuthn API [WebAuthn] and Credential Management Level 1 [CM1DraftGH], the Relying Party requests credential creation using navigator.credentials.create(), passing credential creation options PublicKeyCredentialCreationOptions with pubKeyCredParams containing the pubKeyCredParams list of PublicKeyCredentialParameters. During passkey use, the Relying Party requests an appropriate credential by using navigator.credentials.get() and passing the assertion generation options in

PublicKeyCredentialRequestOptions. This allows passkeys to be restricted by providing the allowCredential list within the PublicKeyCredentialDescriptor object.

**1.2. Secrets that are transmitted must have a maximum lifespan according to the method of delivery.**

What is exchanged during sign-in with a passkey is the challenge that needs to be signed by the locally stored private key within the Relying Party-defined timeout.

The registration or authentication process is initiated when the client triggers navigator.credentials.create() or navigator.credentials.get(). The Relying Party sends a challenge and sets a timeout so that it can easily specify any interaction lifespan that may be required. Passkey creation typically times out after 5 or 6 minutes, as the user needs to locally create and save the passkey. The timeout for sign-ins using autofill is typically set to similar values since this UI is not modal and requires a user gesture for the selection UI to display. The modal 'Sign in with a passkey' UI requires the user to just confirm or select a passkey and is typically set to values such as 2 minutes. To comply with SFA, registration and authentication timeouts should be set to at most 5 minutes.

**1.3. Accounts are protected against online guessing attacks (e.g., rate limiting).**

Passkey authentication relies on asymmetric cryptography and the challenge-response principle, so it is virtually impossible to perform regular brute-force attacks. It is assumed that it is not possible to perform an effective online brute-force attack on a private key.

**1.4. Authentication secrets at rest and in online transit must be cryptographically protected.**

Passkeys comply with SFA as only a public key is stored on the server. On the client side, which is not directly relevant to SFA, the passkey platform typically encrypts and shares the private key. As SFA does not address the use of password managers, the same could apply to passkeys. Furthermore, passkeys can be created as single-device WebAuthn-based credentials to prohibit their sharing and platform backup.

Since passkeys use private and public keys, the passkey secret, unlike passwords, is not shared with or kept by the Relying Party or the service. The private key is likely protected within the authenticator and never transferred to the Relying Party. The KEY_PROTECTION flags defined by FIDO can be used to request or determine the key protection method used by the authenticator. Although the availability of the desired protection method cannot be guaranteed, the secret protection assurance with passkeys is far better than with passwords, where the users may keep them any way they like without any method allowing the Relying Party to enforce behaviour or query this.

Passkey private keys are shared between user devices in the same ecosystem using end-to-end encryption. Their backups in iCloud Keychain are cryptographically protected. In Google Password Manager: 'Additionally, passkey private keys are encrypted at rest on the user's devices, with a hardware-protected encryption key.' [AndrRecovery]

**2. Replacement of a lost authentication factor ensures all of the following, as applicable:**

**2.1. An existing secret must not be sent to the user (e.g., a stored password).**

Since the Relying Party does not possess the private key, it is impossible to violate this requirement.

**2.2. The replacement procedure does not solely rely on knowledge-based authentication (e.g., answering a secret question).**

Passkey re-registration fulfils this requirement, as the user must be properly authenticated by other means before creating a new passkey. However, it implies that if only password-based authentication is used during re-registration, it must also comply with the REFEDS SFA requirements.

**2.3. Human-based procedures (e.g., service desk) ensure a comparable level of assurance of the requesting user's identity as the initial identity vetting.**

This is to be implemented in the same way as for other factors.

2.4. **To restore a lost authentication factor, an OTP may be sent to the user's address of record. All corresponding requirements apply as though this OTP would be a Look-Up Secret, except that it may be transmitted without being cryptographically protected.**

This is to be implemented in the same way as for other factors.

2.5. **For authenticators that are provided to the user as a backup, all requirements of the corresponding authentication factor apply.**

The implementation is dependent on what is used as the backup factor. If passkeys are used as a backup, the same requirements that are applied to passkeys as a primary factor can be satisfied. Additionally, sharing passkeys across devices can be considered a secure backup that is transparent to Relying Parties, as passkeys are encrypted in transit and securely stored in all used authenticators.

# References

[1pPk] https://www.future.1password.com/passkeys/
https://www.theverge.com/2023/5/16/23725223/1password-passkey-date-password-manager

[AndrJpCred] https://developer.android.com/jetpack/androidx/releases/credentials

[AndrRecovery] https://security.googleblog.com/2022/10/SecurityofPasskeysintheGooglePasswordManager.html

[AppleRecovery] https://support.apple.com/en-us/HT213305
https://support.apple.com/guide/security/escrow-security-for-icloud-keychain-sec3e341e75d/web
https://developer.apple.com/passkeys/

[BwPk] https://bitwarden.com/passwordless-passkeys/

[CM1DraftGH] https://w3c.github.io/webappsec-credential-management/

[COSE] https://www.rfc-editor.org/info/rfc8152
https://www.iana.org/assignments/cose/cose.xhtml

[CTAP2] https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html

[DLPyWA] https://github.com/duo-labs/py_webauthn

[DUShWA] https://github.com/sipatel2/shibboleth-webauthn

[FIDO2] https://fidoalliance.org/fido2/
https://fidoalliance.org/specs/fido-v2.2-rd-20230321/fido-client-to-authenticator-protocol-v2.2-rd-20230321.html

[FIDO2Comps] https://twitter.com/john_craddock/status/1224620080924971008

[FIDOReg] https://fidoalliance.org/specs/common-specs/fido-registry-v2.2-ps-20220523.html

[GooDevPkLogin] https://developers.google.com/identity/passkeys

[GooglImplAfTut] https://developers.google.com/codelabs/passkey-form-autofill#0

[Hanko] https://github.com/teamhanko/hanko

[IANAWA] https://www.iana.org/assignments/webauthn/webauthn.xhtml

[KcCredOrd] https://github.com/keycloak/keycloak/issues/12102

[KcWATut] https://keycloak.ch/keycloak-tutorials/tutorial-webauthn/

[KpPk] https://docs.keeper.io/user-guides/passkeys

[LPPk] https://www.lastpass.com/security/passwordless-authentication

[MeetPk] https://developer.apple.com/videos/play/wwdc2022/10092/

[MFA] https://refeds.org/profile/mfa
https://zenodo.org/record/5113296

[MSFIDOKeys] https://learn.microsoft.com/en-us/azure/active-directory/authentication/concept-authentication-passwordless#fido2-security-key-providers

[MSProfile] https://myprofile.microsoft.com/

[pIDEA] https://privacyidea.readthedocs.io/

[pIDEA_PC] https://wiki.geant.org/display/GWP5/privacyIDEA+prototype+configuration

[pIDEA_SSphp] https://github.com/privacyidea/simplesamlphp-module-privacyidea

[pIDEAWAUser] https://github.com/privacyidea/privacyidea/issues/3336
https://community.privacyidea.org/t/supporting-discoverable-credentials/2855

| | |
|---|---|
| [PkAfFlow] | https://developers.yubico.com/WebAuthn/Concepts/Passkey_Autofill/Implementation_Guidance/Simple_Autofill_Flow.html |
| [PkAttestMS] | https://developers.yubico.com/Passkeys/Passkey_relying_party_implementation_guidance/Attestation/ https://medium.com/webauthnworks/webauthn-fido2-demystifying-attestation-and-mds-efc3b3cb3651#4e03 https://fidoalliance.org/specs/mds/fido-metadata-statement-v3.0-ps-20210518.html |
| [PkAutofill] | https://developers.yubico.com/WebAuthn/Concepts/Passkey_Autofill/ |
| [PkCASuprt] | https://developers.google.com/identity/passkeys/supported-environments |
| [PkCT] | https://wiki.geant.org/pages/viewpage.action?pageId=633278560 |
| [PkCTResults] | https://wiki.geant.org/display/GWP5/Passkey+creation+tests |
| [PkDash] | https://www.dashlane.com/blog/ushering-in-the-passwordless-future-at-dashlane |
| [PkDef] | https://passkeys.dev/docs/reference/terms/#passkey |
| [PkDevSuprt] | https://passkeys.dev/device-support/ https://passkeys.dev/docs/reference/ https://www.passkeys.io/passkey-ecosystem |
| [PkDir] | https://passkeys.directory/ |
| [PkInAction] | https://www.youtube.com/watch?v=SWocv4BhCNg |
| [PkIphone] | https://support.apple.com/guide/iphone/passkeys-passwords-devices-iph82d6721b2/ios |
| [PkStats] | https://security.googleblog.com/2023/05/making-authentication-faster-than-ever.html |
| [QqWAProxy] | https://github.com/Quiq/webauthn_proxy |
| [REFEDS_AF] | https://refeds.org/assurance https://wiki.refeds.org/display/ASS/REFEDS+Assurance+Framework+ver+1.0 |
| [RFC8812COSEJOSE] | https://www.rfc-editor.org/info/rfc8812 |
| [SFA] | https://refeds.org/profile/sfa https://zenodo.org/record/5113499 |
| [ShIDEA2FA] | https://github.com/wraezor/privacyIDEA-shibboleth-tfa |
| [ShWAPlug] | https://shibboleth.atlassian.net/wiki/spaces/DEV/pages/1210712272/WebAuthn+IdP+Authentication+Plugin |
| [SSphp] | https://simplesamlphp.org/ |
| [SSphp_PC] | https://wiki.geant.org/display/GWP5/SimpleSAMLphp+prototype+configuration |
| [SSphpWAMod] | https://github.com/simplesamlphp/simplesamlphp-module-webauthn |
| [WADebugger] | https://webauthn.me/debugger |
| [WebAuthn] | https://www.w3.org/TR/webauthn-2/ https://w3c.github.io/webauthn/ |
| [WAIO] | https://webauthn.io/ |
| [WATISIO] | https://webauthntest.identitystandards.io/ |
| [WinHello] | https://learn.microsoft.com/en-us/windows/security/identity-protection/hello-for-business/hello-how-it-works-technology |
| [YubDido2WALib] | https://developers.yubico.com/Software_Projects/WebAuthn-FIDO2/WebAuthn-FIDO2_Server_Libraries/ |

# Glossary

| | |
|---|---|
| **AAI** | Authentication and Authorisation Infrastructure |
| **API** | Application Programming Interface |
| **BLE** | Bluetooth Low Energy |
| **CBOR** | Concise Binary Object Representation |
| **CDA** | Cross-Device Authentication |
| **COSE** | CBOR Object Signing and Encryption |
| **CTAP** | Client to Authenticator Protocol |
| **devicePubKey** | Device-bound Public Key WebAuthn extension |
| **EA** | Enterprise Attestation |
| **EC** | Elliptic Curve |
| **ECDSA** | Elliptic Curve Digital Signature Algorithm |
| **FIDO** | Fast Identity Online |
| **HSM** | Hardware Security Module |
| **HTML** | Hypertext Markup Language |
| **HTTPS** | Hypertext Transfer Protocol Secure |
| **IdP** | Identity Provider |
| **IP** | Internet Protocol |
| **JOSE** | JSON Object Signing and Encryption |
| **JS** | JavaScript |
| **JSON** | JavaScript Object Notation |
| **JWT** | JSON Web Token |
| **LAN** | Local Area Network |
| **LoA** | Level of Assurance |
| **MFA** | Multi-factor Authentication |
| **NFC** | Near-Field Communication |
| **OIDC** | OpenID Connect |
| **OS** | Operating System |
| **OTP** | One-Time Password |
| **PIN** | Personal Identification Number |
| **PKCS** | Public-Key Cryptography Standards |
| **R&E** | Research and Education |
| **RAF** | REFEDS Assurance Framework |
| **REFEDS** | Research and Education Federations group |
| **RFC** | Request for Comments |
| **RSA** | Rivest–Shamir–Adleman |
| **RSASSA** | RSA Signature Scheme with Appendix |
| **SAML** | Security Assertion Markup Language |
| **SFA** | Single-Factor Authentication |
| **SMS** | Short Messaging Service |
| **SSH** | Secure Shell |
| **SHA** | Secure Hash Algorithm |
| **T&I** | Trust and Identity |
| **TCP** | Transmission Control Protocol |

| | |
|---|---|
| **TOTP** | Time-based One-Time Password |
| **TPM** | Trusted Platform Module |
| **U2F** | Universal 2nd Factor |
| **UI** | User Interface |
| **USB** | Universal Serial Bus |
| **UX** | User Experience |
| **W3C** | World Wide Web Consortium |
| **WAN** | Wide Area Network |
| **WebAuthn** | Web Authentication (standard) |