Fosdem 2024

Beyond passwords: secure authentication with passkeys

Identity and Access Management
devroom
Hello, I’m Remy,
Co-founder of passbolt

$ whoami

/passbolt
@stripthis

@passbolt@mastodon.social

passbolt,
FIDO CP-SIG
What is authentication?

Asserting a user identity using something they:

- know (passphrase, password, pin)
- have (token, certificate, key)
- are (biometric)
- or do (typing pattern, gait)
Password based **authentication**

Security issues:
- Credential stuffing.
- Phishing.
- Password loss.
- Bruteforce (online)
- Bruteforce (offline, in case of leak).
~ Adversary in the middle (network)
~ Password logging.
~ User enumeration

Implementation considerations:
+ Checking against breaches & entropy
~ User training
+ Account recovery
+ Captcha (+GDPR) / WAF / Alerts
+ “Costly” hashing mechanism (bcrypt)
+ HTTPs pinned and well configured
+ Additional client side hashing?
~ Vague error messages & constant time?
Who has setup passkeys as a user?
As a developer?
What is a passkey?

Passkeys are passwords replacements. They are public/private key pairs used for user authentication using cryptographic signatures.

Passkeys are user credentials that are discoverable (by the browser, websites, apps).

They are stored within applications or security keys. They may be synced across devices.
Cannot be exported

Exportable and transferable within a given ecosystem (apple, google, etc.)

Credential recovery with another device and/or via provider

Device Bound Passkeys
Ex: yubikey, solokeys, etc.

Cannot be exported

No credential backup, no recovery

Customizable UX

Custom authenticator

Additional signals

Transaction signing

Attestation possible

Bound to an origin (app, domain)

Phishing resistant

Discoverable

Synced Passkeys
Ex. passkey on apple / google devices

App Level Passkeys
Ex: auth app for bank(s)

Synced Passkeys
Ex. passkey on apple / google devices

Exportable and transferable within a given ecosystem (apple, google, etc.)

Credential recovery with another device and/or via provider
Conflicting requirements? More options!

The complex art of balancing the standards to cater for different audiences...

Enterprise
(Security & auditing)

Certifications (NIST, etc.)

Strong authenticator attestation (MDS)

Phishing resistance

Domain bound

Stronger user verification (slower UX)

HTTPS only

Consumer
(Ease of use & privacy)

Privacy (no fingerprinting)

Lighter touch points (speed optimized UX)

Passkey sharing / exports

Authenticator “hinting” (AAGUID unofficial list)
FIDO2 Project
A joint effort between the FIDO Alliance and the W3C.

Authenticator
Proves you are you, either on the device (platform) or off device (roaming).
Examples: iOS Keychain, Microsoft Hello, Yubikey, Dashlane, Etc.

Client
The web-browser & client side software (JavaScript + Credential Management API)
Examples: Chrome, Firefox, Safari, Etc.

Relying Party (RP)
The website that wants to authenticate you
Examples: Google, Mastodon, Etc.

CTAP 2.0 + proprietary apis & monkey patching
Client to authenticator protocol
Web Authentication
HTTPS
WebAuthn
Which ceremonies are supported?

**Attestation** (Registration)

When an authenticator registers a new key pair with a service. Either first one or as an alternative for recovery.

**Assertion** (Login)

When a user chooses to log into a service.

Not supported: listing and deletion of passkeys. RPs are in charge of this (potentially leading to accessibility / security issues).
Attestation ceremony

e.g. a client sends a registration request

```plaintext
Authenticator (App/Device)  Client (Browser)  Relying Party (Website)

Assert parameters
Assert crypto supported
Check existing credential
(Collect user gesture)
Generate credential
Generate signature

navigator.credentials.create(
  PublicKeyCredentialCreationOptions)

authenticatorMakeCredential(
  clientDataHash, rp, user,
  pubKeyCredParams, ..)

POST /webauthn/attestation/options
{username}

200 OK
PublicKeyCredentialCreationOptions:
{rp, user, challenge, pubKeyCredParams,
authenticatorSelection, attestation, etc.}

POST /webauthn/attestation/response
AuthenticatorAttestationResponse:
{clientDataJSON, attestationObject}

200 OK
Set-cookie: session

Assert RP ID
Assert Key
Verify signature etc.

Attestation ceremony
e.g. a client sends a registration request
```
Assertion ceremony

e.g. authentication flow (login flow)

**Authenticator (App/Device)**
- Assert params
- Check credential exist (Collect user gesture)
- Generate signature

**Client (Browser)**
- `navigator.credentials.get`
  - `PublicKeyCredentialRequestOptions`
- `authenticatorGetAssertion`
  - `{rpId, clientDataHash, ..}`

**Relying Party (Website)**
- POST /webauthn/assertion/options
  - `{username}`
- 200 OK
- **PublicKeyCredentialRequestOptions**:
  - `{challenge, rpId, allowCredentials, userVerification, ..}`
- POST /webauthn/assertion/response
  - `AuthenticatorAssertionResponse`
    - `{clientDataJSON, authenticatorData, signature, userHandle}`
- 200 OK
- Set-cookie: session
- Verify sig
- Assert RP ID etc.

**Authenticator (App/Device)**
- Assertion Signature
  - `{selectedCredential id and username, authenticatorData, signature, ..}`

**Client (Browser)**
- 200 OK
- **Set-cookie: session**
What about account recovery?

<table>
<thead>
<tr>
<th>RPs</th>
<th>Authenticators / Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ More than one passkeys</td>
<td>✓ Another device</td>
</tr>
<tr>
<td>✓ Password</td>
<td>~ Recovery contact</td>
</tr>
<tr>
<td>✓ Magic Link</td>
<td>~ Custom procedure</td>
</tr>
</tbody>
</table>

Example for iCloud

“Passkeys can be recovered through iCloud keychain escrow, which is also protected against brute-force attacks, even by Apple. [...]

To recover a keychain, a user must authenticate with their iCloud account and password and respond to an SMS sent to their registered phone number. After they authenticate and respond, the user must enter their [lost] device passcode. iOS, iPadOS, and macOS allow only 10 attempts to authenticate. After several failed attempts, the record is locked and the user must call Apple Support to be granted more attempts. After the tenth failed attempt, the escrow record is destroyed.

Optionally, a user can set up an account recovery contact [...]”

How does it look like?
Registration on MacOS / Chrome (01/24)
Registration on MacOS/Chrome/iOS (01/24)

Create a passkey
Choose how you want to create a passkey for www.passkeys.io

- iCloud Keychain
- Use a phone, tablet or security key
- Your Chrome profile

Create a passkey on a phone or tablet
Scan this QR code with a camera on the device where you want to create a passkey for demo.yubico.com

Use a different device

Follow the steps on your device

Sign In

Use Face ID to sign in?
A passkey for “Yubico demo user” will be saved in iCloud Keychain and available on all your devices.

Continue
Registration on MacOS/Safari or Firefox (01/24)

Currently, YubiKeys can store a maximum of 25 passkeys. (if you've never entered a PIN when you registered your Yubikey you are not using resident keys).

Or from Chrome...

"Currently, YubiKeys can store a maximum of 25 passkeys." (if you've never entered a PIN when you registered your Yubikey you are not using resident keys).
Managing passkeys on MacOS/Chrome
Managing Pass
words keys on iOS
Passkeys security issues

Security issues:
- Device / platform account loss
- Passkey management & review
- Passkeys transfer/sharing
- User enumeration
- CA revocation
- Quantum computers? Weak PQC?
- & more (UI redressing, proximity)

Implementation considerations:
- Account recovery? More passkeys?
- User training? Better UX? Alerts?
- Better signalization of sharing props?
- Random username / fake credential ids?
- Forced rotations? Devices exclusion?
- Crypto agility?
- RTFM?
Passkeys other issues

Other issues:
- Fragmented end user experience
- Specs depth & stability
- Entry barrier for authenticators
- Pay to play

Other considerations:
~ UX Working group
~ Passkeys “the good parts”? RP Guidelines?
~ Monkey patching? EU Fines?
~ Pooling of resources for open source actors
Questions? 🍅?
Thank you Fosdem 🍻❤️