Whitepaper

A technical primer on passkeys

How passkeys work and the supported user journeys for consumer use

Contents

Introduction

In May of 2022, Apple, Google, and Microsoft jointly announced their support for a passwordless future in collaboration with the FIDO Alliance.

At the center of this initiative were passkeys, an evolution of FIDO2 authentication that enables faster, easier, and more secure sign-up and sign-in experiences for consumers. Based on the Web Authentication specification (WebAuthn API) and Client to Authenticator Protocol (CTAP), passkeys give businesses a phishing-resistant alternative to passwords that also improve the user experience.

The joint support for passkeys was a critical inflection point in the fight against passwords for two main reasons:

- It sends a strong signal to the broader security industry—and the world—how urgent and pervasive the problem of password-only authentication is
- As the three main consumer technology ecosystems, consensus on a standardized approach to solving the problem provides an avenue for widespread adoption and the elimination of passwords

Passwords are insecure and inconvenient

Every year, new data is released that confirms what we already know: passwords are a weak link in the security chain that also ruins the user experience.

- 43% of consumers will abandon a purchase or account creation flow due to password policies
- 36% of all data breaches involved an element of phishing

As a brief primer, passkeys are FIDO credentials that are discoverable by browsers or housed within native applications or security keys for passwordless authentication. They come in two forms:

- **Synced passkeys**, which are synced between a user's devices via a cloud service like an operating system (OS ecosystem or password manager)
- **Device-bound passkeys**, which never leave a single device; these can be used on FIDO Certified authenticators and security keys

For a deeper dive on what passkeys are and their importance in the fight against password-based attacks, check out our Passkeys Primer.

This document aims to explain how passkeys work from a technical point of view, illustrate the standard authentication flows they support, and briefly touch on implementation considerations for developers.

How do passkeys work?

As explained by the FIDO Alliance, the underlying FIDO protocols employ standard public key cryptography techniques to provide strong authentication. To register with an online service, the user's device creates a new cryptographic key pair consisting of:

- A public key, which is registered with the online service
- A private key, which is retained as a true secret

Importantly, the keys are generated securely and uniquely for every account — you don't have to worry about users picking a weak private key, and private keys aren't reused across multiple services.

To authenticate with a particular service, the client device proves possession of the account's corresponding private key by signing a challenge provided by the service — the service itself never sees the private key and, by extension, never needs to store or protect this information.

Crucially, the private key can only be used after it is unlocked by the user, with the local unlock typically achieved either by inserting a second-factor device or via the primary device's unlock mechanism — usually a biometric authentication, device PIN, or a pattern.

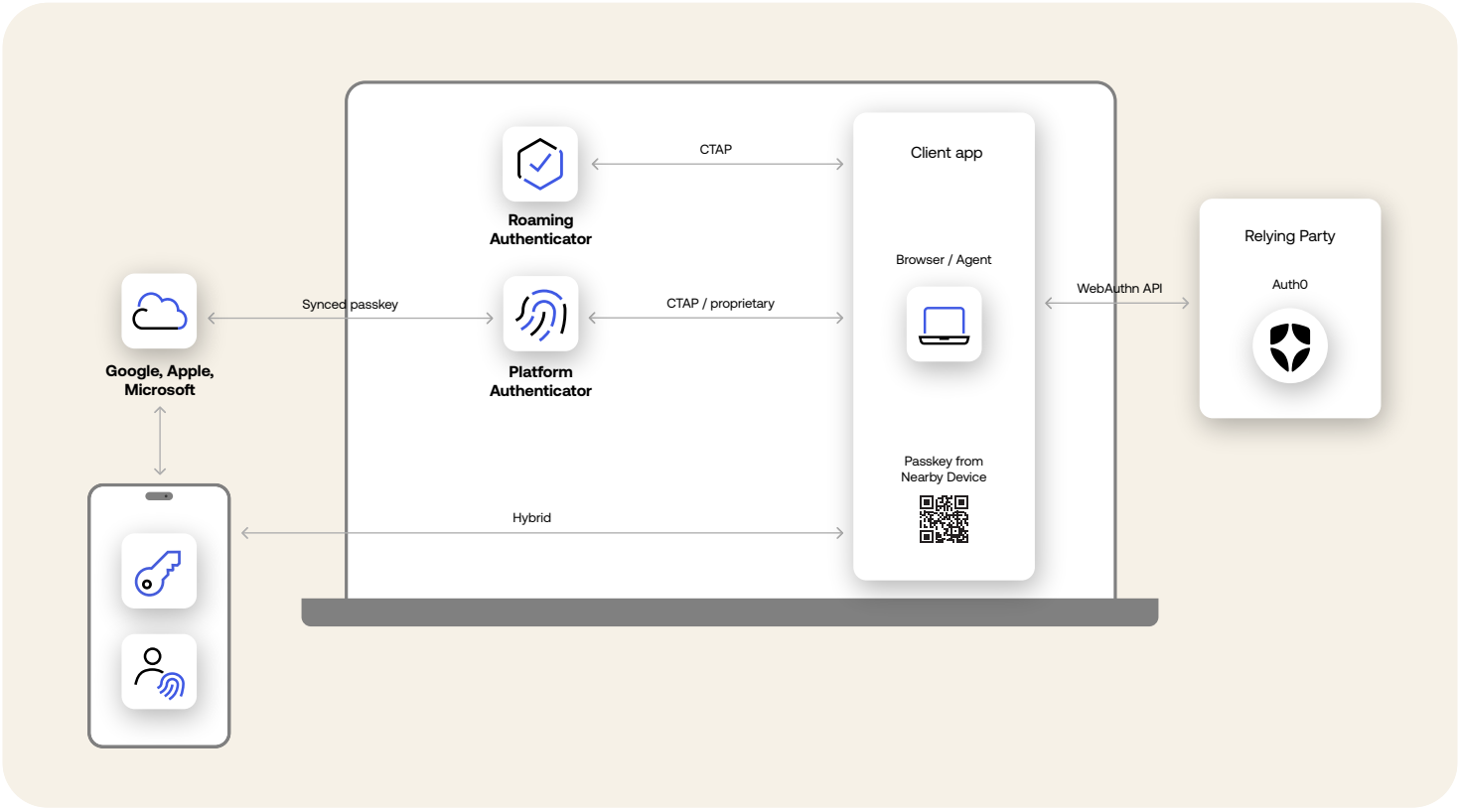
What makes passkeys different from earlier FIDO2 implementations is that they:

- Allow users to use passkeys on any device in a particular ecosystem where passkeys are backed up to the device and cloud
- Allow users to perform cross-device authentication to easily cross ecosystem boundaries without facing the friction of enrolling FIDO credentials on new devices

Supplementary information on passkeys can be found on the FIDO Alliance's website.

Before highlighting the most common flows and user actions relating to passkeys, it's crucial to note the four key entities involved in the authentication ceremony:

- **User:** The user of an application who interacts with the authenticator to authorize operations, e.g. a customer trying to access your application or service
- **Client App/User-Agent:** This is usually a web browser responsible for the client-side communication
- **Relying Party:** The entity that makes use of the passkey to authenticate a user, e.g. an identity service provider such as Auth0 by Okta
- **Authenticator:** Either a hardware or software device that receives requests from the client to either perform an attestation operation or an assertion operation. It generates public key credentials and associates them with a single user and relying party



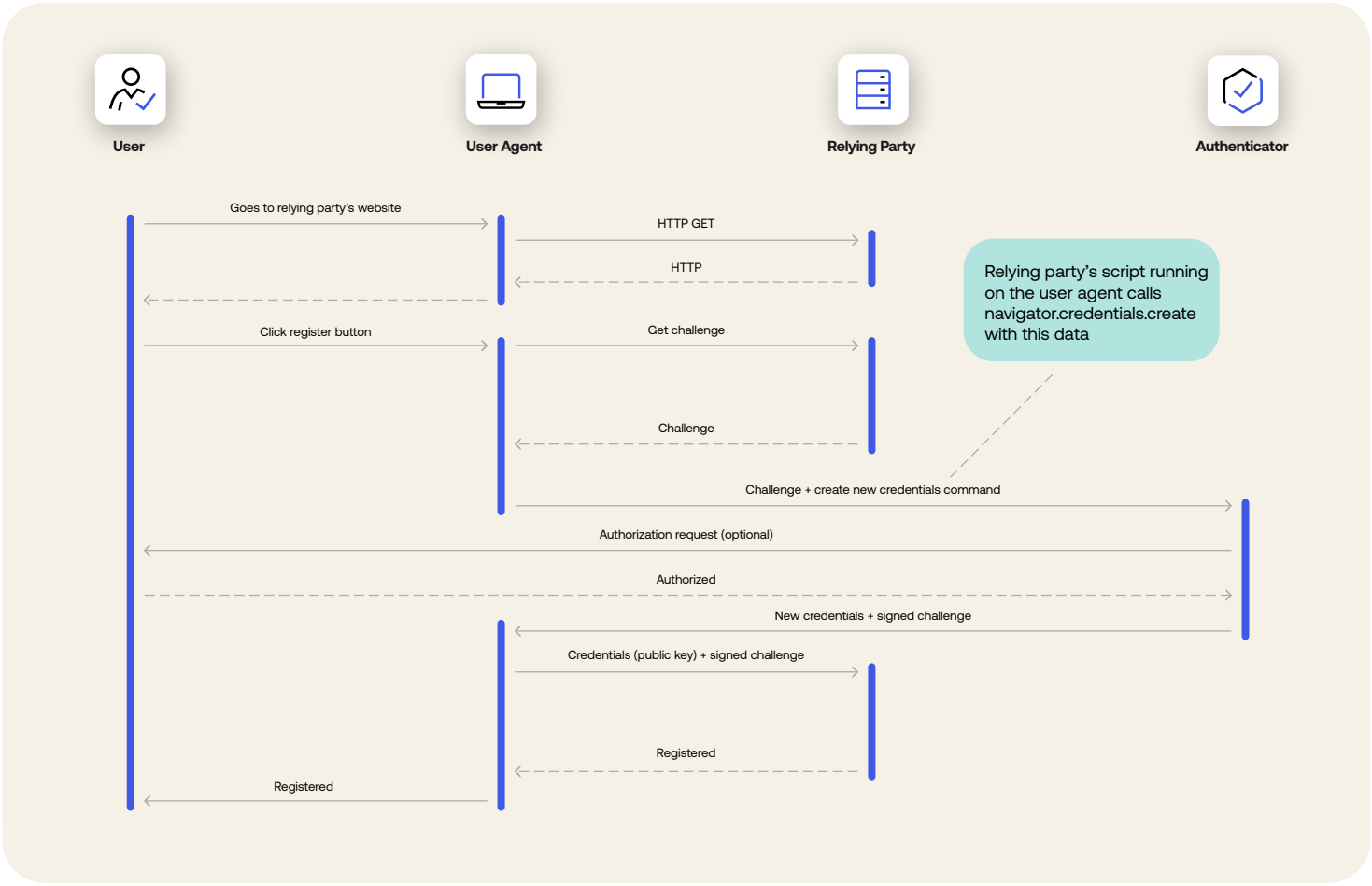
Enrolling a passkey

For new users accessing an application, passkeys can be used as a first factor for passwordless account creation. For returning users who previously signed up using traditional forms of authentication (e.g. a username and password), a passkey can be created for subsequent login.

In both scenarios, users must go through an enrollment process to use a passkey for a given application.

In this flow, the authenticator creates a new set of public-key credentials that can be used to sign a challenge generated by the relying party. The public part of these new credentials and the signed challenge can be sent back to the relying party for storage. The relying party can later use these credentials to verify the identity of a user whenever required.

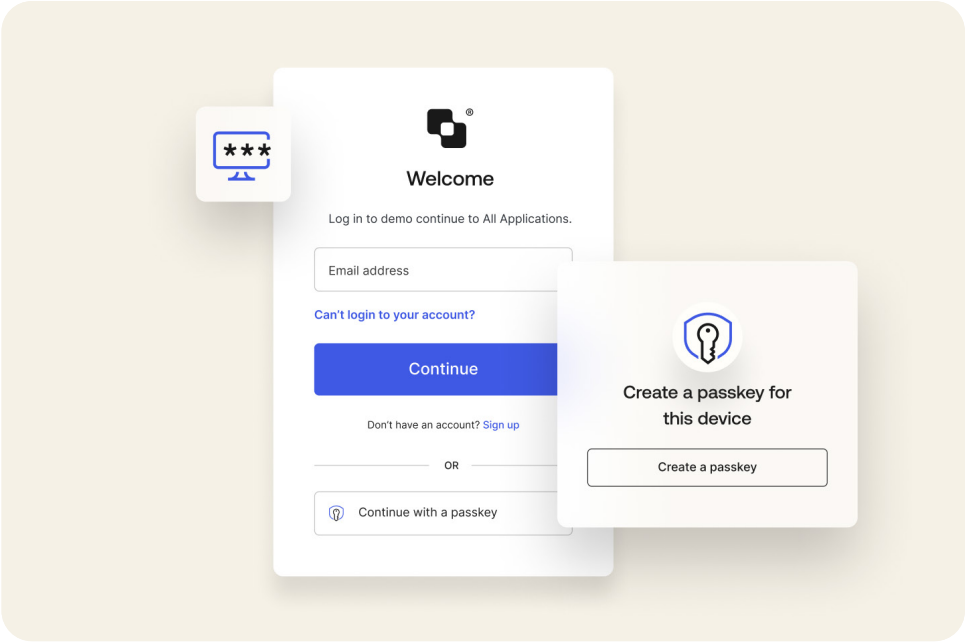
The complete flow, based on the WebAuthn specification, can be seen in the diagram below.



To help illustrate passkey enrollment, we'll take a practical example of a new user trying to access an application on their Personal Computer via a browser where Auth0 is the Relying Party.

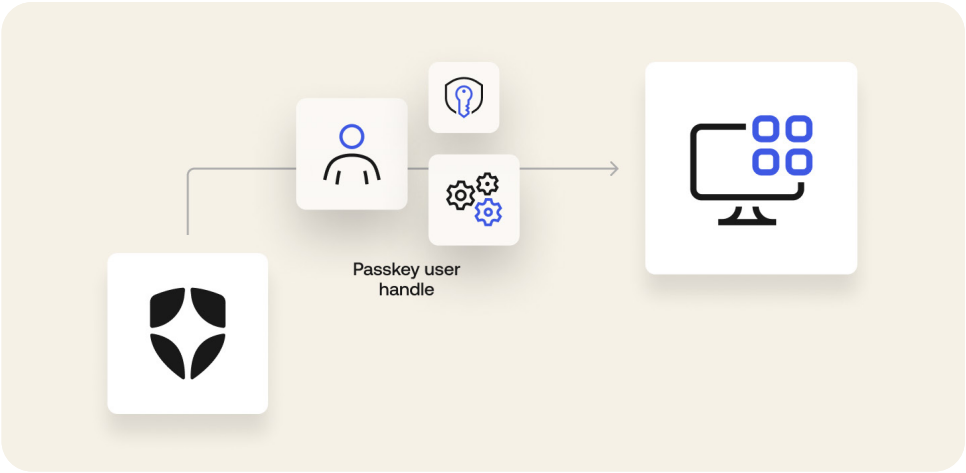
Step 1

The user goes through the registration process on www.atko.com, supported by the relying party. Upon entering their email address, they are prompted to create a passkey.



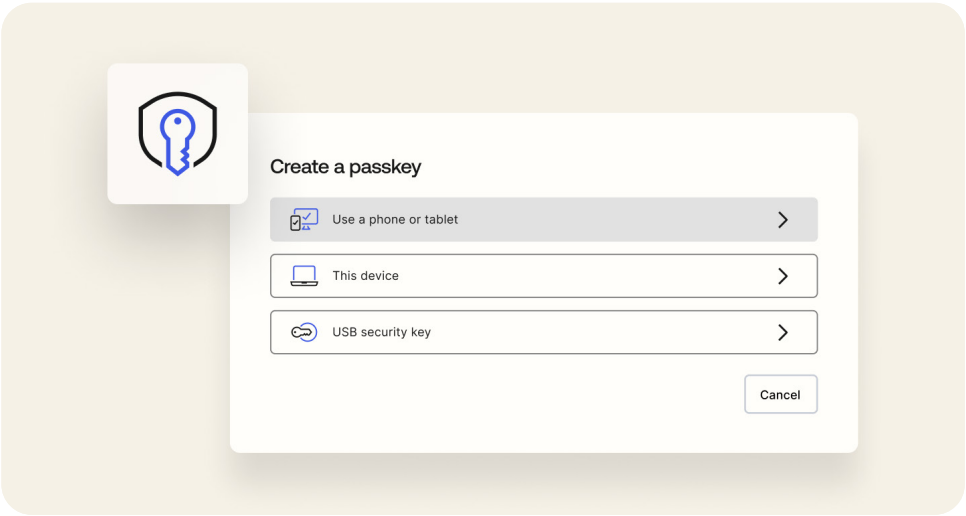
Step 2

The relying party generates a challenge and sends it along with the user handle (non-PII data) and domain name (Relying Party ID) back to the user agent, in this case, the browser.



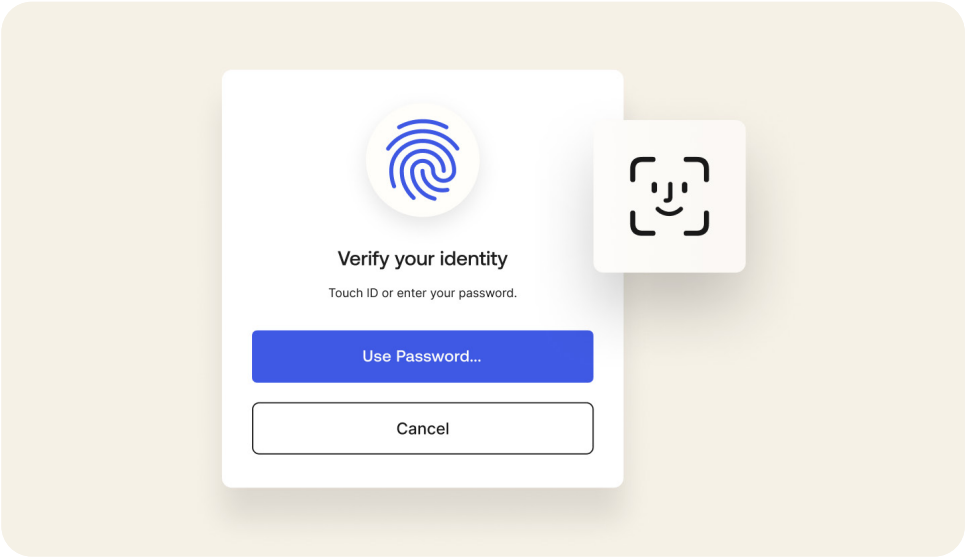
Step 3

The user agent sends the data to the authenticator, which prompts the user to create a passkey for the specific domain name and user handle. It's important to note that the platform ecosystem dictates the user experience at this stage.



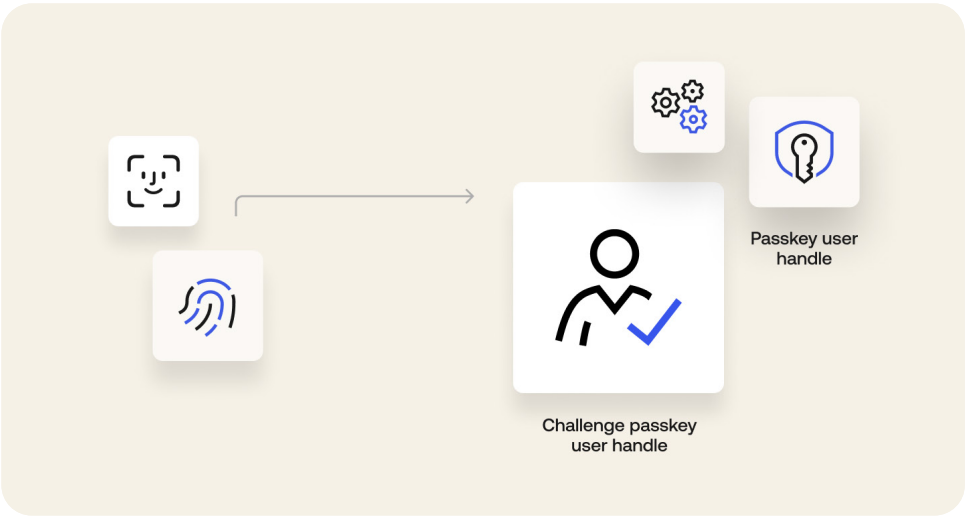
Step 4

The user chooses to create a passkey using their device, at this point, the authenticator prompts the user for verification using the device unlock mechanism, such as biometrics.



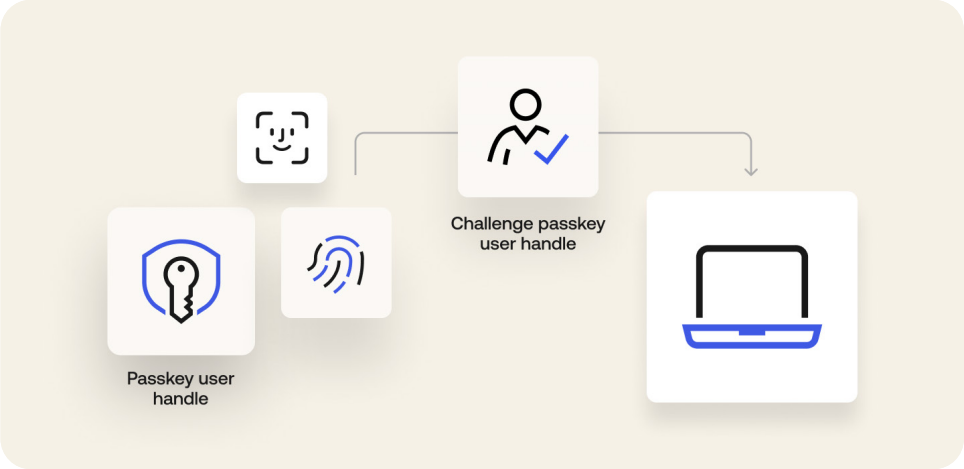
Step 5

The authenticator generates a new public/private key pair credential tied to the domain name and user handle.



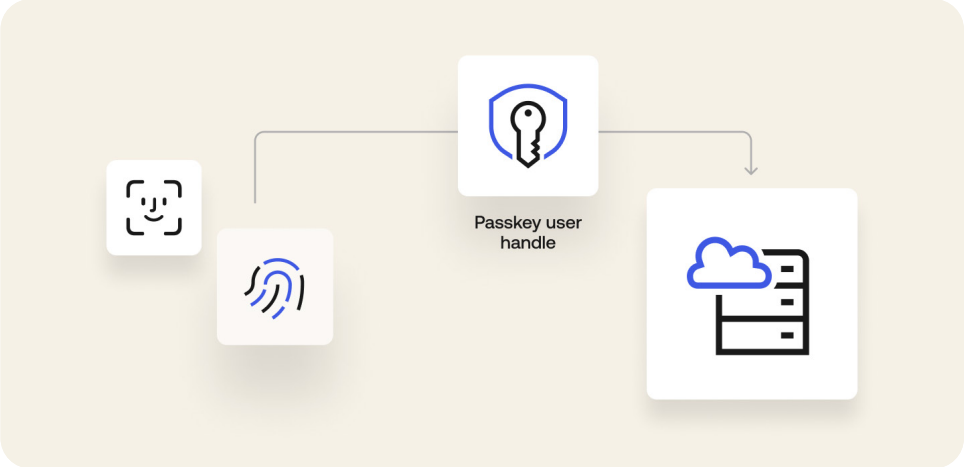
Step 6

The authenticator stores the private key and returns the public key credential and the signed challenge to the user agent.



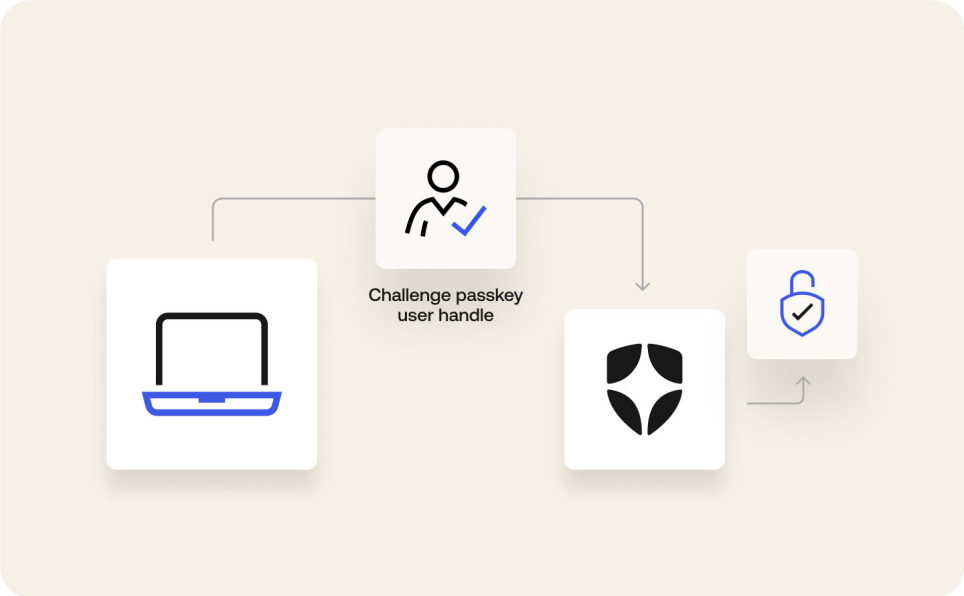
Step 7

On supported ecosystems, the Operating System generates a backup of the private key on their cloud vault (iCloud keychain / Google Password Manager).



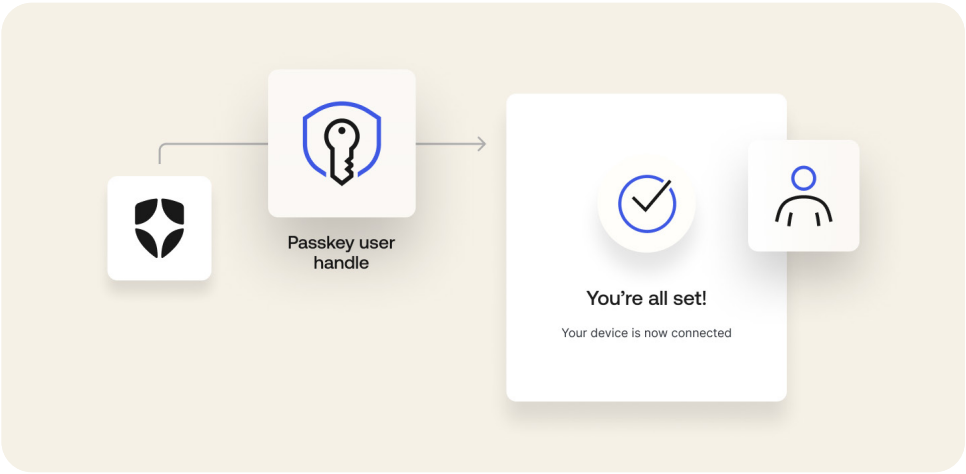
Step 8

The user agent forwards the public key credential and the signed challenge to the relying party to register the user. The relying party validates the credential to ensure the challenge value matches.



Step 9

The relying party stores the credential (public key, user handle), issues a token and redirects the user to the application.



In the example above, step seven is a crucial difference and benefit of passkeys for consumer use. Unlike previous implementations of FIDO authentication, passkeys can be backed up to the cloud, enabling them to sync across multiple devices within the same ecosystem, promoting easier adoption.

However, some criticism of synced passkeys has focused on the shift from browser-based synchronization to OS-based synchronization. This direction was a deliberate decision by the FIDO Alliance, based on the conclusion that operating systems provide a more secure means of making passkeys available to multiple devices than using browsers to do so. Additionally, synced passkeys provide far more phishing resistance than traditional forms of authentication like passwords.

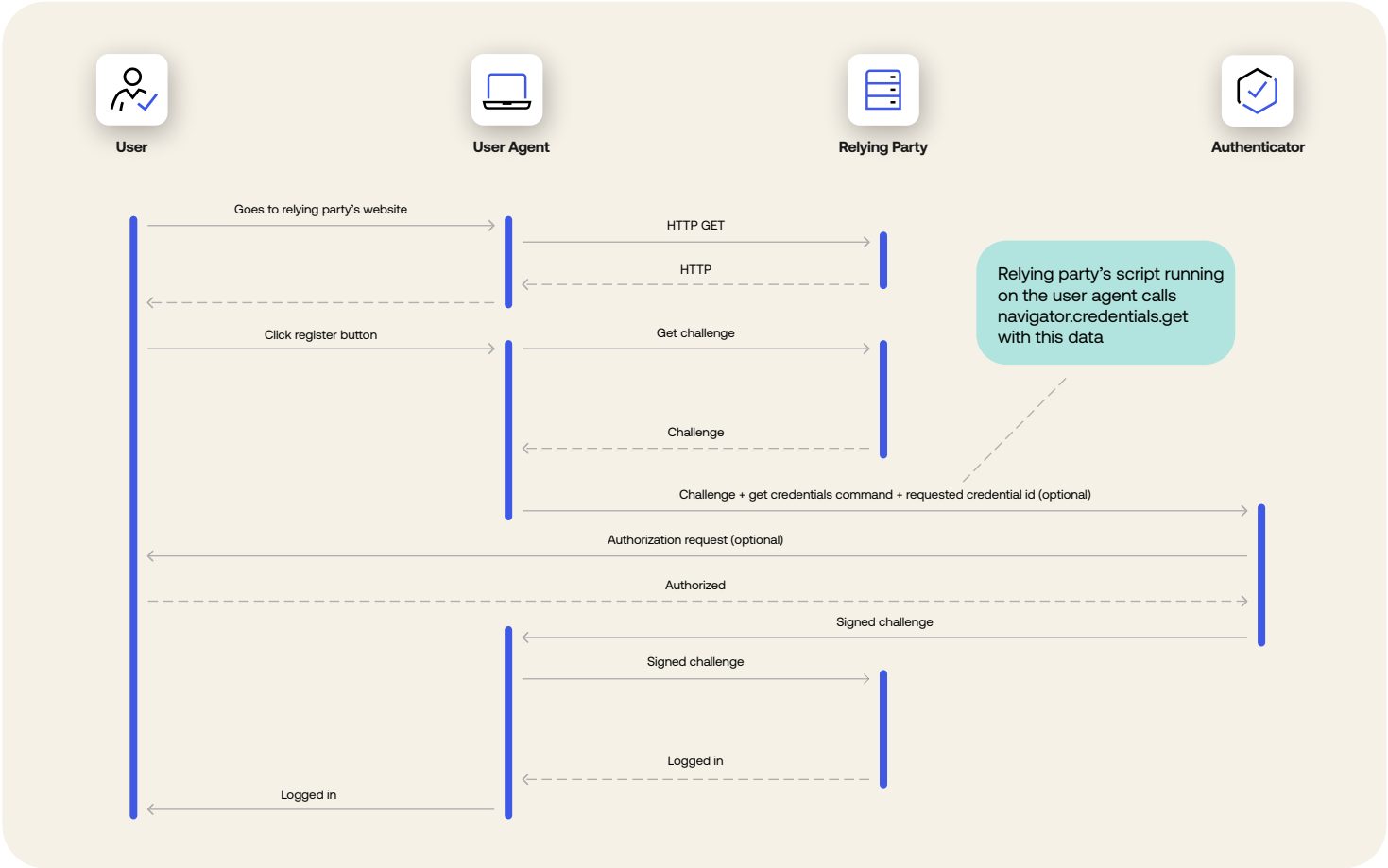
Implications for signup and login

Passkeys are phishing-resistant because they are associated with a particular domain name. When deploying your applications, make sure that the domain name used in the registration and authentication flow remains constant to ensure users retain access to their passkeys.

Signing in with a passkey

Now that the passkey has been enrolled, users can use their passkey to sign in to applications in a seamless and secure manner. Upon login, a digital challenge is issued from the applications server to the user, which can only be solved by proving possession of the corresponding private key from the enrollment phase. A digital signature is generated based on the private key provided by the authenticator and relying party. The private key is kept secret, and only the authenticator needs to know it.

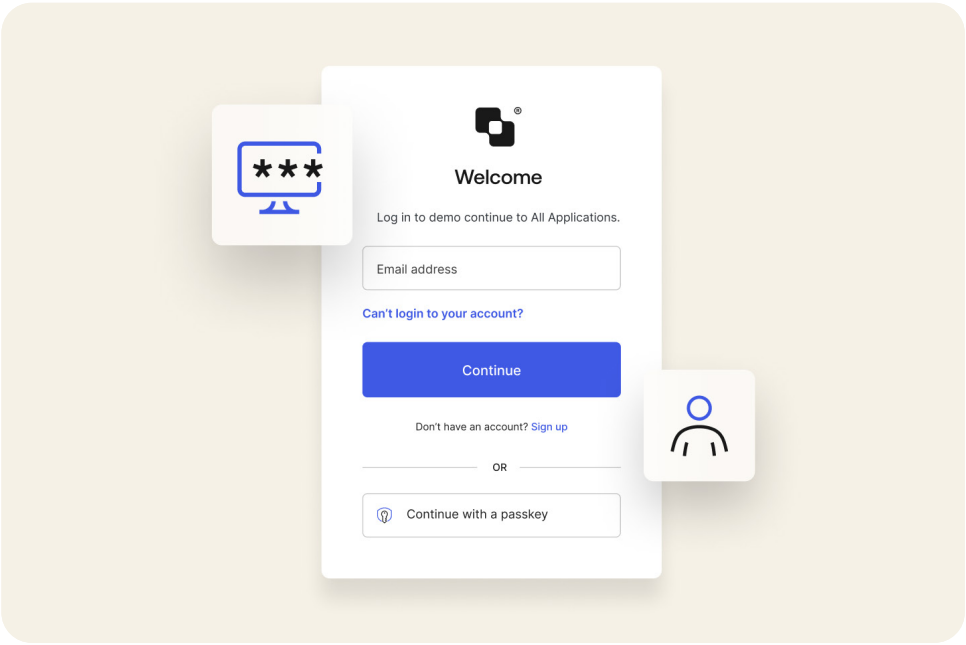
The public key, in contrast, can be seen or stored by anyone. The public key can be used to verify signatures generated by the private key. No other key, besides the private key, can generate a signature that the public key, stored in the relying party, can verify as valid and prove the user identity.



We'll continue our example from the enrollment section to show the mechanics of signing in with a passkey below.

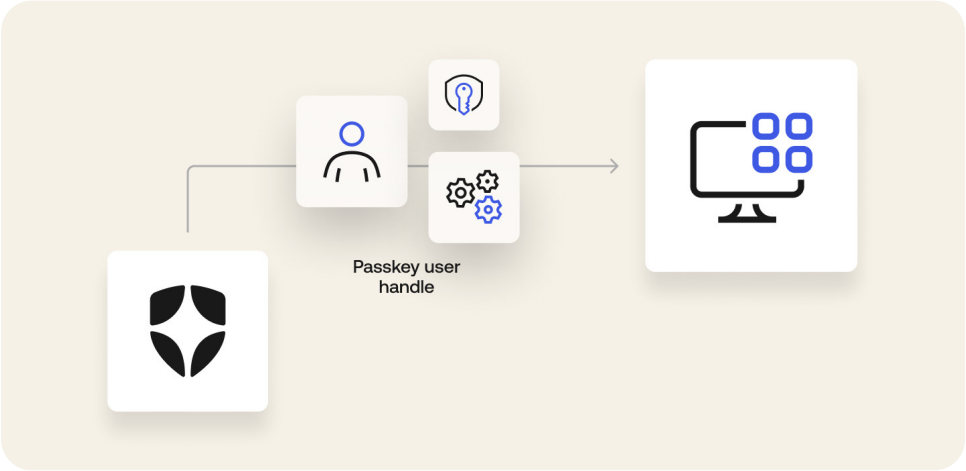
Step 1

The user goes to the relying party's website to start the login process.



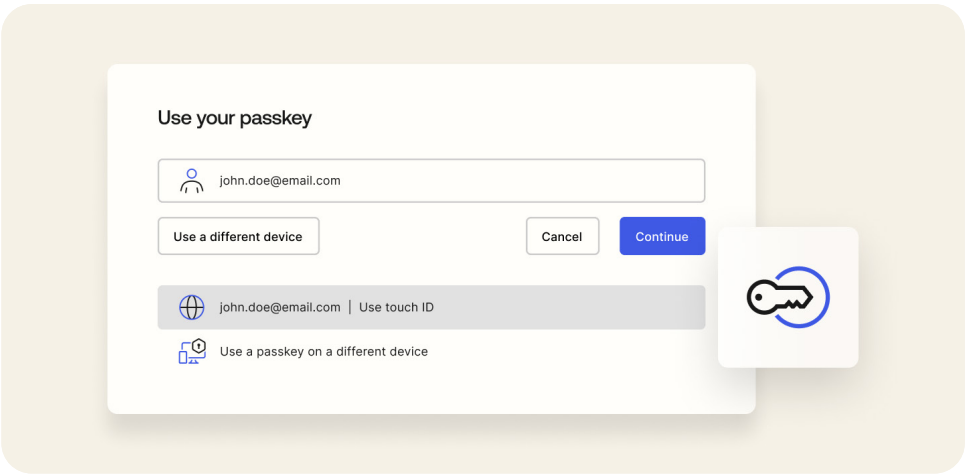
Step 2

Upon clicking the login button, the relying party generates a challenge and sends it back to the user agent. Optionally, additional information can be provided by the relying party, for example, how the authentication should happen or if a specific credential should be used.



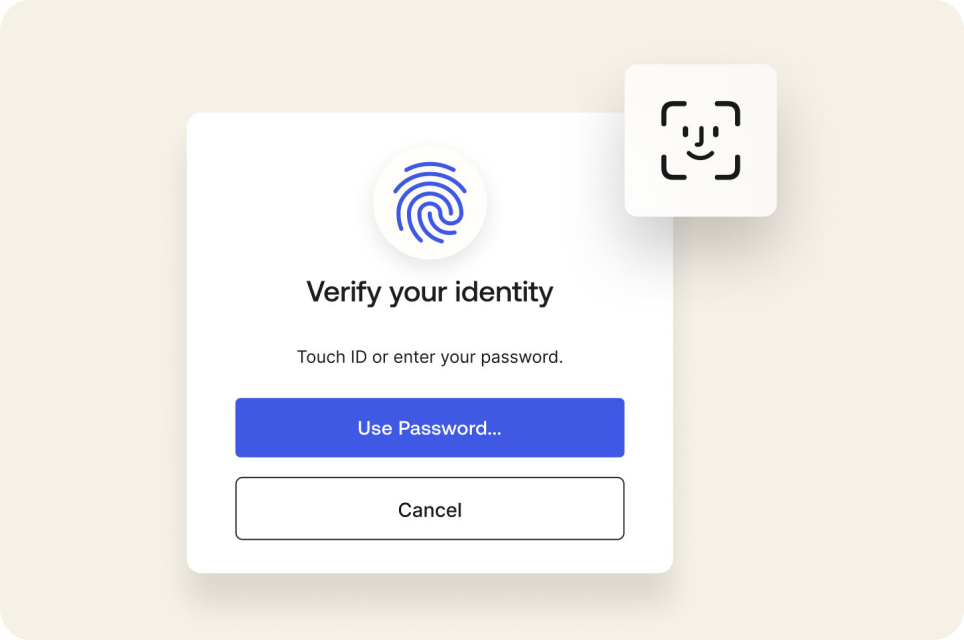
Step 3

The user agent sends the data to the authenticator, which prompts the user to use a passkey to authenticate. A list of passkeys available for the domain name is displayed.



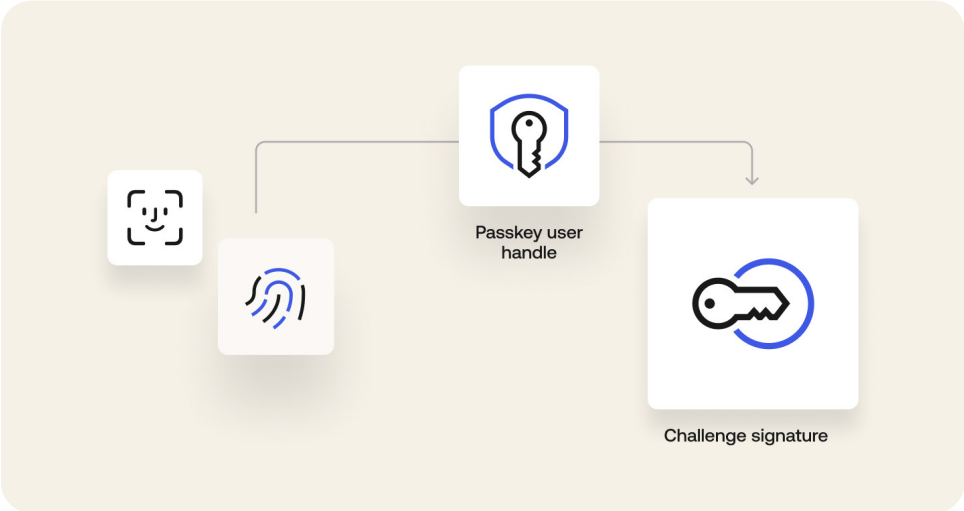
Step 4

The user selects the passkey, and the authenticator prompts the user for verification with the registered biometric, PIN or pattern. The user authorizes the operation by unlocking the authenticator.



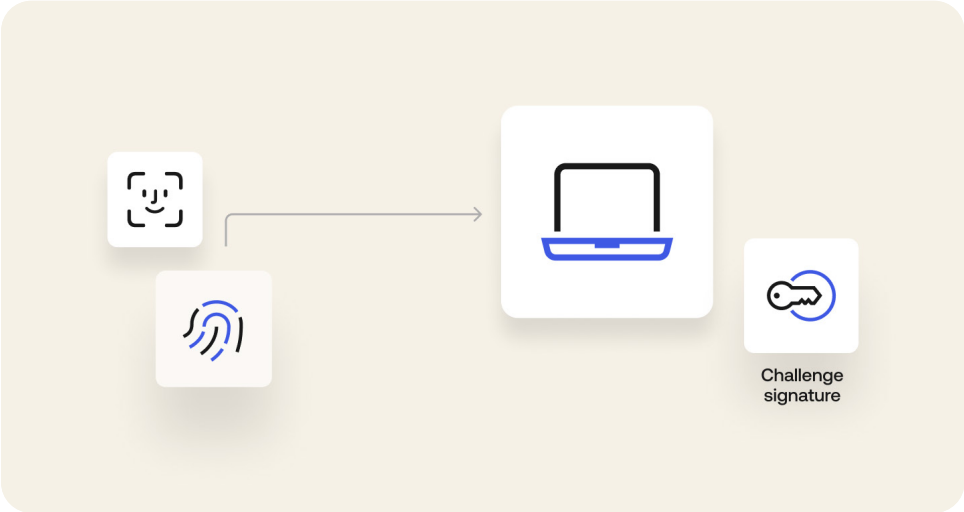
Step 5

The authenticator generates a signature and signs the challenge with the private key based on the domain name.



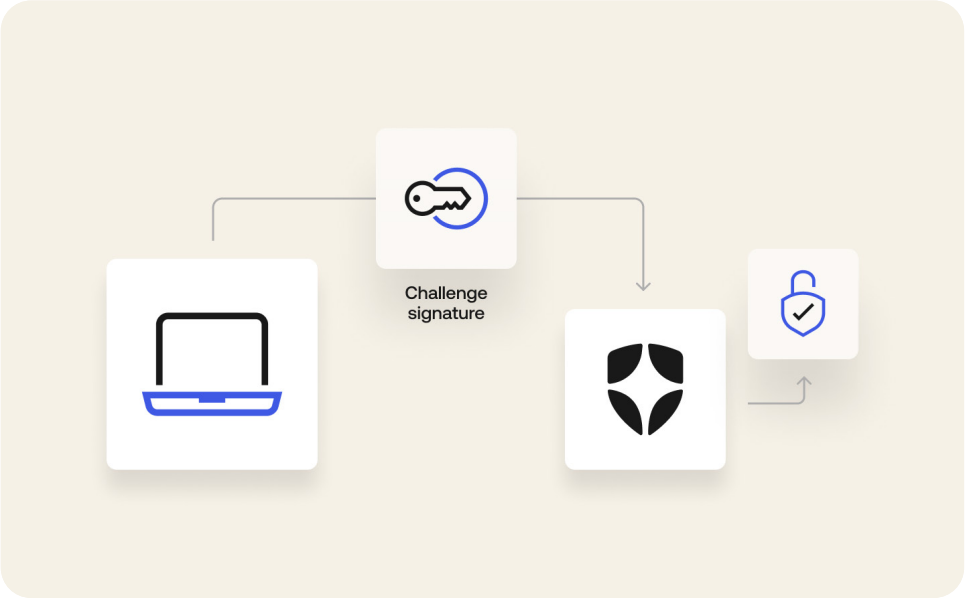
Step 6

The authenticator returns the signed challenge to the user agent.



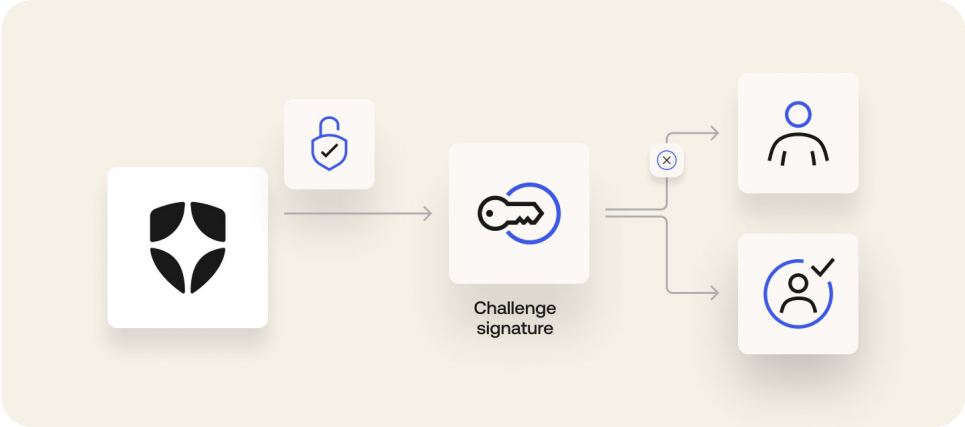
Step 7

The user agent forwards the signed challenge to the relying party to authenticate the user.



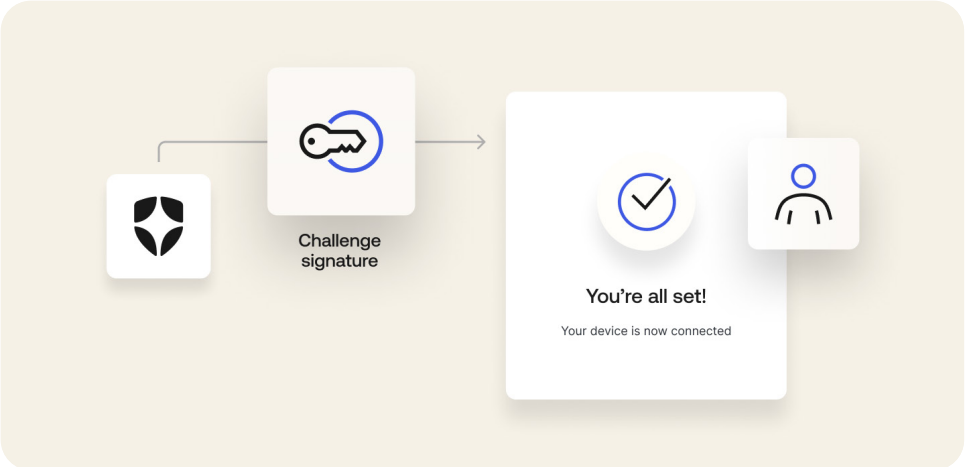
Step 8

The relying party validates the signature with the public key associated with the user account.



Step 9

If the signature is valid, the relying party authenticates the user, issues a token, and redirects the user to the application.



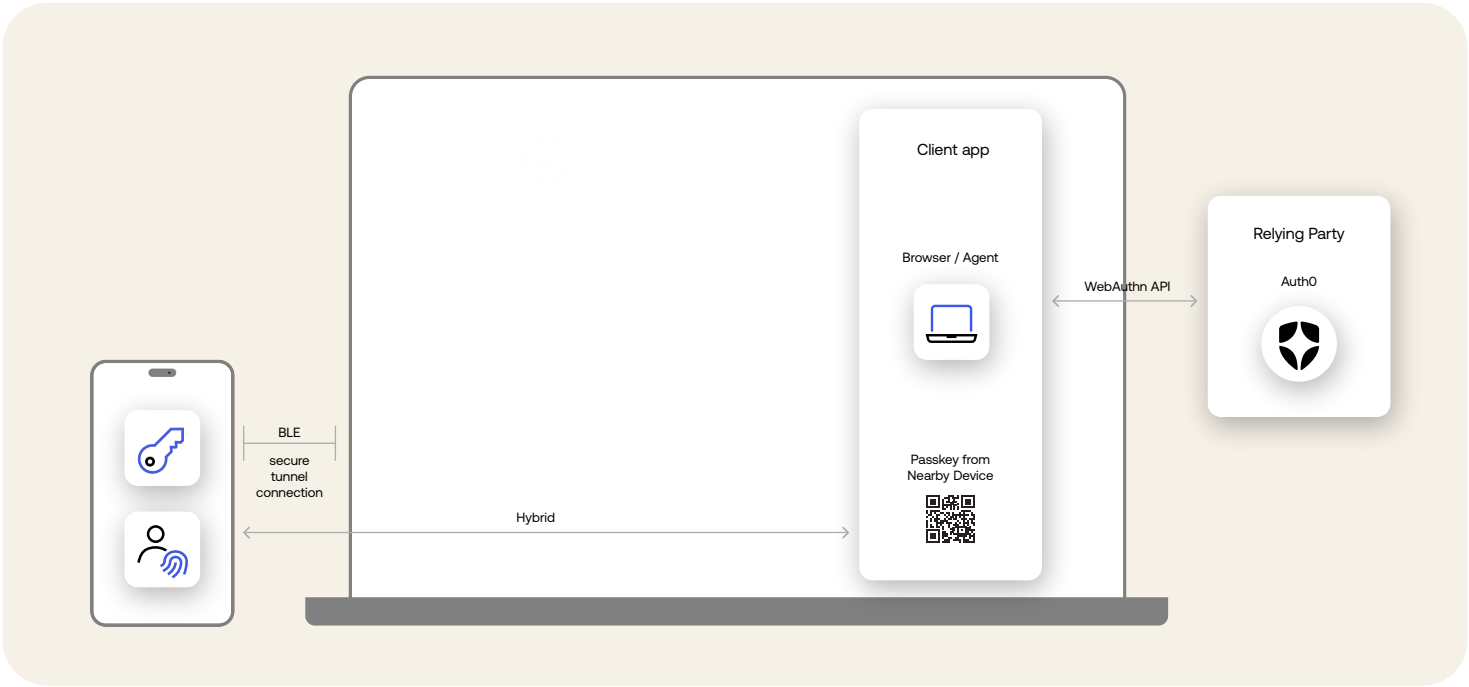
Performing cross-device authentication

Synced passkeys allow users to have a passwordless experience across all their devices in the same ecosystem. However, for users with devices in different ecosystems (e.g. an Apple Macbook and a Google Pixel phone), the FIDO implementation supports end-users to continue their passwordless journey via cross-device authentication.

Cross-device authentication involves scanning a QR code from a device without a passkey, with a device that has a passkey. The protocol leverages Bluetooth technology to verify the physical proximity of the two devices, which is implemented by hybrid transport based on the Client to Authenticator Protocol (CTAP) 2.2 specification. The hybrid transport is intended to connect authenticators with cameras, typically phones, to a client platform which originated the authentication request. This transport involves network communication via a tunnel service, and (Bluetooth Low Energy) transmissions to ensure proximity between the devices, and is implemented by the authenticator and the client platform, not the relying party.

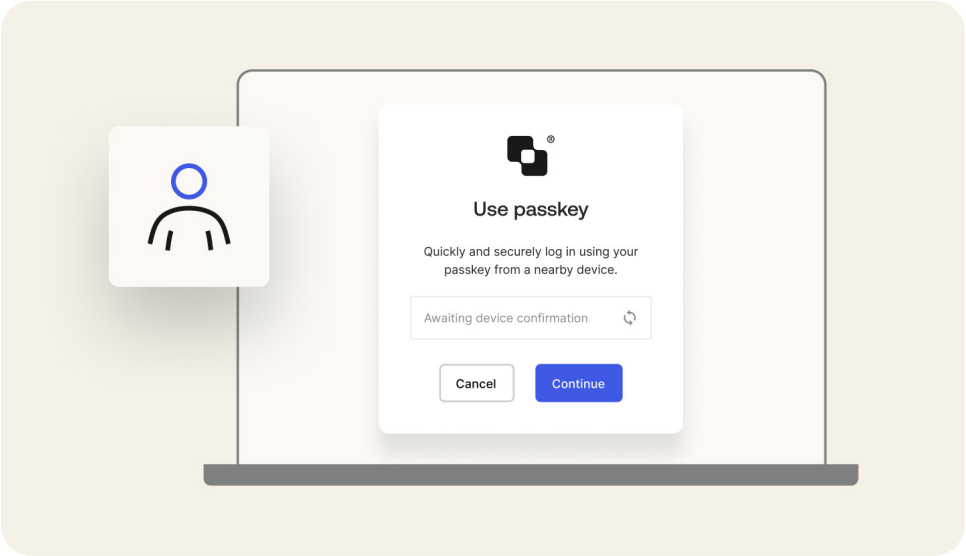
Some criticism of cross-device authentication calls out the security risk associated with using a visible QR code. A bad actor in close proximity may be able to use the QR code for account takeover. Though this is a concern, it drastically reduces the attack surface to someone in close proximity, as opposed to passwords which can be used to compromise an account from anywhere in the world. Additionally, the QR code used for cross-device authentication can only be used once and expires quickly when not used.

We'll continue our examples from previous sections to help illustrate the mechanics of cross-device authentication. In this scenario, the first device (the laptop) is running on an OS that does not yet have a passkey for the web app, and the second device (mobile phone) is running an OS that already has a passkey for the web app.



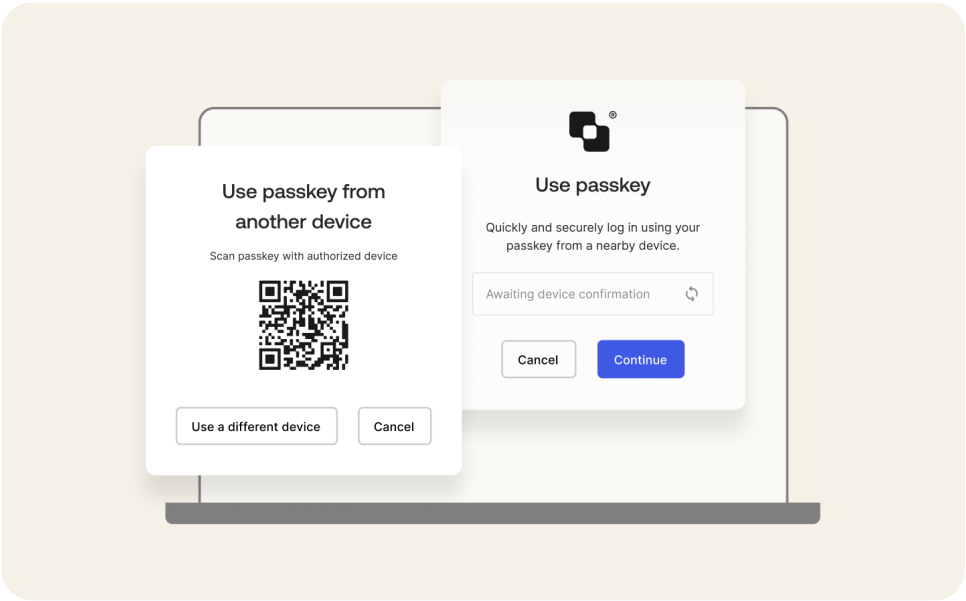
Step 1

The user opens a web application using Device 1 and is offered the option to authenticate using a passkey from nearby devices.



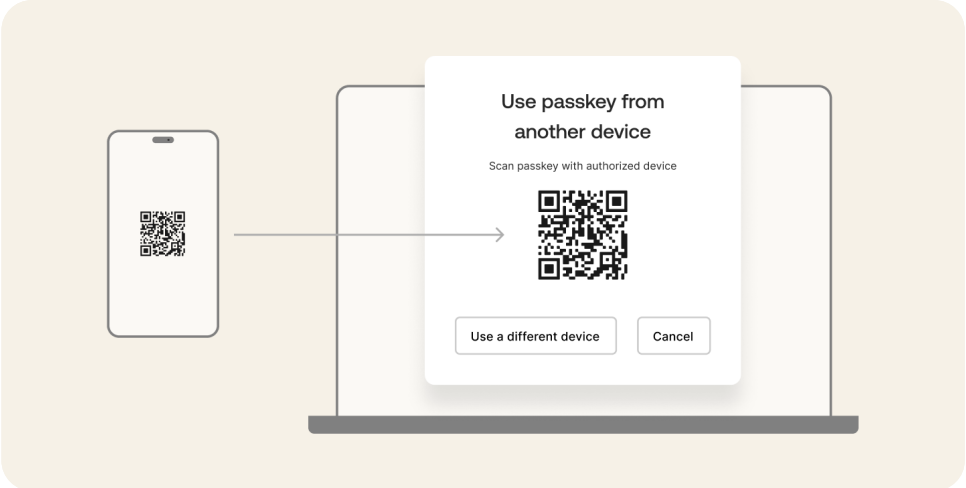
Step 2

The user selects that option, and the web page displays a QR code.



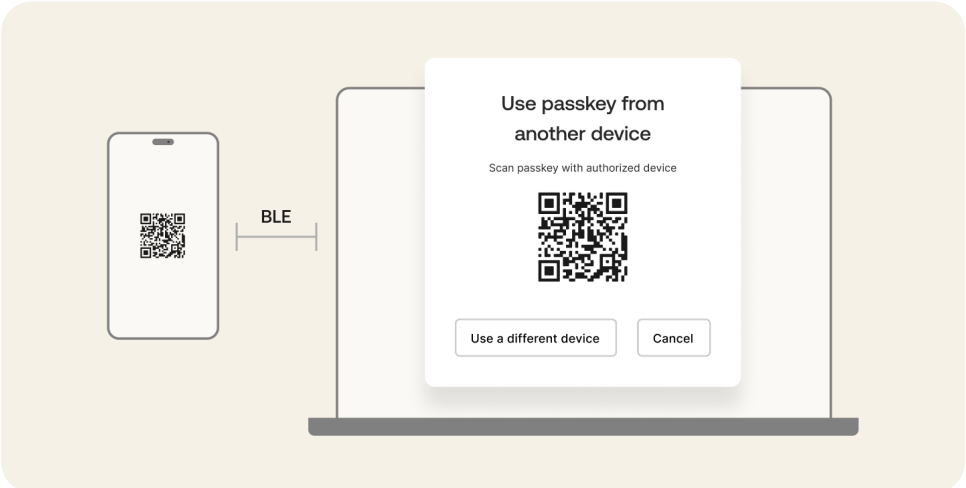
Step 3

The user points Device 2's camera to the QR to initiate an authentication ceremony.



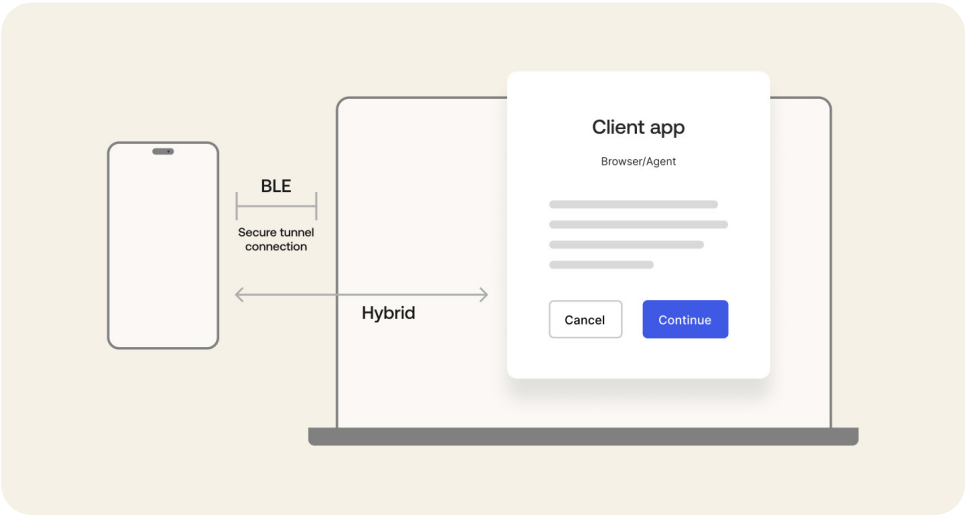
Step 4

To prevent attacks, this transport requires proof of proximity where the notification of the connection attempt comes in the form of a BLE (Bluetooth Low Energy) advertisement.



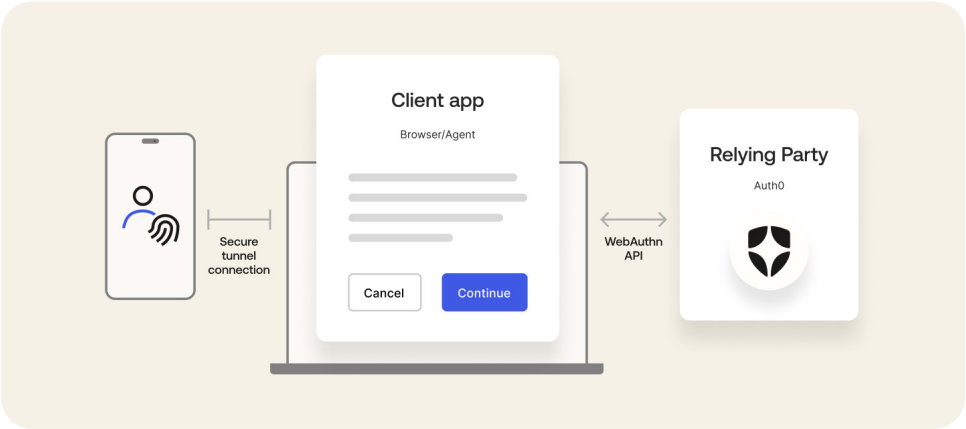
Step 5

A WebSocket tunnel is established between the two devices, and a cryptographic handshake establishes a secure, authenticated connection.



Step 6

Device 2 performs the passkey authentication ceremony, and upon completion, the user is signed into the web application on the first device.



At this point, it is a recommended practice for the app to ask the user whether or not they want to create a new passkey on the first device. If the user chooses to do so, they will now have a passkey for that app available across that device's OS ecosystem.

For example, if the first device is an Apple MacBook and the second device is a Google Pixel phone, then at the start of the flow, the user only had a passkey for the web app on Google devices — but at the end of the flow, the user has both the passkey within the Google ecosystem and a new passkey for the same web app within the Apple ecosystem.

Recovering passkeys

Recall that in the case of device-bound passkeys, the private key is restricted to the authenticator. While this creates a secure authentication solution, it poses challenges for users and organizations regarding the backup of the key data, loss of the authenticator, and adding new authenticators for the user.

The FIDO Alliance recommends a two-step strategy for managing account recovery in this situation:

1. To reduce the number of account recoveries, use multiple authenticators per account
2. To execute account recovery, re-run identity proofing / user onboarding mechanisms

These steps are detailed in Recommended Account Recovery Practices for FIDO Relying Parties.

Synced passkeys, once configured, are available across all devices synced with the passkey provider, and the user doesn't need to enroll multiple FIDO credentials with a relying party to ensure continued access in the event of a lost authenticator.

For example, a user who loses the originating device can log into the OS ecosystem or the password manager on another device (even a new one) to recover access to their passkeys.

However, the fact that synced passkeys roam within one vendor's set of synced devices (or within a password manager service), and access to them is gated to the corresponding account, means that access to the passkey collection is as secure as access to the account on which the roaming features hinge.

For consumer applications, this may be an attractive feature of synced passkeys, as the major operating system vendors have invested considerable expertise in safeguarding access to the underlying accounts. However, many administrators might not be comfortable with this arrangement, and time — and evolving passkey implementations — will reveal the degree to which syncing impacts passkey adoption in different industries and scenarios.

How do I implement passkeys into my app?

Broadly speaking, developers have two approaches when it comes to extending authentication to support passkeys:

- Implementing passkeys yourself, via APIs and SDKs
- Leveraging an Identity service provider

More information about deploying passkeys

FIDO Alliance has published a series of white papers for IT administrators, enterprise security architects, and executives considering deploying FIDO authentication across their organization.

The DIY approach

From the implementation perspective, synced passkeys look just like platform authenticators that do not provide an attestation statement. That means that from the protocol perspective, if your web app already supports WebAuthn, and as long as it doesn't require an attestation response, you technically already support synced passkeys. From the user experience perspective, however, that might not be entirely true. For example:

- The prompts and language you have in your current enrollment pages likely refer to device-bound credentials (e.g., "Sign in faster from this device"), which is no longer the whole truth with synced passkeys.
- Chances are that you are using platform authenticators to enable second authentication factors only, given that before synced passkeys, you were directly responsible for account recovery.

None of the changes above are particularly hard, especially if you already implemented WebAuthn, but you do need to do a bit of work to offer a good experience.

To help developers, in October 2022 the W3C WebAuthn Community Adoption Group and the FIDO Alliance launched passkeys.dev — an online resource that (among other things) contains documentation and tracks device support.

Identity Unlocked

Passkeys with Andrew Shikiar and Tim Cappalli

Shortly after synced passkeys were announced (as multi-device credentials), Andrew Shikiar (Executive Director & CMO, FIDO Alliance) and Tim Cappalli (Digital Identity Standards Architect at Microsoft) joined host Vittorio Bertocci, (Principal Architect at Auth0) on the **Identity Unlocked podcast**.

Tune in to learn about the evolution of FIDO credentials and to gain more develop-focused insights into how multi-device credentials work.

Using an Identity service provider

Identity is difficult — even seasoned professionals find creating effective and efficient implementations to be challenging. Plus, customer expectations are always increasing, with every user comparing each experience to the best ones they've encountered, placing businesses under considerable pressure to continually evolve the UX they deliver.

However, Identity needs must be satisfied without drawing heavily upon precious engineering resources that are needed to extend core competencies — and both of these goals must be satisfied without overlooking regulatory requirements or compromising on security.

For these reasons, many organizations find it both more efficient and cost-effective to integrate an Identity service into their applications and technology stack. Plus, partnering with an identity service provider helps businesses cater to a broader set of requirements in Customer Identity and Access Management (CIAM) including:

- Authentication
- Authorization
- User Management

It's a certainty that established Identity providers will support passkeys, providing a convenient option for application developers to extend their authentication options and keep pace with a rapidly evolving authentication landscape.

For example, if you already have an app configured to use Auth0 for authentication, you'll be able to flip that switch and enable passkey authentication without touching your code at all.

However, different Identity providers offer different features, so due diligence is strongly recommended. Here are a few things to look for as you make your short list:

- Independent and neutral → Your CIAM solution should enable you, not restrict you. That means it should integrate with your existing solutions, should leverage open standards to avoid vendor lock-in, and should work with your preferred cloud provider.
- Comprehensive and customizable → Every customer is unique with complex needs. Your CIAM solution should help you build seamless, consistent, and trustworthy experiences for every type of user.
- Easy to build with, maintain, and use → For virtually every piece of technology, engineering teams aim to reduce effort and time that it takes to deploy, configure, and operate it — and your CIAM solution should support this mission.
- Trusted → Failing to meet compliance requirements, or experiencing an unavailable or degraded service can result in significant brand, legal, and financial consequences. Your CIAM solution should cause you to worry less about these risks.

Don’t wait for perfect when better is already here

Passwords need to disappear, or at least become much less common, and everyone within the Identity industry should work to find ways to leverage the benefits of passkeys while minimizing the drawbacks.

We at Okta are committed to doing our part, both by providing timely, state-of-the-art, developer-friendly features enabling passkeys — and by actively participating in the industry discussions shaping the future of this technology.

Embracing passwordless will get easier as platform vendors and device manufacturers align on standardized flows for recovery, issuance, and non-proliferation. For those who want to introduce or extend passwordless authentication, we recommend looking for authenticators that support:

- Frictionless authentication
- Fewer sign-in errors
- Easy user enrollment
- Resistance to phishing attempts

For what it’s worth, passkeys check all these boxes.