USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

The State of Passkeys: Studying the Adoption and Security of Passkeys on the Web

Louis Jannett, *Ruhr University Bochum;* Andreas Mayer and Maximilian Westers, *Heilbronn University of Applied Sciences;* Vladislav Mladenov, *Ruhr University Bochum;* Christian Mainka, *University of Wuppertal;* Jörg Schwenk, *Ruhr University Bochum*

https://www.usenix.org/conference/usenixsecurity26/presentation/jannett

This paper is included in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Proceedings of the 35th USENIX Security Symposium is sponsored by

جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

The State of Passkeys: Studying the Adoption and Security of Passkeys on the Web

Louis Jannett [1], Andreas Mayer [2], Maximilian Westers [2],
Vladislav Mladenov [1], Christian Mainka [3], and Jörg Schwenk [1]

[1]Ruhr University Bochum
[2]Heilbronn University of Applied Sciences
[3]University of Wuppertal

Abstract

Passkeys provide a secure and phishing-resistant authentication method based on FIDO2 and WebAuthn. They have recently gained popularity, with an increasing number of websites adopting them. Nevertheless, a comprehensive security analysis that evaluates such websites at scale has not been fully addressed. We present PASSKEYS-RADAR, a continuously updated dataset that tracks the deployment of passkeys on the Internet since 2021. To build this dataset, we aggregated diverse sources, including community directories, Tranco 1M, CrUX 18M, and historic Internet archive data. We analyzed the collected data of 872 passkey-enabled websites and shed light on how passkeys are implemented and managed. We identify major differences in how websites allow users to add or delete passkeys and find that websites request authenticators to use deprecated cryptographic algorithms.

To perform a comprehensive security evaluation of passkey-enabled websites, we developed PASSKEYS-ATTACKER. The tool allows for precise manipulation of WebAuthn messages at every step of the protocol and integrates 15 attack types of which 10 were not covered in previous work. Among them, 2 attack types have *critical* CVSS scores. We discovered them on 18 out of 103 evaluated websites. These attacks take over user accounts, delete their passkeys, or lock them out of their accounts. Nearly half of the tested sites (53) were vulnerable to at least one attack with a *high* CVSS score, exposing users to threats such as phishing and session fixation.

1 Introduction

"Hackers don't break in – they sign in" [66]. In May 2025, security researchers found 184 million leaked passwords [96]. Within a few weeks, this estimate was revised to 16 billion passwords, making it the largest known password compromise to date [95]. Such incidents highlight the urgent need to move beyond passwords towards passwordless authentication.

Passkeys. Passkeys are the modern, phishing-resistant, and user-friendly authentication method that replaces passwords with strong public-key cryptography for improved security. They are based on standards developed by the Fast IDentity Online (FIDO) Alliance [36] and W3C [89]. Today, passkeys are supported across all major browsers, including Chrome, Firefox, and Safari, as well as on platforms such as Windows, macOS, iOS, and Android [77]. Large corporations such as Apple, Google, and Microsoft drive the adoption of passkeys. These companies are members of the FIDO Alliance and actively integrate and promote passkeys in their ecosystems.

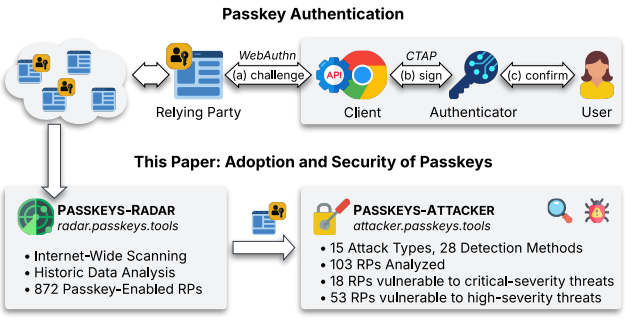


Figure 1: Scope of this Paper. First, we quantify the adoption of passkeys on real-world websites using PASSKEYS-RADAR (§3) and we study how they are used and managed by these sites (§4). Second, we implement all security tests from the WebAuthn specification into PASSKEYS-ATTACKER (§5), and evaluate how secure websites implement passkeys (§6).

Passkey Authentication Scheme. Passkeys are built on two core protocols, jointly referred to as FIDO2: (1) Web Authentication API (WebAuthn), which defines how the Relying Party (RP) communicates with the client, and (2) Client-to-Authenticator Protocol (CTAP), which specifies how the client communicates with the authenticator. This paper examines WebAuthn and evaluates the security of RPs, see Figure 1.

(a) First, the RP initiates the authentication process by providing a randomly generated challenge to the client.
(b) The client then asks the authenticator to sign this challenge with the private key that it has stored for that RP.

(c) The user confirms the request through a gesture and, if required, is verified via biometrics or PIN.

Finally, the signed challenge is returned to the RP, which verifies it using the public key that the user registered when enabling passkeys for the website. Surprisingly, there has been little security research on how RPs are verifying the actual authentication process on the web [44, 56, 98].

To fill this gap, we formulated four research questions (RQs) to better understand passkey adoption and security.

RQ1: How accurately do current sources reflect the adoption of passkeys? (§3). An essential prerequisite for studying passkeys on the web is to determine which websites currently support passwordless authentication. Several methods exist for this. For example, Jannett et al. [50] monitored WebAuthn API calls on the Tranco 1M. Community directories also list sites that have adopted passkeys, such as `fidoalliance.org`, `keepersecurity.com`, and `passkeys.directory`. Websites may further announce support through well-known URLs. However, the accuracy of these data sources remains unclear. To address this, we built PASSKEYS-RADAR, a fully automated scanner that continuously detects passkey-enabled websites. We merged 12 public directories and complemented them with monthly scans of the CrUX 18M. Our results show that scanning domains uncovers about 125% more RPs than all community directories combined. PASSKEYS-RADAR identified 872 RPs that support passkeys.

RQ2: How are passkeys used on websites? (§4). At first glance, managing passkeys appears straightforward, but our study proved us wrong. We manually created test accounts on 208 websites and registered passkeys on them. Even during registration, we observed several peculiarities. Some sites skipped password confirmation for adding a passkey, although they required it for changing a password. Others did not allow deleting a passkey once registered or supported only a single passkey. 61 sites accepted passkeys only as a second factor, still requiring a password. Some used the same passkey for both passwordless login and two-factor authentication, depending on the user-chosen passkey authentication mode. All but one site requested deprecated cryptographic algorithms, and only 90 of 208 RPs used the recommended ones.

RQ3: How comprehensive are state-of-the-art passkey security frameworks? (§5). Web security largely depends on RPs correctly implementing passkeys. To study this at scale, we reviewed existing tools from prior work. Research so far has covered authenticators [15, 54, 55, 64, 82] and provided formal analyses [9, 10, 11]. Yet, standardized WebAuthn validations on the RP side have received little attention. Previous studies explored other attacker models, such as malware [56] or malicious browser extensions [98], or performed manual checks on a few validations [44, 45]. To address this gap, we developed PASSKEYS-ATTACKER, a framework that can intercept, analyze, and modify all WebAuthn messages. To our knowledge, it is the first system that automates active and extended attacks on passkey registration and authentication.

RQ4: How many websites are compliant with the passkey security considerations? (§6). The WebAuthn standard [89] defines detailed security and privacy requirements. From these, we derived 15 attack types and implemented 28 detection methods in PASSKEYS-ATTACKER. Instead of assigning custom names and impact to each attack type, we relied on standard frameworks. We used CWEs to categorize our attack types. To assess their severity, we mapped them to known CVEs and rated them using the CVSS score. For 7 cases without existing CVEs, we assigned scores based on comparable vulnerabilities. We then analyzed 103 RPs with PASSKEYS-ATTACKER. The results are alarming: none of the tested RPs passed all security checks mandated by the standard. Even more critically, 18 RPs contained at least one vulnerability rated as critical and 53 as high. Such flaws enable attackers to bypass authorization, delete passkeys of other users, or prevent them from accessing their accounts.

Contributions. We make the following contributions:

- We systematize the passkey registration and authentication flows with a focus on security (§2).
- To support future research, we introduce PASSKEYS-RADAR, the most comprehensive automatically generated and continuously updated directory of passkey-enabled websites (§3). PASSKEYS-RADAR identified 872 RPs with support for passkeys. On 208 of them, we successfully registered passkeys and analyzed their management, supported features, and whether passkeys are used for passwordless authentication or 2FA (§4).
- We present PASSKEYS-ATTACKER, the largest set of automated security tests for passkeys to date. Beyond predefined checks, it allows fine-grained manipulation of each step in the passkey message flow (§5). Using PASSKEYS-ATTACKER, we conducted a semi-automatic analysis of 103 passkey-enabled websites. None passed all required security tests, and over half contained vulnerabilities with a high or critical CVSS score (§6).

2 Passkeys and How They Work

The FIDO Alliance [36] is an open industry group that develops and promotes passkey standards. Our work focuses on websites and, thereby, on the WebAuthn protocol [89].

2.1 Passkeys Protocol Flow

Surprisingly, there is no dedicated source that directly describes the WebAuthn protocol flow. Instead, the flow is only implicitly defined in the WebAuthn standard [89], which we summarize in the following.

Protocol Overview. To log in to RP, a user must first register a passkey with their account (→ *registration phase*). If the user does not yet have an account, they can create one and immediately set up a passkey as their primary login

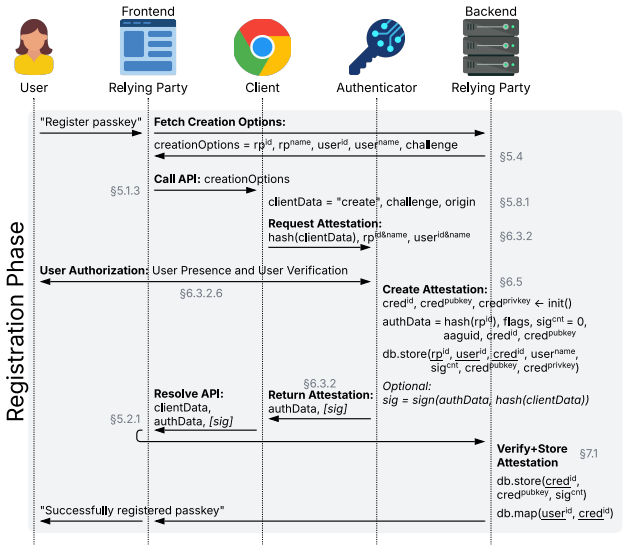


Figure 2: Passkey Registration Phase (we refer to the corresponding sections in the WebAuthn standard [89]).

method. For existing accounts, the user must first authenticate using their current login method, i.e., passwords, Single Sign-On (SSO), or email magic links. Once authenticated, they can navigate to their account settings to register a new passkey. After registration, the user can use the passkey to sign the RP's challenge and log in to the RP (→ *authentication phase*).

Protocol Roles. Providing a passwordless login experience with passkeys requires orchestration between multiple roles.

(1) *Relying Party:* The RP is the entity to which the user wants to authenticate, for example, a website.

(2) *Authenticator:* The authenticator generates the public and private key pair during registration. Later, it uses the private key to sign the challenge during authentication. Authenticators can be physical devices [99], for example, a phone or hardware security key. They can also be software-based [43]. Examples include a password manager like Chrome's built-in Google Password Manager and 1Password. Likewise, they can be integrated into the operating system's keychain, such as Windows Hello or the iCloud Keychain.

(3) *Client:* The client facilitates communication between RPs and authenticators. In our web scenario, it is the browser.

2.2 Passkey Registration Phase

Figure 2 was generated by utilizing the implicitly defined registration phase within the WebAuthn standard [89]. We assume that the user is logged in and initiates the passkey registration from their account settings.

Fetch Creation Options. First, the RP sends a request to its back end. It uses the Fetch API [69] to obtain fresh creationOptions. These options include:

(1) rp^{name} and $user^{name}$ are human-readable names for the RP and the user.
(2) $user^{id}$ is a unique and random identifier for the user.
(3) challenge is a fresh session-bound random value.
(4) rp^{id} is a domain that identifies the RP. By default, it is set to the effective domain of the origin on which the passkey is created. It mitigates phishing by protecting passkeys from being accessed by other origins.

The creationOptions can include extra parameters, such as preferences for cryptographic algorithms or types of authenticators (e.g., hardware vs. software).

Call API. The RP's front end passes the creationOptions to the client's `navigator.credentials.create` API [67].

Request Attestation. The client constructs the clientData. It contains the current context ("create" for the registration phase), the challenge, and the RP's origin. The client must verify that the specified rp^{id} matches the RP's origin or is a registrable suffix.

User Authorization. Covert passkey registrations would enable user tracking. To prevent this threat, the user must confirm each registration. There are two possibilities for this purpose: *user presence* and *user verification*. User presence means that the user has intentionally approved the creation of the passkey. For example, the user has clicked on a button on a hardware security key. User verification means that the user was explicitly authenticated during registration. For example, the user entered a PIN or used biometrics.

Create Attestation. The authenticator generates a new credential, which consists of a random identifier ($cred^{id}$), a public key ($cred^{pubkey}$), and a private key ($cred^{privkey}$). It then assembles the authData, which includes: (1) the SHA256-hashed rp^{id}, (2) flags indicating whether the user was present and verified, and whether the credential is eligible for backup and backed up, (3) sig^{cnt} – a signature counter that increments with each authentication, (4) AAGUID – an Authenticator Attestation Globally Unique Identifier (AAGUID) that identifies the authenticator's manufacturer and model, and (5) the $cred^{id}$ and $cred^{pubkey}$.

Return, Resolve, and Verify. After the authenticator returns the attestation to the client, the client augments its clientData and completes the API call. As a result, the RP's front end receives the attestation and forwards all data to its back end. The back end validates the attestation and, if successful, associates the $cred^{pubkey}$ with the user's account.

2.3 Passkey Authentication Phase

We depict the authentication phase in Figure 3.

Discoverable vs. Non-Discoverable Mode. There are two authentication modes: the *discoverable mode* and the *non-discoverable mode*. In the discoverable mode, there's no initial indication of which user is logging in to the RP. Typically, the RP's login page only displays a "Login with Passkey" button. The authenticator retains the $user^{name}$ alongside the

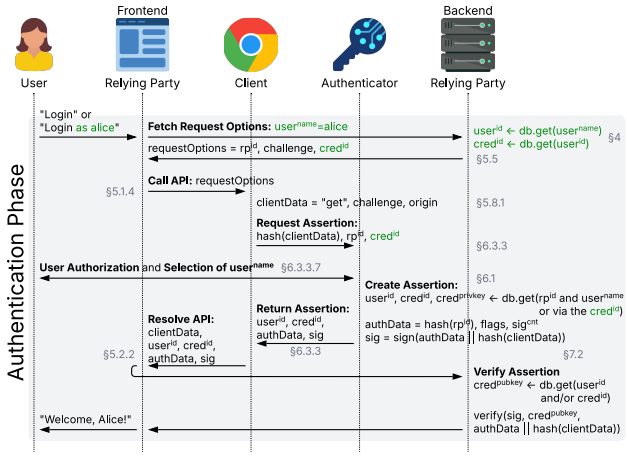


Figure 3: Passkey Authentication Phase (we refer to the corresponding sections in the WebAuthn [89] standard). Green parameters and steps are only used in *non-discoverable mode*.

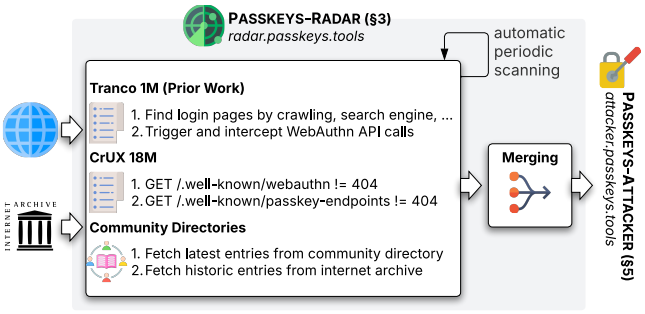


Figure 4: PASSKEYS-RADAR enables our fully automatic detection of passkey implementations on websites.

rp^{id} and $cred^{privkey}$, enabling it to request the user to choose the $user^{name}$ for login. For example, consider a user who has two different accounts on the same RP. In the *discoverable mode*, the user does not provide any hint to the RP about which account the user wants to use to log in. Instead, the authenticator prompts the user to choose one of those two accounts. This task is easy for software authenticators, such as a browser's password manager, but not for hardware security keys. Thus, there is the *non-discoverable mode*. In this mode, the user needs to submit a $user^{name}$ directly on the RP's login page. This way, the RP back end can perform a database lookup of the $user^{name}$ and determine the registered $cred^{id}$. The $cred^{id}$ is passed to the authenticator, allowing it to select the appropriate $cred^{privkey}$ for signing the challenge.

Fetch Request Options. First, the RP's front end obtains fresh requestOptions from its back end. They include the rp^{id}, a fresh challenge, and, in non-discoverable mode, the $cred^{id}$ of the credential that should be used to sign the challenge.

Call API. The RP's front end passes the requestOptions to the client's `navigator.credentials.get` API [68].

Request Assertion. The client constructs the clientData. It includes the current context, which equals "get" for the authentication phase. Also, it includes the challenge and the RP's origin. The client sends a hash of the clientData to the authenticator, along with the rp^{id} and, in non-discoverable mode, the $cred^{id}$.

User Authorization. To prevent silent authentication and protect against stolen credentials, the user's presence and identity must be verified. In discoverable mode, the authenticator additionally prompts the user to choose a $user^{name}$ for login.

Return Assertion. The authenticator constructs the authData, which includes the hashed rp^{id}, flags, and the signature counter. It loads the $cred^{privkey}$ corresponding to the $user^{id}$ and $cred^{id}$, and uses it to sign the authData and clientDataHash (which includes the challenge).

Return, Resolve, and Verify. The authenticator returns the assertion to the client, which augments its clientData and resolves the API call. The front end forwards all data to the back end, which verifies that the correct challenge was signed using the $cred^{pubkey}$ that it has stored for the $user^{id}$ and $cred^{id}$.

3 PASSKEYS-RADAR: Fully Automatic Detection of Passkey Implementations on the Web

This section answers RQ1 and presents PASSKEYS-RADAR, a fully automated large-scale method for detecting passkey-enabled websites, as detailed in Figure 4.

3.1 Detecting Passkey-Enabled Websites

We combine three techniques to build the most comprehensive collection of passkey-enabled websites to date. To support future large-scale analyses of passkeys, we run automated periodic scans that continuously update this dataset. This provides researchers access to the most current list and allows us to monitor the long-term evolution of passkeys.

Tranco 1M (Prior Work). Recently, Jannett et al. [50] have published a tool that scans the Tranco top 1M websites and their login pages. They automatically detect passkey-enabled websites, but they did not further investigate this information. Since the authors made their dataset and source code publicly available [85], we could build upon their work. Investigating their source code, we discovered that they solely used WebAuthn API calls to detect passkey logins. We enhanced this approach with further techniques as outlined below.

CrUX 18M – Well-Known Documents. Well-known documents [94] serve various purposes, such as proving domain ownership [87], delivering metadata [31], or providing machine-readable instructions for responsible vulnerability disclosure [38]. These documents are always located at fixed paths on a website, making them easy to detect automatically. In the context of passkeys, there are two relevant well-known documents: (1) The *webauthn document* [91] is hosted at

`/.well-known/webauthn`. It enables the Related Origin Requests (ROR) feature, which allows a single passkey to be shared across multiple related sites (see §6.2.3). (2) The *endpoints document* [3] is located at `/.well-known/passkey-endpoints`. It allows RPs to signal their support for passkeys. In addition, it specifies where users can create new passkeys and manage existing ones.

Starting in February 2025, we have conducted monthly scans of these two documents. Each month, we retrieve the latest Chrome User Experience Report (CrUX) dataset from Google BigQuery [20], which covers 18M sites. We checked each site for the presence of these well-known files.

Community Directories. Several websites curate lists of passkey-enabled websites. We refer to these as *community directories* because they are manually maintained by companies or the community. When a website adopts passkeys, it must be added to these directories by hand to indicate its support. These lists are commonly published by password managers [1, 70, 72, 73, 74, 76],[1] identity management providers [63], community-driven platforms [2, 71, 92, 93], and the FIDO alliance itself [37]. For instance, 1Password leverages its list to notify users when a website in their vault has added passkey support, encouraging them to upgrade their login credentials.

To compile our set of community directories, we searched Google for the terms "passkey/webauthn/2fa directory/websites/lists/community" and reviewed the first five pages of results. We excluded forums, news articles, and blog posts announcing new passkey adoptions. Since our focus is on websites, we also removed resources listing operating systems, user agents, or authenticators. Finally, we used advanced search features in ChatGPT and Perplexity, but these revealed no further directories beyond those already identified.

Since February 2025, we have conducted weekly scans of these community directories to collect the most recent entries. To analyze historical trends, we also extended our tool to retrieve all archived snapshots of these directories from the Internet Archive's Wayback Machine [90], with data dating back to May 2021.

3.2 Merging Passkey-Enabled Websites

Each detection technique has its limitations. For example, community-maintained directories may overlook lesser-known websites, and well-known documents are still not widely deployed on the web. To address these shortcomings and improve the overall completeness of our dataset, we adopt a methodology inspired by the Tranco top sites ranking [61, 62]. At the core of this approach is the merger that aggregates entries from all detection sources into a single, unified list.

While merging might seem trivial at first glance, it involves significant complexity due to the heterogeneous structure and information content of the sources. All entries are consolidated into a single input file, each containing at least a domain, a name, or both. To align entries, we apply normalization and pattern-matching techniques, such as stripping whitespace and converting to lowercase. We then compare names using substring matching. For example, "Google" is considered part of "Google LLC", so we merge these entries. Subdomains like `login.google.com` are reduced to their registrable base `google.com`. We merge domains if they match or share the same base domain, such as `google.com` and `google.co.uk`. Nevertheless, some entries remained distinct despite using the same authentication system, such as `gmail.com` and `google.com`. To resolve such cases, we additionally leveraged the Tracker Radar Entity Map [30], which identifies domains and names belonging to the same organizational entity. This prevents large providers from disproportionately inflating the overall passkey adoption rate.

[1] E.g., 1Password, 2Stable, Bitwarden, Dashlane, Enpass, Keeper

4 Passkeys in the Wild: Adoption and Real-World Usage on Websites

In this section, we answer RQ2 by presenting the first comprehensive analysis of passkeys on real-world websites. Our study consists of two parts: (1) We analyze the adoption of passkeys over the past four years, highlighting their growth across websites. (2) We then conduct the first in-depth investigation of how passkeys are implemented in practice, assessing, among other aspects, which features of the WebAuthn standard [89] are employed and how resilient these implementations prove to be. Moreover, we report on how passkeys can be managed by users, such as being added or deleted, based on our observations during the analysis.

4.1 Part 1: Adoption of Passkeys

When aggregating all upstream sources (§3.1), we identified 8,523 websites. Many of these were duplicates, as the same sites appeared across multiple community directories. After merging (§3.2), 872 unique websites remained.

Figure 5 depicts the adoption of passkeys on real-world websites over the past four years. Most directories emerged in late 2023 and early 2024, with varying update frequencies. 1Password [1] (`passkeys.directory`) proved to be the most valuable source, offering the second largest but most frequently updated collection of passkey-enabled sites.

The number of sites deploying the *webauthn document* [91] has grown significantly in the last 6 months, from 27 to 177 websites (+650%). In contrast, adoption of the *endpoints document* [3] declined slightly, from 387 to 344 domains (-12%).

By combining multiple detection techniques for the first time, we more than doubled the known number of passkey-enabled sites at the start of our study (see logarithmic scale in Figure 5). As a result, our PASSKEYS-RADAR currently

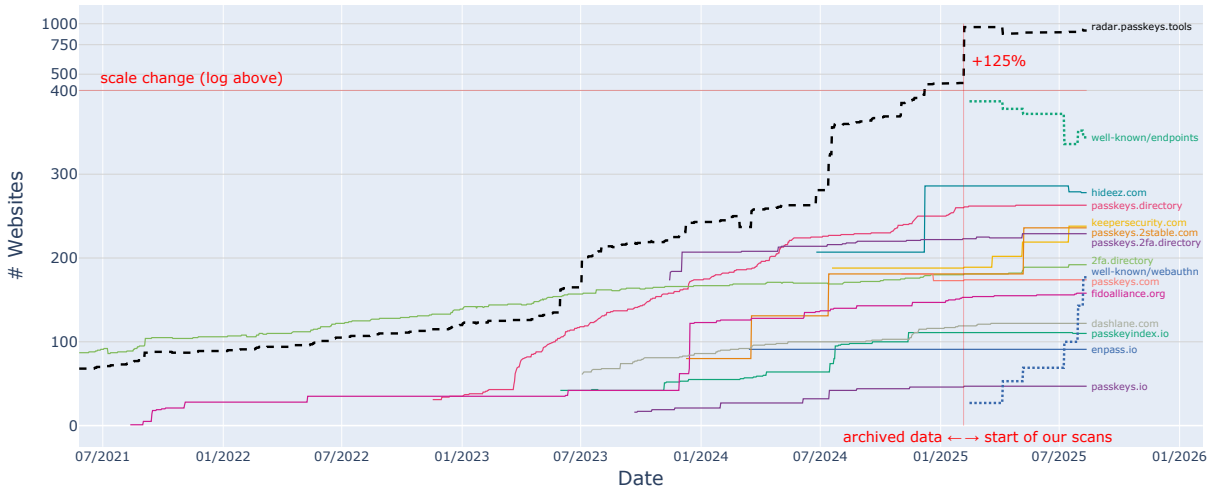


Figure 5: Adoption of Passkeys from 05/2021 to 08/2025.

represents the most comprehensive dataset of passkey-enabled sites, providing a solid foundation for future security analyses.

4.2 Part 2: How Websites Implement Passkeys

As a starting point, we used our merged list from April 22nd 2025, comprising 872 yet unconfirmed passkey-enabled domains. We manually reviewed all 872 sites and excluded 664 from further analysis for 19 different reasons, as detailed in Table 3. For example, we excluded 152 false positives caused by missed duplicates and misclassifications (§7). We also dismissed 334 potential passkey-enabled websites that could not be confirmed because we were unable to create accounts. Thus, we successfully verified 386 confirmed passkey-enabled websites. Among them, 178 sites relied on third-party passkey providers (e.g., `aid.no`). Since all sites using the same provider shared the same SDK-based setup, we analyzed each provider only once to avoid inflated results. After deduplication, 208 confirmed independent passkey implementations remained that we were able to analyze.

Confirming Passkey Registrations. When testing passkey registrations, we observed that websites handle confirmation in different ways. Registering a passkey is a sensitive action, comparable to changing a password, as it grants lasting access to the account. Only 71 sites (34%) asked for an extra confirmation, most often through a password (37), email (25), or SMS (6). The majority of sites (66%) allowed immediate registration without any additional confirmation. These sites are more exposed to attacks in which an attacker could trick a user into unintentionally adding the attacker's passkey to their account [19], something that would be much harder if an extra confirmation were required.

Removing Passkeys. When users lose access to their authenticator, e.g., due to theft, it is crucial that they can remove passkeys from their account to prevent unauthorized access. Most websites provide some form of passkey management in the account settings, but there are exceptions.

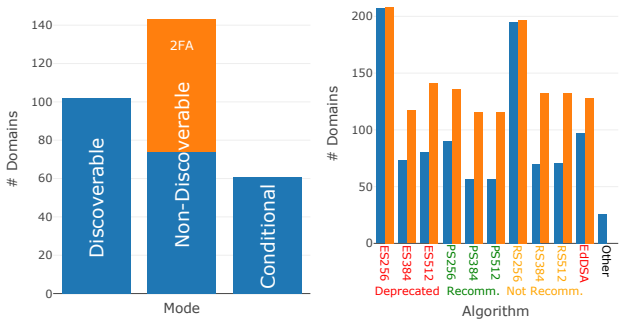
On 7 sites, a passkey can only be registered immediately after login and cannot be removed later, effectively creating a one-way upgrade: once a passkey is added, users are locked into it. A similar restriction exists on 6 sites where a passkey must be registered at account creation and cannot be replaced afterward. On 5 sites, the management interface lacks a delete button. On 3 sites, deleting a passkey requires authentication with the *same* passkey, making deletion impossible if it is lost. Finally, on another 3 sites, users cannot delete individual passkeys but must remove all at once.

Equally problematic, users should be able to register multiple passkeys (e.g., a backup authenticator) to maintain access if their primary one is lost. Yet, on 25 domains, only a single passkey can be used.

Moreover, deleting a passkey in a website's account settings does not automatically remove the corresponding credential stored on the authenticator. To address this mismatch, browsers offer the `signalAllAcceptedCredentials()` [79] and `signalUnknownCredential()` [80] APIs. They allow websites to notify the browser of all accepted credentials or explicitly mark a credential as invalid. The browser then forwards this information to the authenticator to remove outdated credentials and stay synchronized. In practice, however, we found only 7 websites implementing these APIs, leaving many "dead passkeys" on authenticators.

Passkey Authentication Modes. Websites can start passkey authentications in three ways:

(1) *Discoverable Mode*: Here, the user clicks on a "Log in with Passkey" button. The authenticator then shows all passkeys that are valid for that website. If the user has multiple accounts, several passkeys may appear from which the user must choose one. In this case, the account is *discovered* on the authenticator.



(a) Modes for Passwordless and 2FA Passkey Authentications.

(b) Requested vs. Supported Algs. for Passkey Registrations.

Figure 6: Modes and Algorithms.

(2) *Non-Discoverable Mode*: Here, the user *first* enters a username on the website. The website checks its database to see if passkeys are registered for that account. If they exist, the website passes the allowed credential to the WebAuthn API, so the authenticator already knows which passkey to use. In this case, the account is *not discovered* but predefined on the website. This mode can also be used for two-factor authentication, where the user submits both a username and a password before the passkey is requested as the second factor.

(3) *Conditional Mode*: This works like the discoverable mode, but instead of clicking a button, the WebAuthn API is called automatically when the page loads. This improves usability, as a login with a passkey requires only one click.

Figure 6a shows that most websites use non-discoverable mode, although many only use it for two-factor authentication. For passwordless authentication, discoverable mode is more common. Interestingly, 8 websites use non-discoverable mode for 2FA while also supporting discoverable mode for passwordless login.

Passkey Algorithms. The WebAuthn standard [89] does not limit the signature and key algorithms that can be used. Instead, all algorithms are defined in the IANA COSE algorithms registry [16]. Figure 6b shows that websites usually support more algorithms than they request. Although deprecated and not recommended [16], `ES256` (universally supported) and `RS256` are the most common. An authenticator limited to `ES256` is therefore compatible with all sites. In contrast, the RSASSA-PSS algorithms, although recommended [16], are the least requested and supported. We also identified 26 algorithm identifiers that are deprecated, unassigned, or incompatible with passkeys. For instance, the deprecated RSASSA-PKCS1-v1_5 with SHA-1 was still requested 5 times. Also, 8 sites requested symmetric HMAC algorithms, which cannot work with the asymmetric design of passkeys. On 22 sites, all algorithms were supported except `ES384`, a unique pattern not observed for any other algorithm and likely caused by a library that omits this algorithm.

Passkey Scopes. The scope of a passkey defines the origins that can use it. If a passkey is created for `www.rp.com`, then only this domain and its subdomains can access it. In addition, the passkey can be scoped to a parent domain up to the registrable domain, such as `rp.com`. This broadens access to `rp.com` and all its subdomains. Best practice is to keep the scope as narrow as possible and limited to the domains that actually need access to the passkey. For instance, if passkeys are only required on `login.rp.com`, scoping them to `rp.com` creates unnecessary risks. If an attacker compromises any subdomain, the passkey can be misused to take over accounts.

We observed that 80 sites (38%) explicitly scope their passkeys to the top-level domain, leaving them exposed to subdomain takeover attacks [86], a well-known threat on the modern web [84]. Common cases include passkeys created on subdomains like `www` (22), `account[s]` (16), `app` (13), or `sso` (4), which are then upscoped to the top-level domain.

User Verification. To prevent passkeys from being stolen and misused, they enable user verification through biometrics (e.g., facial recognition) or a PIN. This ensures two-factor authentication: the user must both possess the device ("something you have") and verify their identity ("something you are / know"). When verification succeeds, the authenticator sets the *user verified bit* in its flags to true. We found only 82 sites that required authenticators to perform user verification. 18 sites explicitly disabled it, but since they used passkeys only as a second factor, accounts remain protected if the authenticator is lost. More concerning, another 17 sites explicitly disabled user verification while relying on passkeys for passwordless authentication. In these cases, if the authenticator is lost or stolen, an attacker could sign in to the victim's account.

Authenticator Selection. RPs can request the use of either a software authenticator (e.g., a phone) or a hardware authenticator (e.g., a security key). A software authenticator often runs on the same device where the user is signing in and offers an enhanced experience. A hardware authenticator is always external and therefore provides hardware-backed two-factor authentication. Only 22 sites (11%) explicitly requested a hardware authenticator and 5 sites actually enforce it. These websites cryptographically verify the use of a genuine hardware authenticator through attestation [89, §6.5].

User Identifier. Because authenticators can reveal user^{ids} without user verification, the WebAuthn standard requires that RPs must not store personal information (such as emails or usernames) in the user^{id} [89, §14.6.1]. However, 8 sites still embed email addresses or usernames directly in the user^{id}.

Key Attestation. By default, registering a $\text{cred}^{\text{pubkey}}$ at a website does not require proving possession of the corresponding $\text{cred}^{\text{privkey}}$. Key attestation can provide this proof. During registration, the authenticator signs the $\text{cred}^{\text{pubkey}}$ with the $\text{cred}^{\text{privkey}}$. This allows RPs to confirm that the registering party truly controls the private key. Such verification prevents an attacker from adding a victim's $\text{cred}^{\text{pubkey}}$ to their account and using it for session fixation attacks (see §6). However, key attestation currently has no practical relevance on the

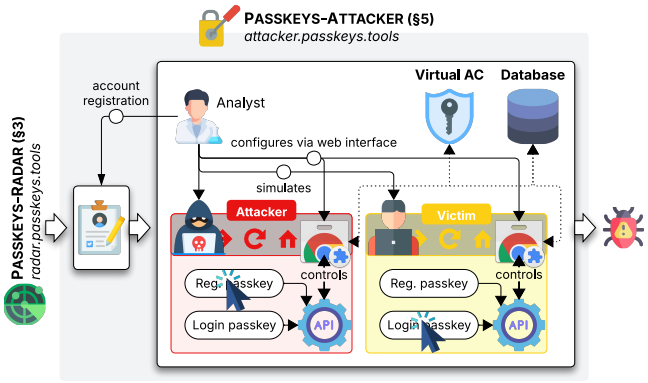


Figure 7: PASSKEYS-ATTACKER enables semi-automatic security analyses of passkey implementations on websites.

						Basic Attacks				Extended Attacks			
Tools	Authors	Open Source	Decoding	Encoding	Passive Testing	Active Testing	# Passive Tests	# Active Tests	Replay	Key Swap	User Swap	Session Swap
Passkeys Playground	r-n-o	✓	○	○	○	○	–	–	○	○	○	○
WebAuthn.io	Duo Labs	✓	◐	○	○	○	–	–	○	○	○	○
DebAuthn	[29]	✓	◐	○	○	○	–	–	○	○	○	○
WebAuthn CBOR Decoder	@srikanthramu	✓	◐	○	○	○	–	–	○	○	○	○
SimpleWebAuthn	@MasterKale	✓	●	○	○	○	–	–	○	○	○	○
fido2viewer	@sbweeden	✓	●	○	○	○	–	–	○	○	○	○
WebAuthn Playground	@opotonniee	✓	●	○	○	○	–	–	○	○	○	○
Passkeys Playground	Auth0	✗	●	○	○	○	–	–	○	○	○	○
WebAuthn Playground	Thinktecture Labs	✓	●	○	○	○	–	–	○	○	○	○
WebAuthn Viewer	@inabajunmr	✓	●	○	○	○	–	–	○	○	○	○
Passkeys Playground	OwnID	✓	●	○	○	○	–	–	○	○	○	○
Passkeys Debugger	Corbado	✗	●	○	○	○	–	–	○	○	○	○
WebAuthn.me	Auth0	✓	●	○	○	○	–	–	○	○	○	○
Passkeys Playground	Passwordless.ID	✓	●	○	○	○	–	–	○	○	○	○
Passkey Scanner	@alexcowperthwaite	✓	●	○	●	○	5	0	○	○	○	○
Passkey Raider	@siamthanathack	✓	●	●	●	●	0	0	○	○	○	○
WebDevAuthn	[45]	✓	●	◐	●	●	5	5	○	○	◐	○
PASSKEYS-ATTACKER	This Paper	✓	●	●	●	●	9	15	●	●	●	●

Table 1: Requirements for Passkey Security Testing. Only the PASSKEYS-ATTACKER meets all requirements to comprehensively test RPs against the specification [89].

web. We observed that only 66 sites request it and 85 sites explicitly disable it, yet none reject registrations without it.

5 PASSKEYS-ATTACKER: Semi-Automatic Security Analysis of Passkey Implementations

The PASSKEYS-ATTACKER forms the answer to RQ3. In contrast to prior work, it enables a semi-automated analysis pipeline for evaluating the state-of-the-art security of passkey implementations, as shown in Figure 7.

5.1 Requirements for Passkey Security Testing

The WebAuthn specification [89, §7] lists detailed security considerations. Based on these, we derived requirements that are necessary for security testing of passkey implementations. We summarize them in Table 1. We then identified related tools that could assist in implementing them. We searched online with keywords such as "webauthn/passkey/fido playground/debugger/tool" and inspected the first five pages. Table 1 compares our tool with existing ones. In summary, most tools primarily focus on decoding passkey messages, which is essential for protocol analysis but insufficient for active testing. Existing tools cannot handle advanced attacks like replay or session swap, which need awareness for user sessions or multiple profiles. For this reason, we created PASSKEYS-ATTACKER. It is the first tool that enables complete security testing of RPs in line with the WebAuthn standard [89]. In the following, we elucidate the identified requirements in depth.

Decoding WebAuthn Messages. Most existing tools are designed for developer support rather than security testing and research. They are usually demo websites that trigger the WebAuthn API. The browser and authenticator then process the messages, and the website displays the decoded results.

Encoding WebAuthn Messages. To manipulate WebAuthn messages, tools must support both decoding and re-encoding. Only PASSKEY RAIDER and WEBDEVAUTHN provide this. However, WEBDEVAUTHN restricts modifications to parameters that are hardcoded in its test cases.

Basic Attacks. Passive testing requires intercepting all WebAuthn messages executed by the target website and decoding them. Two approaches exist:

(1) *Intercepting HTTP Traffic*: PASSKEY SCANNER and PASSKEY RAIDER are Burp Suite extensions that scan HTTP traffic for WebAuthn messages. This approach is limited since there is no generic way to detect such messages in HTTP. For example, PASSKEY RAIDER requires custom regular expressions for each website, making large-scale studies impractical.

(2) *Intercepting WebAuthn API Calls*: WEBDEVAUTHN and PASSKEYS-ATTACKER hook directly into the browser API, providing a generic and website-independent way of intercepting messages. Both tools emulate the browser and authenticator, giving full control over all messages.

Active testing requires an additional capability: modifying intercepted and decoded messages (e.g., altering a signature or challenge) and re-encoding them. Only PASSKEY RAIDER and WEBDEVAUTHN can perform active attacks, with WEBDEVAUTHN implementing 7 selected tests. To the best of our knowledge, our tool – PASSKEYS-ATTACKER – is the first one to support all passive (see §4.2) and active (see §6.2) tests from the specification, doubling the coverage of prior tools.

Extended Attacks. PASSKEYS-ATTACKER can additionally automate advanced attacks:

(1) *Replay Attacks*: We track all WebAuthn messages and manipulations, enabling automated replay by substituting parameters with previously used values.

(2) *Key Swapping Attacks*: Our credential management system stores cred^{ids} and keys separately, enabling automated testing of attacks where an attacker registers their $\text{cred}^{\text{pubkey}}$ under a victim's cred^{id}.

(3) *User Swapping Attacks*: Our tool extracts user information from messages and manages it automatically. This

message tracing allows for automated user swapping attacks where an attacker registers a key under a victim's account. WEBDEVAUTHN supports these types of attacks only with manual extraction and copy-paste of identifiers.

(4) *Session Swapping Attacks*: We simulate two independent browsers, one for the victim and one for the attacker, with the tool tracking context awareness. This enables automated replacement of parameters between sessions.

5.2 Semi-Automated Analysis Approach

Our Desired Goal: Full Automation. To examine whether complete automation of our passkey analyses is feasible, we randomly selected five passkey-enabled websites. We built a prototype tool using Playwright [34] and the agentic LLM-based Browser Use framework [14]. The tool attempted to execute passkey registrations, authentications, and deletions without any manual steps. In practice, it succeeded on only one of the five sites, for the reasons discussed below.

Automation Hurdles. Our analyses require repeated execution of passkey registrations, authentications, and deletions. These are custom and highly sensitive operations, and websites protect them with strong measures that make automation hard if not impossible. Common obstacles include a combination of (1) customized UIs, (2) confirmation prompts, (3) password re-entry, (4) 2FA checks via email, SMS, OTP, backup codes, or security questions, (5) CAPTCHAs, and (6) rate limiting. Another challenge was the absence of common success indicators, as error messages were often unclear.

Our Result: Semi-Automated Approach. Thus, we adopted a semi-automated methodology, which is in line with prior work on post-authentication web security [81]. An important consideration in our design was ethical responsibility. In particular, we did not use automatic CAPTCHA solvers, since this would bypass protections designed to prevent automation. Our manual effort is limited to: (1) registering two test accounts per site, (2) adding and deleting passkeys, and (3) starting the passkey authentication. Once the WebAuthn API is invoked, the PASSKEYS-ATTACKER takes over and runs the entire security analysis automatically. Developers who already maintain automated tests for account management could integrate our tool to achieve fully automated evaluation of their passkey implementations.

Manual Account Registration. We created two Gmail addresses, one for the attacker and one for the victim, and used them to register accounts on all passkey-enabled sites. We used realistic fictional data for usernames and personal details, along with two valid phone numbers for verification codes. Non-English sites were translated with Chrome's built-in translation feature. Whenever possible, we registered via passwords, resorting to SSO only when necessary.

5.3 PASSKEYS-ATTACKER Architecture

The PASSKEYS-ATTACKER consists of three components as depicted in Figure 7.

(1) **Browser Extension**: The extension hooks into all WebAuthn APIs and intercepts their calls. Instead of using the browser's built-in WebAuthn handling, it opens a popup that loads the virtual client and authenticator, forwarding the full request with all parameters. It also adds contextual metadata, such as the origin and current operation (`create` or `get`), which is required for correct simulation. After processing, the extension resolves the original API call with the modified result. It can run in "attacker" or "victim" mode to ensure the virtual client and authenticator act in the right context.

(2) **Virtual Client and Authenticator**: This core component decodes, inspects, modifies (applies attacks), and re-encodes WebAuthn messages. It manages users and keys in a context-aware way. For example, it restricts access to attacker-owned keys when in attacker mode.

(3) **Database**: The virtual client and authenticator can run either standalone or with a shared database. In *standalone mode*, all data (users, keys, history) is stored in the browser's `localStorage`, suitable for testing with a single browser context. In *shared mode*, the database enables testing across multiple browser contexts (i.e., victim and attacker). This allows fully automated cross-context scenarios, such as replaying an attacker challenge in the victim's session.

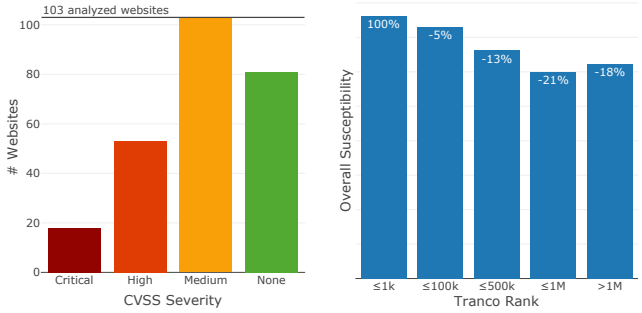
6 Passkey Security: From Standard Violations to RP-Specific Key Management Failures

In this section, we answer RQ4 with the first comprehensive analysis of how secure passkeys are implemented on the web.

6.1 Methodology

Attack Scope. We focus on threats known to the standardization community that directly affect RPs. As a basis, we carefully studied the WebAuthn standard [89], which outlines security and privacy considerations for RPs in §13.4 and §14.6. It also defines mandatory validation checks for registrations and authentications in §7.1 and §7.2. From this, we identified 15 Attack Types (ATs) that RPs must defend against. We then created 28 Detection Methods (DMs) to examine whether RPs implement these protections securely. Table 2 provides an overview of our attack catalog.

Impact Assessment. To evaluate the impact of missing validations, we researched all known CVEs related to WebAuthn and passkeys. For each AT, we checked for matching CVEs and included them in Table 2. From these, we extracted the CVSS scores and CWE classes to decide upon severity and weakness. If no related CVEs existed, we reported the weaknesses as new findings and estimated their severity by similar vulnerabilities or attack patterns from other domains.



(a) Number of vulnerable websites by CVSS severity (critical, high, medium, none). *Lower means less vulnerable websites.*

(b) Susceptibility of websites by their Tranco rank. Lower ranked sites are overall less susceptible. *Lower means more secure.*

Figure 8: Vulnerabilities by Tranco Rank and Attack Types.

Attacker Model. We assume that the victim's browser and authenticator are secure and follow the specification. The attacker controls their browser and authenticator via the PASSKEYS-ATTACKER, allowing arbitrary manipulation of WebAuthn messages. We further assume the attacker can lure victims to a malicious site and trick them into registering or using passkeys ("web attacker"). If an attack has additional requirements, we outline them in Table 2.

Evaluation Scope. For our security analysis, we had to manually register a fresh passkey for each DM. On 105 of the 208 websites, however, this was difficult or infeasible for several reasons. As described in §4, websites integrate passkeys in different ways. Some deploy them only as a second factor, which is incompatible with our attacker model. Others restrict accounts to a single passkey without offering a removal mechanism. In addition, some sites enforce further hurdles, such as email or SMS verification, CAPTCHAs, 2FA, or strict rate limiting. Consequently, we were able to conduct a full security evaluation on 103 websites.

6.2 Vulnerabilities

Table 2 summarizes the results of our semi-automatic security evaluation. We found that all of the analyzed websites (103) are vulnerable to at least one AT. This highlights a significant gap between the specification and real-world practice, likely caused by the complexity of the WebAuthn standard.

Attack Severity. Figure 8a shows that most missing validations fall into attack types with medium or negligible impact. However, 53 sites are vulnerable to high-severity threats and 18 sites to critical-severity threats, which is concerning.

Tranco Ranking. We initially expected insufficient validations to be more common on less popular sites, given the complexity of the standard. To test this, we compared the total number of validation issues across all analyzed websites. Figure 8b reveals a different picture: lower-ranked sites and even unranked ones are less susceptible than higher-ranked sites in the Tranco top 1k. This observation is an indication that larger platforms face greater challenges in fully implementing all validations. In contrast, smaller sites may benefit from relying on secure third-party libraries.

Registration vs. Authentication. Most attack types affect both the registration and authentication phase, and RPs must apply the same protections in both phases. However, we observed that many websites fail a validation in one phase while securing the other. For example, 9 sites do not validate the session binding of the challenge during registration, and 14 fail to do so during authentication. This may result from developers implementing checks separately or from libraries lacking a unified configuration for both phases.

In the following, we present selected Attack Types (ATs) in more detail and illustrate them with real-world examples.

6.2.1 CWE-347: Improper Verification of Cryptographic Signature

AT: Signature (Critical). The signature verification is the most critical step in the passkey authentication flow. Still, we found 5 websites that completely skip this check, which is a critical security issue. We confirmed these vulnerabilities and were able to sign in to the victim's account.

AT: Context (Medium). A less severe but still relevant issue for 12 sites arises from improper verification of the signing context. The idea of this attack is to use a signed registration challenge for user authentication. This context switch is known as signature confusion.

6.2.2 CWE-639: Authorization Bypass Through User-Controlled Key

AT: Credential Overwrite (Critical). This novel AT targets how the RP manages passkeys in its database. Every RP hosts several users, each distinguished by a user^{id}. Users may possess various credentials, each labeled by a cred^{id}. The relation is one-to-many: a user may own many credentials, but each credential must belong to exactly one user. The attacker creates a credential that seems to belong to both the victim and the attacker simultaneously, challenging this assumption. This ambiguity causes confusion during sign-in.

Requirements. This AT requires knowing the victim's cred^{id}. It is public since RPs reveal them without authentication (see §6.2.7). The victim's $\text{cred}^{\text{pubkey}}$ is harder to obtain, as it is stored only on the authenticator and the RP's database. Still, the specification assumes that security must hold even if the database leaks, so we treat this as a plausible threat.

DM: Swap cred^{id}. The attacker registers a new passkey by combining their own $\text{cred}^{\text{pubkey}}$ with the victim's cred^{id}. A secure implementation must detect the duplicate cred^{id} and reject it. If misconfigured, three outcomes are possible:

(1) *Database Update*: The RP replaces the victim's $\text{cred}^{\text{pubkey}}$ with the attacker's $\text{cred}^{\text{pubkey}}$. This locks the victim

Attack Type (AT)	CVSS	CVE	CWE	Requirement	Validation [89]	RW	Detection Method (DM)	# Vuln. RPs Reg.	Auth.	∪
Signature	9.8	–	Improper Verification of Cryptographic Signature (CWE-347)	–	Use $cred^{pubkey}$ to verify signature over authData and clientData (§7.2.21)		**Bit Flip**: Flip the last bit of the signature	–	5	5
Credential Overwrite	9.1	[23]	Authorization Bypass Through User-Controlled Key (CWE-639)	Leaked $cred^{id}$ and/or $cred^{pubkey}$	$cred^{id}$ is not yet registered for any user (§7.1.26)		**Swap $cred^{id}$+$cred^{pubkey}$**: Register passkey with victim's $cred^{id}$ and $cred^{pubkey}$ **Swap $cred^{id}$**: Register passkey with victim's $cred^{id}$ and attacker's $cred^{pubkey}$ **Swap $cred^{pubkey}$**: Register passkey with attacker's $cred^{id}$ and victim's $cred^{pubkey}$	13 12 2	– – –	14
Related Origins	8.8	–	Externally Controlled Reference to a Resource in Another Sphere (CWE-610)	–	Include only active and trusted related origins (§5.11)		**Dangling Domains**: Check if related origins contain registrable domain	–	1	1
Challenge	8.2	[27]	Session Fixation (CWE-384)	Leaked or injected challenge (e.g. via XSS or CSRF)	clientData.challenge is options.challenge (§7.1.8, §7.2.11)	[44]	**Bit Flip**: Flip last bit of challenge **Reuse**: Use challenge that has already been used **Session Binding**: Use valid challenge from another session	1 8 9	1 5 14	22
Origin	8.1	[26]	Origin Validation Error (CWE-346)	Subdomain Takeover [84]	clientData.origin is trusted (§7.1.9, §7.2.12)	[44] [44]	**Cross-Site**: Set clientData.origin to a different site **Subdomain**: Set clientData.origin to a subdomain **Parent Domain**: Set clientData.origin to a parent domain	7 23 31	5 23 23	40
RP ID	8.1	[26]	Origin Validation Error (CWE-346)	Subdomain Takeover [84]	authData.hash(rp^{id}) is hash(options.rp^{id}) (§7.1.14, §7.2.15)	[44]	**Cross-Site**: Set authData.hash(rp^{id}) to the hash of a different site **Subdomain**: Set authData.hash(rp^{id}) to the hash of a subdomain **Parent Domain**: Set authData.hash(rp^{id}) to the hash of a parent domain	8 7 5	4 4 3	9
User Verified	6.8	[21]	Improper Authentication (CWE-287)	Physical access to authenticator	authData.flags.UV is true if options.userVerification is required (§7.1.16, §7.2.17)	[44]	**Unset**: Set authData.flags.UV to false	8	6	10
Context	6.4	–	Improper Verification of Cryptographic Signature (CWE-347)	Access to valid attestation signature (e.g. via XSS)	clientData.type is "webauthn.create" / "webauthn.get" (§7.1.7, §7.2.10)		**Nonsense**: Set context to "abc.def" **Swap**: Swap context create ↔ get	5 11	4 8	12
User Present	5.9	[22]	Improper Authentication (CWE-287)	Malware	authData.flags.UP is true (§7.1.15, §7.2.16)		**Unset**: Set authData.flags.UP to false	21	16	27
Allow Credentials	5.3	[25]	Observable Response Discrepancy (CWE-204)	–	Return imaginary $cred^{ids}$ in requestOptions.allowCredentials for non-existing accounts (§14.6.2, §14.6.3)		**Random**: Identify as random user. requestOptions.allowCredentials empty?	–	68	68
Framing	5.1	–	Improper Restriction of Rendered UI Layers or Frames (CWE-1021)	No generic clickjacking protections in place [17]	If clientData.crossOrigin is true, check if framing by other origins is allowed and if clientData.topOrigin is trusted (§7.1.10+11, §7.2.13+14)		**Cross-Origin**: Set clientData.crossOrigin to true **Cross-Site**: Set clientData.topOrigin to a different site **Subdomain**: Set clientData.topOrigin to a subdomain **Parent Domain**: Set clientData.topOrigin to a parent domain	99 93 94 84	87 92 92 84	102
Signature Counter	4.8	[24]	Improper Authentication (CWE-287)	Physical access to cloned authenticator	authData.sig^{cnt} is greater than previously stored sig^{cnt} (§7.2.22)	[44]	**Reduce**: Set authData.sig^{cnt} to 2. Login. Set authData.sig^{cnt} to 1. Login.	–	57	57
Backup State	0.0	–	–	Authenticator is lost	authData.flags.BS is false if authData.flags.BE is false (§7.1.17, §7.2.18)		**Mutual Exclusion**: Set authData.flags.BE to false and authData.flags.BS to true	58	57	65
Backup Eligible	0.0	–	–	Authenticator is lost	authData.flags.BE is equal in registration and authentication (§7.2.19)		**Swap On**: Register authData.flags.BE=false. Login authData.flags.BE=on. **Swap Off**: Register authData.flags.BE=true. Login authData.flags.BE=false.	– –	55 55	57
Length of $cred^{id}$	0.0	–	–	–	$cred^{id}$.length ≤ 1023 bytes (§7.1.25)		**Length 1024**: Set $cred^{id}$ to 1024 random bytes	48	–	48
							Vulnerable websites with CVSS severity: Critical (18), High (53), Medium (103), None (81), All combined: 103 (of 103 analyzed RPs)			

Table 2: **Evaluation Results and Test Catalog**. We analyzed 103 RPs with PASSKEYS-ATTACKER and found at least one security issue in all of them. Prior work covered 5 ATs. We extend these with new DMs and introduce 10 additional ATs.

out, since the victim does not have access to the matching $cred^{privkey}$ owned by the attacker. We found 5 vulnerable sites.

(2) *Database Append*: The RP stores two $cred^{pubkeys}$ under the same $cred^{id}$. During authentication, querying this $cred^{id}$ returns multiple keys, which causes internal errors and blocks logins. We observed this on 3 sites. If the attacker deletes the duplicate from their account, the victim can sign in again.

(3) *Database Delete*: The RP deletes the victim's credential when a duplicate $cred^{id}$ is added. On 3 sites, attackers can delete passkeys of all users and weaken their account security.

DM: Swap $cred^{id}$+$cred^{pubkey}$. Here, the attacker registers the victim's $cred^{pubkey}$ with the victim's $cred^{id}$ in the attacker's account. If the RP uses the $cred^{id}$ without the $user^{id}$ for credential lookup during authentication, session swapping becomes possible. When the victim later signs in with the $cred^{privkey}$, the signature is valid and the victim is logged in to the attacker's account. This can cause sensitive data disclosure, for example when the victim uploads files. We successfully carried out this AT on 3 sites.

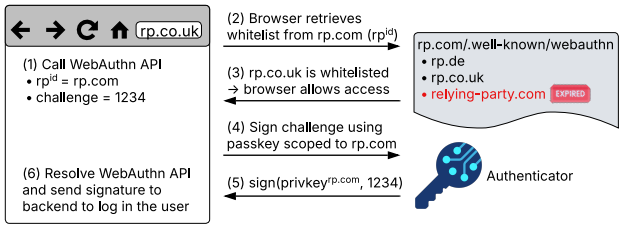


Figure 9: CWE-610: Externally Controlled Reference to a Resource in Another Sphere.

6.2.3 CWE-610: Externally Controlled Reference to a Resource in Another Sphere

AT: Related Origins (High). Some organizations need to use the same passkey across multiple country code Top-Level Domains (ccTLDs), such as `rp.com` and `rp.co.uk`. By default, this is blocked by the browser. The ROR feature [89, §5.11] enables shared domains by allowing RPs to publish a list of related origins that may share the same passkey.

Consider a user who logs in to `rp.co.uk` (see Figure 9): (1) The RP calls the WebAuthn API with the rp^{id} set to `rp.com`. (2) Since `rp.co.uk` and `rp.com` are cross-site, the browser

fetches the whitelist from a well-known location at `rp.com`. (3) Because `rp.co.uk` appears in the whitelist, the browser allows access. (4) The browser requests a signature of the challenge using the passkey for the rp^{id}. (5) The authenticator signs the challenge and returns it. (6) The API call completes.

DM: Dangling Domains. Maintaining such whitelists is risky if they contain dangling domains that are no longer controlled by the RP or have expired. For example, `relying-party.com` could be an expired domain that can be re-registered (cf. Figure 9). If an attacker acquires it and serves a malicious site, they gain access to the victim's passkey on `rp.com` and can compromise the account.

We identified 2,177 related origins listed in whitelists across 177 domains. The largest whitelist contained 102 origins, making management especially difficult. We resolved all domains using Google's DNS API [51] and checked for dangling entries. Using the Namecheap API [8], we found one allowlisted domain available for registration for 10€/year. If acquired, it would allow an attacker to take over accounts on 72 related origins.

6.2.4 CWE-384: Session Fixation

AT: Challenge (High). The challenge is the only randomized input in the signature generation and changes with every authentication flow. It binds the signed assertion to both, the current flow and the user's session. This binding protects against replay attacks if a signed assertion is leaked (e.g., via XSS). It additionally blocks injection attempts (e.g., via CSRF) where an attacker tries to log a victim into the attacker's account. Despite its importance, we found 22 sites that do not properly validate the challenge, leaving them vulnerable if attackers can steal or inject signed assertions.

6.2.5 CWE-346: Origin Validation Error

AT: Origin & rp^{id} (High). Passwords are prone to phishing; however, passkeys are phishing-resistant because browsers block malicious sites from accessing passkeys belonging to different sites. Unfortunately, browsers allow subdomains to use passkeys scoped to their parent domain. This makes related-domain attackers relevant, such as those who gain control of a subdomain or sibling domain through subdomain takeover. Prior work [84] shows that such cases are common.

To mitigate this, the WebAuthn standard requires browsers to include the full origin of the passkey operation, so that the RP can decide whether to trust it and reject requests from untrusted sub- or sibling domains. Yet, we found 40 sites that do not validate the origin, allowing attackers on malicious sub- or sibling domains to use the RP's passkey, making phishing attacks possible again.

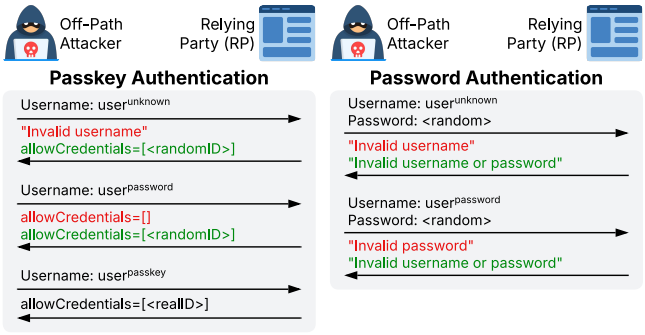


Figure 10: CWE-204: Observable Response Discrepancy. Red errors are distinguishable, enabling account enumeration. Green errors are indistinguishable, preventing such leaks.

6.2.6 CWE-287: Improper Authentication

AT: User Verified (Medium). In §4.2, we showed that 82 sites explicitly requested user verification, for example through biometrics. However, 10 sites did not check whether verification was actually performed. This creates a false sense of security: the site assumes verification took place, even if it did not. An attacker who steals a victim's security key could then log in without biometrics or a PIN. While this usually requires physical access to the authenticator, there are more scenarios. For instance, if a victim carries an NFC-enabled security key in a pocket, an attacker could hold a phone nearby and immediately sign in to the victim's account [83].

AT: User Presence (Medium). Malware on a user's system can misuse the authenticator as a signing oracle and generate valid assertions for other websites. To prevent this, the specification requires a user presence check for every signing operation, such as touching the security key. If this check is missing, an attacker could obtain signatures unnoticed and use them to compromise the victim's account.

AT: Signature Counter (Medium). Each authenticator maintains a signature counter that increases with each signing operation. When an attacker logs into the victim's account with a cloned authenticator [82], the counter increases. Later, when the victim signs in with the original device, the counter is lower. The RP can detect this mismatch and warn the victim about the inconsistency. We found that over half of the websites ignore the counter, while the rest block the logins when counters are out of sync.

6.2.7 CWE-204: Observable Response Discrepancy

AT: Allow Credentials (Medium). Passkey logins are prone to account enumeration, where attackers learn whether an account exists. In non-discoverable mode, the attacker submits a username on the RP's login page. If the account exists and has passkeys, the RP returns the user's $cred^{ids}$ in the `allowCredentials` parameter of the requestOptions. Although $cred^{ids}$ are random bytes, their presence reveals that (1) the

victim has an account, and (2) the victim uses passkeys on this site. The WebAuthn standard addresses this by requiring RPs to return random placeholder cred^{ids} for accounts without passkeys or for non-existent accounts [89, §14.6.2].

DM: Random. Detecting account enumeration requires three user accounts: (1) $\text{user}^{\text{unknown}}$: a random, non-existent account, (2) $\text{user}^{\text{password}}$: an existing account with password authentication, and (3) $\text{user}^{\text{passkey}}$: an existing account with passkey authentication. The detection process works as follows (see left side of Figure 10): (1) Submitting $\text{user}^{\text{unknown}}$ should return a random placeholder in `allowCredentials`. (2) Submitting $\text{user}^{\text{password}}$ should again return a random placeholder. (3) Submitting $\text{user}^{\text{passkey}}$ must return the real cred^{id}. Any deviation (e.g., `allowCredentials` being empty) exposes an observable difference. We also tested password authentication [97] (see right side of Figure 10), by submitting $\text{user}^{\text{unknown}}$ and $\text{user}^{\text{password}}$ with wrong passwords and comparing the responses.

We found 68 websites vulnerable to account enumeration in their passkey authentication. Fewer sites (43) were vulnerable in their password authentication, often returning uniform errors such as "Invalid username *or* password". More concerning, 25 sites that correctly protected their password authentication failed to do so for passkeys. In these cases, adopting passkeys reduced security, which is alarming.

6.2.8 CWE-1021: Improper Restriction of Rendered UI Layers or Frames

AT: Framing (Medium). During each WebAuthn operation, the browser reports to the RP whether it is displayed in a frame and, if so, on which origin. The RP must check this to prevent clickjacking attacks [18]. Such attacks use hidden or overlapping layers to mislead a user into interacting with a different page than intended. We found that all but one website failed to validate the built-in clickjacking protections of WebAuthn. Without additional safeguards, attackers could trick users into registering or authenticating passkeys on unintended sites.

7 Limitations

Overestimation of Adoption Rate. Our automated detection of passkey-enabled websites inevitably produces false positives, which leads to an overestimation of the adoption.

(1) *Missed Duplicates*: Although we merge websites to reduce duplicates, complete deduplication is not possible. For instance, `minecraft.com` and `microsoft.com` rely on the same authentication backend but are not covered in [30].

(2) *TOTP Misclassifications*: Some community directories incorrectly include TOTP-only sites. While majority voting across sources could mitigate this, it would also remove less frequently listed sites, which we deliberately retained.

(3) *U2F Misclassifications*: Our analysis focuses on WebAuthn and FIDO2. We excluded U2F-only deployments, but FIDO2-based 2FA deployments remain within scope.

(4) *Scope Drift*: Some community directories include not only websites but also third-party passkey providers or platform-level support (e.g., OS or browser features).

Limited Representativeness of the Dataset. Our 208-site dataset (§4.2) is biased toward publicly accessible, consumer-facing services. Certain sectors, such as online banking, could not be systematically evaluated due to the lack of suitable test accounts. Similarly, the 103-site security evaluation (§6) is biased toward websites that permit passkey registration without additional verification steps. Such sites may systematically exhibit weaker protections, as they apply fewer safeguards during passkey enrollment.

Different Authentication Backends. It is not possible to automatically determine whether multiple domains operated by the same provider (e.g., `gmail.com` and `youtube.com`) share a common authentication backend. Across our 872-site dataset, we manually identified only four cases in which merged domains later turned out to rely on distinct passkey backends (e.g., `linkedin.com` and `microsoft.com`).

8 Related Work

Adoption. Previous studies on passkey adoption were mainly focused on manual analyses. In 2021, Ulqinaku et al. [88] inspected the Alexa Top 100 and found FIDO2 support on 23 sites. A year later, Kepkowski et al. [53] scanned the Cisco Umbrella Top 1M and identified 684 sites indicating WebAuthn usage by detecting the `navigator.credentials.create` property in JavaScript resources. In 2023, Kuchhal et al. [56] analyzed the Tranco Top 1k and found 85 sites supporting WebAuthn while Gavazzi et al. [40] found 12 sites on the Tranco Top 5k. Later, Blessing et al. [13] studied the Tranco Top 200 and reported 28 sites offering WebAuthn. Our work goes beyond these efforts with automated continuous scanning, merging multiple detection methods, and performing archived data analysis.

Security of RPs. Only a few studies have examined the real-world security of RPs. Grammatopoulos et al. [45] introduced WebDevAuthn, a tool for capturing WebAuthn requests and responses for manual analysis and inspection. They used it to assess conformance and security of 16 RPs [44], but their tool and evaluation missed advanced attack types such as replay, key swapping, and session swapping. Kuchhal et al. [56] systematized threats under the assumption of a compromised client (e.g., malware) and studied FIDO2 configurations of 29 RPs. They found that many sites used weak configurations, leaving users vulnerable once their devices were compromised. In contrast, we conducted an active security evaluation of 103 RPs using the web attacker model. Yadav et al. [98] analyzed FIDO2's resilience against local threats such as malicious browser extensions, XSS, or physical access to authenticators. They identified seven attacks and

demonstrated them with a malicious browser extension on ten popular web servers using FIDO2.

Usability Studies. While FIDO has advanced passwordless authentication, adoption still faces many barriers. Challenges arise from usability issues [7, 39, 58, 65], difficulties for employees and companies [32, 33, 52, 59, 60] and the public sector [49], technical hurdles in implementation [5, 6], user misconceptions [57], and interpersonal threats [28].

Formal Analyses. The FIDO2 standards have been examined with formal methods to prove their security and privacy, identify potential weaknesses, and investigate post-quantum security of the underlying protocols [9, 10, 11, 12, 35, 46, 47, 48]. While these works concentrate on the standards, our study focuses on how WebAuthn is used on the Internet.

9 Concluding Remarks

This paper sheds first light on how passkeys are deployed on the web. With PASSKEYS-RADAR, we conducted the largest evaluation of real-world passkey implementations to date and identified 872 passkey-enabled websites. Historic data shows a steady rise in deployments over the past four years.

Passkeys mark a major step in securing website logins. They reduce the attack surface of passwords by embedding authentication directly into browsers and authenticators, where layered protections improve resilience against flaws in RPs.

Our study, however, reveals that adoption remains inconsistent. Many RPs apply the WebAuthn specification only partially. Almost all request deprecated algorithms, and key attestation is rarely used despite its role as a security feature.

To measure compliance, we developed the PASSKEYS-ATTACKER. It lets researchers inspect and manipulate messages at every step in the WebAuthn protocol. One limitation is that it cannot fully automate account and passkey registrations, which restricts scalability. Another limitation is coverage: we could not test all 872 websites, since account creation often failed and many services blocked repeated passkey registrations. As a result, we focused on a subset of 103 sites, tested manually by three researchers over three weeks.

Still, the PASSKEYS-ATTACKER is first to automate advanced scenarios such as session swapping and message replay. We will announce the tool to the passkey standardization community and make it easily accessible, so that developers, researchers, and penetration testers can quickly find it and use it for their analyses. Future work may also extend its use beyond websites to test passkey libraries [78] and third-party services that promise to streamline passkey adoption [75].

Our evaluation shows that vulnerabilities persist, especially at the interface between the browser's WebAuthn API and RPs. Although the specification mandates strict validations, none of the tested sites enforced them fully. About every second site contained critical or high-severity issues. Weaknesses are common in passkey management, where poor handling of key identifiers enables attacks such as unauthorized removal of passkeys, account blocking, or even takeover.

Passkeys do not remove all risks, but they make large-scale account takeovers far harder and shift many threats to compliance issues. They represent clear progress for web authentication, though careful attention to protocol compliance and management remains essential to avoid severe problems.

Acknowledgments

The authors would like to thank the reviewers for their valuable feedback. This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2092 CASA – 390781972. Louis Jannett was supported by the research project "North-Rhine Westphalian Experts in Research on Digitalization (NERD II)", sponsored by the state of North Rhine-Westphalia – NERD II 005-2201-0014.

Ethical Considerations

Our research focuses on testing the security of websites by manipulating the WebAuthn protocol flow and messages during our own passkey registrations and authentications. The stakeholders involved in this study are: (1) the RPs whose websites we tested, (2) their users, (3) our research team members, and (4) all users relying on passkeys for authentication.

RP Interactions. For the 208 analyzed RPs, we manually created two testing accounts per site, using only fictitious data without any personal or sensitive information. Our activities were limited to registering accounts, registering passkeys, authenticating with them, and manually navigating the websites. Since all interactions were performed manually, no excessive load was placed on the tested services. We did not employ automated CAPTCHA solvers or similar tools.

Security Testing. All security tests were conducted exclusively with our own two testing accounts, ensuring that no other users were affected. Identified vulnerabilities were not exploited beyond what was necessary to demonstrate proof of concept. In particular, we did not access, compromise, or interfere with any other accounts or passkeys.

Responsible Disclosure. We responsibly disclosed all critical and high-severity findings to 61 affected websites. To date, 12 websites have fixed the reported issues, including 6 that awarded a bug bounty. As of December 2025, we cannot confirm remediation for the remaining websites due to a lack of response (15), missing follow-up communication after initial acknowledgment (8), requests for website-specific proof-of-concept material that we cannot provide due to unmet attack requirements (16), or the issues being considered out of scope by the affected parties (6). For 3 websites, reporting was not possible due to platform restrictions (e.g., minimum reputation requirements on HackerOne). In one case, the reported

issue had already been resolved prior to our disclosure.

We will continue to support the vendors by retesting deployed patches and providing feedback to help strengthen the overall security of passkey authentication. By the time of publication, all vendors had at least 6 months to address the reported vulnerabilities.

Retrospective Ethical Considerations

Releasing our Tool. The release of PASSKEYS-ATTACKER is intended to assist researchers and developers in identifying and addressing vulnerabilities in passkey implementations. Thereby, we improve the overall passkey security and contribute to safer and more robust authentication systems.

We acknowledge the risk that malicious actors could use PASSKEYS-ATTACKER to identify vulnerabilities, potentially leading to harmful applications. However, PASSKEYS-ATTACKER is designed for use with two testing accounts that are owned by the analyst, and it only probes with proof of concepts that do not interfere with anything else than one of their own testing accounts. Indeed, it remains a risk that PASSKEYS-ATTACKER could be adapted to attack accounts that are not owned by the analyst and generate payloads that violate their intended ethical constraints. In the worst case, the malicious actor could gain access to the victim's account or overwrite their passkeys.

Overall, we see PASSKEYS-ATTACKER as being in line with well-established and publicly-accessible vulnerability scanning tools[2]. We clearly see the benefits of releasing our tool to outweigh the risks it may introduce. Through our responsible disclosure, we noticed that vendors and triagers actively used and showed particular appreciation for our tool. One passkey developer remarked: "It enables in-depth testing that was simply not possible before." The tool is further usable only in a semi-automated fashion, making security analyses for pentesters easier but still not allowing for automated vulnerability scanning. We strongly emphasize the need for responsible use of the tool, explicitly discourage any unethical applications, and provide clear guidelines to minimize the likelihood of misuse.

We will contact the FIDO Alliance to promote our tool and request its inclusion on the public lists of passkey tools[3]. Additionally, we will discuss the integration of PASSKEYS-ATTACKER into the official FIDO compliance tool [42] with the alliance. We strongly advocate for open science and will therefore release all tools and artifacts on GitHub to maximize availability, functionality, and reproducibility.

Research Benefits. We chose to conduct this research to identify and help mitigate flaws in passkey implementations that could already be exploited by adversaries. To minimize harm and enable timely remediation, we followed a coordinated vulnerability disclosure process: we first shared our findings with the relevant vendors to enable them to implement the necessary fixes and protect their users. Only after this remediation phase do we disclose our results to the wider public through this publication, benefiting all users who rely on passkeys for authentication. We believe that this process, and the resulting improvements to the ecosystem, justify both the study and its publication.

[2] https://owasp.org/www-community/Vulnerability_Scanning_Tools

[3] https://passkeys.dev/docs/tools-libraries/test-sites/

Open Science

Our project website is available at https://passkeys.tools. All tools, source code, and artifacts are publicly available via GitHub (https://github.com/RUB-NDS/state-of-passkeys-artifacts) and Zenodo (https://doi.org/10.5281/zenodo.17898769).

A continuously running instance of our PASSKEYS-RADAR is available at https://radar.passkeys.tools. It offers live statistics on passkey adoption and visualizes the performance of individual detection sources. Researchers can explore aggregated lists of passkey-enabled websites for any given date, with detailed coverage per domain and source. All merged lists are available for download as JSON files, and the full dataset can be accessed through our public API. Our ongoing scans will ensure long-term monitoring of passkeys.

We also provide a publicly accessible instance of the PASSKEYS-ATTACKER at https://attacker.passkeys.tools. Together with the browser extension (downloadable from the website) and our preconfigured public database, this allows anyone to replicate our testing setup without requiring local installation. The entire setup is containerized with Docker and Docker Compose, simplifying local deployments.

Our artifacts are organized as follows:

(1) `./radar`: Source code of the PASSKEYS-RADAR.
(2) `./detector`: Source code for scanning well-known files.
(3) `./tools`: Source code of the PASSKEYS-ATTACKER.
(4) `./data`: Artifacts for the PASSKEYS-RADAR, including the community directories, combined, and merged lists.
(5) `./notebooks`: Jupyter notebooks for analyzing evaluation data (`sheet.csv`) and generating paper figures.

References

[1] 1Password. *Passkeys.directory*. [Online; accessed 2025-08-06]. URL: https://passkeys.directory/.

[2] *2FA Directory*. [Online; accessed 2025-08-06]. URL: https://2fa.directory/de/.

[3] *A Well-Known URL for Relying Party Passkey Endpoints*. [Online; accessed 2025-08-06]. June 2025. URL: https://w3c.github.io/webappsec-passkey-endpoints/.

[4] *aID.no*. [Online; accessed 2025-08-18]. URL: https://www.aid.no/aid/.

[5] Aftab Alam et al. "Poster: Let History not Repeat Itself (this Time) – Tackling WebAuthn Developer Issues Early On". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (Nov. 2019). Conference Name: CCS '19: 2019 ACM SIGSAC Conference on Computer and Communications Security ISBN: 9781450367479 Place: London United Kingdom Publisher: ACM, pp. 2669–2671. DOI: 10.1145/3319535.3363283. URL: https://dl.acm.org/doi/10.1145/3319535.3363283 (visited on 03/19/2025).

[6] Fatima Alqubaisi et al. "Should We Rush to Implement Password-less Single Factor FIDO2 based Authentication?" In: *2020 12th Annual Undergraduate Research Conference on Applied Computing (URC)*. Apr. 2020, pp. 1–6. DOI: 10.1109/URC49805.2020.9099190. URL: https://ieeexplore.ieee.org/document/9099190 (visited on 03/16/2025).

[7] Youssef Amer et al. "Understandability of the Technology and Benefit May Not Be Enough to Nudge Users: An Exploratory Study in the Context of FIDO2 Adoption Behavior". In: *2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*. 2025, pp. 607–618. DOI: 10.1109/COMPSAC65507.2025.00083.

[8] *API Methods for Developers | Namecheap.com*. [Online; accessed 2025-08-12]. URL: https://www.namecheap.com/support/api/methods/.

[9] Manuel Barbosa et al. "Privacy and Security of FIDO2 Revisited". In: *Proceedings on Privacy Enhancing Technologies* (2025). ISSN: 2299-0984. URL: https://petsymposium.org/popets/2025/popets-2025-0100.php (visited on 08/26/2025).

[10] Manuel Barbosa et al. "Provable Security Analysis of FIDO2". In: *Advances in Cryptology – CRYPTO 2021*. Ed. by Tal Malkin et al. Cham: Springer International Publishing, 2021, pp. 125–156. ISBN: 978-3-030-84252-9. DOI: 10.1007/978-3-030-84252-9_5.

[11] Nina Bindel et al. "FIDO2, CTAP 2.1, and WebAuthn 2: Provable Security and Post-Quantum Instantiation". In: *2023 IEEE Symposium on Security and Privacy (SP)*. ISSN: 2375-1207. May 2023, pp. 1471–1490. DOI: 10.1109/SP46215.2023.10179454. URL: https://ieeexplore.ieee.org/document/10179454 (visited on 03/16/2025).

[12] Nina Bindel et al. "To Attest or Not to Attest, This is the Question – Provable Attestation in FIDO2". In: *Advances in Cryptology – ASIACRYPT 2023*. Ed. by Jian Guo et al. Singapore: Springer Nature, 2023, pp. 297–328. ISBN: 978-981-99-8736-8. DOI: 10.1007/978-981-99-8736-8_10.

[13] Jenny Blessing et al. "SoK: Web Authentication and Recovery in the Age of End-to-End Encryption". In: *Proceedings on Privacy Enhancing Technologies* 2025.3 (July 2025), pp. 560–589. ISSN: 2299-0984. DOI: 10.56553/popets-2025-0113. URL: https://petsymposium.org/popets/2025/popets-2025-0113.php (visited on 01/13/2026).

[14] *browser-use/browser-use: Make websites accessible for AI agents. Automate tasks online with ease.* [Online; accessed 2025-08-07]. URL: https://github.com/browser-use/browser-use.

[15] Marco Casagrande et al. *CTRAPS: CTAP Client Impersonation and API Confusion on FIDO2*. arXiv:2412.02349 [cs]. Dec. 2024. DOI: 10.48550/arXiv.2412.02349. URL: http://arxiv.org/abs/2412.02349 (visited on 03/16/2025).

[16] *CBOR Object Signing and Encryption (COSE)*. [Online; accessed 2025-08-18]. URL: https://www.iana.org/assignments/cose/cose.xhtml#algorithms.

[17] *Clickjacking*. [Online; accessed 2025-08-20]. May 2025. URL: https://developer.mozilla.org/en-US/docs/Web/Security/Attacks/Clickjacking#clickjacking_defenses.

[18] *Clickjacking | OWASP Foundation*. [Online; accessed 2025-08-26]. URL: https://owasp.org/www-community/attacks/Clickjacking.

[19] *Cross Site Request Forgery (CSRF) | OWASP Foundation*. [Online; accessed 2025-08-19]. URL: https://owasp.org/www-community/attacks/csrf.

[20] *CrUX on BigQuery | Chrome UX Report | Chrome for Developers*. [Online; accessed 2025-08-06]. URL: https://developer.chrome.com/docs/crux/bigquery.

[21] *CVE Record: CVE-2020-8236*. [Online; accessed 2025-08-22]. URL: https://www.cve.org/CVERecord?id=CVE-2020-8236.

[22] *CVE Record: CVE-2021-38299*. [Online; accessed 2025-08-22]. URL: https://www.cve.org/CVERecord?id=CVE-2021-38299.

[23] *CVE Record: CVE-2023-44039*. [Online; accessed 2025-08-22]. URL: https://www.cvedetails.com/cve/CVE-2023-44039/.

[24] *CVE Record: CVE-2023-45669*. [Online; accessed 2025-08-22]. URL: https://www.cve.org/CVERecord?id=CVE-2023-45669.

[25] *CVE Record: CVE-2024-39912*. [Online; accessed 2025-08-22]. URL: https://www.cve.org/CVERecord?id=CVE-2024-39912.

[26] *CVE Record: CVE-2025-24180*. [Online; accessed 2025-08-22]. URL: https://www.cve.org/CVERecord?id=CVE-2025-24180.

[27] *CVE Record: CVE-2025-53102*. [Online; accessed 2025-08-22]. URL: https://www.cve.org/CVERecord?id=CVE-2025-53102.

[28] Alaa Daffalla et al. "A Framework for Abusability Analysis: The Case of Passkeys in Interpersonal Threat Models". In: 34th USENIX Security Symposium. 2025.

[29] Martiño Rivera Dourado et al. "Implementing a Web Application for W3C WebAuthn Protocol Testing". In: *3rd XoveTIC Conference*. XoveTIC Conference. Basel Switzerland: MDPI, Aug. 18, 2020, p. 5. DOI: 10.3390/proceedings2020054005. URL: https://www.mdpi.com/2504-3900/54/1/5 (visited on 07/08/2025).

[30] *DuckDuckGo Tracker Radar Entity Map*. [Online; accessed 2025-08-06]. URL: https://raw.githubusercontent.com/duckduckgo/tracker-radar/refs/heads/main/build-data/generated/entity_map.json.

[31] Email: *Final: OpenID Connect Discovery 1.0 incorporating errata set 2*. [Online; accessed 2025-08-06]. URL: https://openid.net/specs/openid-connect-discovery-1_0.html.

[32] Florian M. Farke et al. "Exploring User Authentication with Windows Hello in a Small Business Environment". In: *Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022)*. Boston, MA: USENIX Association, Aug. 2022, pp. 523–540. ISBN: 978-1-939133-30-4. URL: https://www.usenix.org/conference/soups2022/presentation/farke.

[33] Florian M. Farke et al. "You still use the password after all – Exploring FIDO2 Security Keys in a Small Company". In: 2020, pp. 19–35. ISBN: 978-1-939133-16-8. URL: https://www.usenix.org/conference/soups2020/presentation/farke (visited on 03/16/2025).

[34] *Fast and reliable end-to-end testing for modern web apps | Playwright*. [Online; accessed 2025-08-07]. URL: https://playwright.dev/.

[35] Haonan Feng et al. "FIDO Gets Verified: A Formal Analysis of the Universal Authentication Framework Protocol". In: *IEEE Transactions on Dependable and Secure Computing* 20.5 (Sept. 2023). Conference Name: IEEE Transactions on Dependable and Secure Computing, pp. 4291–4310. ISSN: 1941-0018. DOI: 10.1109/TDSC.2022.3217259. URL: https://ieeexplore.ieee.org/document/9930658 (visited on 03/16/2025).

[36] *FIDO Alliance Overview | FIDO Alliance*. [Online; accessed 2025-08-21]. URL: https://fidoalliance.org/overview/.

[37] *FIDO Directory of Passkey Implementations | FIDO Alliance*. [Online; accessed 2025-08-06]. Sept. 2021. URL: https://fidoalliance.org/passkeys-directory/.

[38] E. Foudil. *RFC 9116: A File Format to Aid in Security Vulnerability Disclosure*. [Online; accessed 2025-08-06]. Apr. 2022. URL: https://www.rfc-editor.org/rfc/rfc9116.html.

[39] Ingunn Langtangen Furuberg et al. "From Password to Passwordless: Exploring User Experience Obstacles to the Adoption of FIDO2 Authentication". Accepted: 2023-10-03T17:22:13Z. MA thesis. NTNU, 2023. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/3093908 (visited on 03/17/2025).

[40] Anthony Gavazzi et al. "A Study of Multi-Factor and Risk-Based Authentication Availability". In: *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 2043–2060. ISBN: 978-1-939133-37-3. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/gavazzi.

[41] *Gitea Official Website*. [Online; accessed 2025-08-18]. URL: https://about.gitea.com/.

[42] Github. *Certification Test Tools Resources*. [Online; accessed 2026-01-13]. URL: https://github.com/fido-alliance/conformance-test-tools-resources.

[43] Github. *Passkeys Authenticator AAGUID Explorer*. [Online; accessed 2025-04-02]. URL: https://passkeydeveloper.github.io/passkey-authenticator-aaguids/explorer/.

[44] Athanasios Vasileios Grammatopoulos et al. "Blind software-assisted conformance and security assessment of FIDO2/WebAuthn implementations". In: *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications* 13.2 (June 2022), pp. 96–127. DOI: 10.22667/JOWUA.2022.06.30.096. URL: https://doi.org/10.22667/JOWUA.2022.06.30.096 (visited on 03/19/2025).

[45] Vasileios Athanasios Grammatopoulos et al. "A web tool for analyzing FIDO2/WebAuthn Requests and Responses". In: *Proceedings of the 16th International Conference on Availability, Reliability and Security*. ARES '21. New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 1–10. ISBN: 978-1-4503-9051-4. DOI: 10.1145/3465481.3469209. URL: https://dl.acm.org/doi/10.1145/3465481.3469209 (visited on 03/19/2025).

[46] Jingjing Guan et al. “A Formal Analysis of the FIDO2 Protocols”. In: *Computer Security – ESORICS 2022: 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26–30, 2022, Proceedings, Part III*. Copenhagen, Denmark: Springer-Verlag, 2022, pp. 3–21. ISBN: 978-3-031-17142-0. DOI: 10.1007/978-3-031-17143-7_1. URL: https://doi.org/10.1007/978-3-031-17143-7_1.

[47] Iness Ben Guirat et al. “Formal verification of the W3C web authentication protocol”. In: *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*. Raleigh North Carolina: ACM, Apr. 2018, pp. 1–10. ISBN: 978-1-4503-6455-3. DOI: 10.1145/3190619.3190640. URL: https://dl.acm.org/doi/10.1145/3190619.3190640 (visited on 04/12/2025).

[48] Lucjan Hanzlik et al. *Token meets Wallet: Formalizing Privacy and Revocation for FIDO2*. Publication info: Published elsewhere. Minor revision. IEEE S&P 2023. 2022. URL: https://eprint.iacr.org/2022/084 (visited on 03/16/2025).

[49] Jan-Ulrich Holtgrave et al. “A Qualitative Study of Adoption Barriers and Challenges for Passwordless Authentication in German Public Administrations”. In: (2025).

[50] Louis Jannett et al. “SoK: SSO-MONITOR - The Current State and Future Research Directions in Single Sign-on Security Measurements”. In: *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*. 2024, pp. 173–192. DOI: 10.1109/EuroSP60621.2024.00018.

[51] *JSON API for DNS over HTTPS (DoH) | Public DNS | Google for Developers*. [Online; accessed 2025-08-12]. URL: https://developers.google.com/speed/public-dns/docs/doh/json.

[52] Michal Kepkowski et al. “Challenges with Passwordless FIDO2 in an Enterprise Setting: A Usability Study”. In: *2023 IEEE Secure Development Conference (SecDev)*. Oct. 2023, pp. 37–48. DOI: 10.1109/SecDev56634.2023.00017. URL: https://ieeexplore.ieee.org/document/10305624 (visited on 03/16/2025).

[53] Michal Kepkowski et al. “How Not to Handle Keys: Timing Attacks on FIDO Authenticator Privacy”. In: *Proceedings on Privacy Enhancing Technologies* 2022.4 (Oct. 2022), pp. 705–726. ISSN: 2299-0984. DOI: 10.56553/popets-2022-0129. URL: https://petsymposium.org/popets/2022/popets-2022-0129.php (visited on 04/14/2025).

[54] Donghyun Kim et al. “HiPass: Hijacking CTAP in Passkey Authentication”. In: *IEEE Access* 13 (2025), pp. 92086–92101. DOI: 10.1109/ACCESS.2025.3570377.

[55] Donghyun Kim et al. “Session Replication Attack Through QR Code Sniffing in Passkey CTAP Registration”. In: *ICT Systems Security and Privacy Protection*. Ed. by Nikolaos Pitropakis et al. Cham: Springer Nature Switzerland, 2024, pp. 294–307. ISBN: 978-3-031-65175-5. DOI: 10.1007/978-3-031-65175-5_21.

[56] Dhruv Kuchhal et al. “Evaluating the Security Posture of Real-World FIDO2 Deployments”. In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’23. New York, NY, USA: Association for Computing Machinery, Nov. 2023, pp. 2381–2395. ISBN: 979-8-4007-0050-7. DOI: 10.1145/3576915.3623063. URL: https://dl.acm.org/doi/10.1145/3576915.3623063 (visited on 03/17/2025).

[57] Leona Lassak et al. “"It’s Stored, Hopefully, on an Encrypted Server”: Mitigating Users’ Misconceptions About {FIDO2} Biometric {WebAuthn}”. In: 2021, pp. 91–108. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/lassak (visited on 03/16/2025).

[58] Leona Lassak et al. “A Comparative Long-Term Study of Fallback Authentication Schemes”. In: *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. CHI ’24. Honolulu, HI, USA: Association for Computing Machinery, 2024. ISBN: 9798400703300. DOI: 10.1145/3613904.3642889. URL: https://doi.org/10.1145/3613904.3642889.

[59] Leona Lassak et al. “From TOTPs to Security Keys: Studying the Reality of Passwordless FIDO2 Authentication With PIN and Biometrics in a Corporate Environment”. In: *Twenty-First Symposium on Usable Privacy and Security (SOUPS 2025)*. 2025, pp. 371–389.

[60] Leona Lassak et al. “Why Aren’t We Using Passkeys? Obstacles Companies Face Deploying FIDO2 Passwordless Authentication”. In: *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 7231–7248. ISBN: 978-1-939133-44-1. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/lassak.

[61] Victor Le Pochat et al. “Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation”. In: *Proceedings of the 26th Annual Network and Dis-*

tributed System Security Symposium. NDSS 2019. Feb. 2019. DOI: 10.14722/ndss.2019.23386.

[62] *List methodology - Tranco*. [Online; accessed 2025-08-06]. URL: https://tranco-list.eu/methodology.

[63] *List of FIDO2 and Passkey Supported Websites and Services | Hideez*. [Online; accessed 2025-08-06]. URL: https://hideez.com/en-de/pages/supported-services.

[64] Ahmed Tanvir Mahdad et al. "Breaching Security Keys without Root: FIDO2 Deception Attacks via Overlays exploiting Limited Display Authenticators". In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. Salt Lake City UT USA: ACM, Dec. 2024, pp. 1686–1700. ISBN: 979-8-4007-0636-3. DOI: 10.1145/3658644.3690286. URL: https://dl.acm.org/doi/10.1145/3658644.3690286 (visited on 03/19/2025).

[65] Alexander Matzen et al. "Challenges and Potential Improvements for Passkey Adoption—A Literature Review with a User-Centric Perspective". In: *Applied Sciences* 15.8 (2025). ISSN: 2076-3417. DOI: 10.3390/app15084414. URL: https://www.mdpi.com/2076-3417/15/8/4414.

[66] Microsoft. *Passwordless authentication | Microsoft Security*. [Online; accessed 2025-04-02]. URL: https://www.microsoft.com/en-us/security/business/solutions/passwordless-authentication.

[67] Mozilla. *CredentialsContainer: create() method - Web APIs | MDN*. [Online; accessed 2025-04-02]. Mar. 2025. URL: https://developer.mozilla.org/en-US/docs/Web/API/CredentialsContainer/create.

[68] Mozilla. *CredentialsContainer: get() method - Web APIs | MDN*. [Online; accessed 2025-04-02]. Mar. 2025. URL: https://developer.mozilla.org/en-US/docs/Web/API/CredentialsContainer/get.

[69] Mozilla. *Fetch API - Web APIs | MDN*. [Online; accessed 2025-04-02]. Mar. 2025. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API.

[70] *Passkey Catalogue - Enpass*. [Online; accessed 2025-08-06]. URL: https://www.enpass.io/passkeys-catalogue/.

[71] *Passkey Directory*. [Online; accessed 2025-08-06]. URL: https://passkeys.2fa.directory/de/.

[72] *Passkey-Verzeichnis – Keeper Security*. [Online; accessed 2025-08-06]. URL: https://www.keepersecurity.com/de_DE/passkeys-directory/.

[73] *Passkeys Directory*. [Online; accessed 2025-08-06]. URL: https://passkeys-directory.dashlane.com/.

[74] *Passkeys Index*. [Online; accessed 2025-08-06]. URL: https://passkeyindex.io/.

[75] *Passkeys Integration: Add Passkeys Authentication to Your Platform*. [Online; accessed 2025-08-27]. URL: https://www.passkeys.com/integrations.html.

[76] *Passkeys Services | Passkeys App*. [Online; accessed 2025-08-06]. URL: https://passkeys.2stable.com/services/.

[77] *passkeys.dev - Device Support*. [Online; accessed 2025-08-25]. URL: https://passkeys.dev/device-support/.

[78] *passkeys.dev - Libraries*. [Online; accessed 2025-08-27]. Sept. 2022. URL: https://passkeys.dev/docs/tools-libraries/libraries/.

[79] *PublicKeyCredential: signalAllAcceptedCredentials() static method - Web APIs | MDN*. [Online; accessed 2025-08-18]. July 2025. URL: https://developer.mozilla.org/en-US/docs/Web/API/PublicKeyCredential/signalAllAcceptedCredentials_static.

[80] *PublicKeyCredential: signalUnknownCredential() static method - Web APIs | MDN*. [Online; accessed 2025-08-18]. June 2025. URL: https://developer.mozilla.org/en-US/docs/Web/API/PublicKeyCredential/signalUnknownCredential_static.

[81] J. Rautenstrauch et al. "To Auth or Not To Auth? A Comparative Analysis of the Pre- and Post-Login Security Landscape". In: *2024 IEEE Symposium on Security and Privacy (SP)*. ISSN: 2375-1207. Los Alamitos, CA, USA: IEEE Computer Society, May 2024, pp. 97–97. DOI: 10.1109/SP54263.2024.00094. URL: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00094.

[82] Thomas Roche et al. "A Side Journey To Titan". In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 231–248. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/roche.

[83] Dr.-Ing. Dominik Schürmann. *PIN Bypass in Passwordless WebAuthn on microsoft.com and Nextcloud | Hardware Security SDK*. [Online; accessed 2025-08-20]. Aug. 2020. URL: https://hwsecurity.dev/2020/08/webauthn-pin-bypass/.

[84] Marco Squarcina et al. “Can I Take Your Subdomain? Exploring Same-Site Attacks in the Modern Web”. In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 2917–2934.

[85] *SSO-Monitor*. [Online; accessed 2025-08-06]. URL: https://sso-monitor.me/.

[86] *Subdomain takeovers - Security | MDN*. [Online; accessed 2025-08-19]. June 2025. URL: https://developer.mozilla.org/en-US/docs/Web/Security/Subdomain_takeovers.

[87] *The "keybase.txt" Well-Known Resource Identifier*. [Online; accessed 2025-08-06]. URL: https://keybase.io/docs/keybase_well_known.

[88] Enis Ulqinaku et al. “Is Real-time Phishing Eliminated with FIDO Social Engineering Downgrade Attacks against FIDO Protocols”. In: 2021, pp. 3811–3828. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/ulqinaku (visited on 03/17/2025).

[89] W3C. *Web Authentication: An API for accessing Public Key Credentials - Level 3*. [Online; accessed 2025-04-02]. Jan. 2025. URL: https://www.w3.org/TR/webauthn-3/.

[90] *Wayback Machine*. [Online; accessed 2025-08-06]. URL: https://web.archive.org/.

[91] *Web Authentication: An API for accessing Public Key Credentials - Level 3*. [Online; accessed 2025-08-06]. Jan. 2025. URL: https://www.w3.org/TR/webauthn-3/#sctn-related-origins.

[92] *Websites and Apps with Passkey Support*. [Online; accessed 2025-08-06]. URL: https://www.passkeys.io/who-supports-passkeys.

[93] *Websites that Use and Support Passkeys: Passkey Supported Sites Directory*. [Online; accessed 2025-08-06]. URL: https://www.passkeys.com/websites-with-passkey-support-sites-directory.

[94] *Well-Known URIs*. [Online; accessed 2025-08-06]. URL: https://www.iana.org/assignments/well-known-uris/well-known-uris.xhtml.

[95] Davey Winder. *16 Billion Apple, Facebook, Google And Other Passwords Leaked*. [Online; accessed 2025-08-25]. June 2025. URL: https://www.forbes.com/sites/daveywinder/2025/06/20/16-billion-apple-facebook-google-passwords-leaked---change-yours-now/.

[96] Davey Winder. *184,162,718 Passwords and logins leaked — Apple, Facebook, Snapchat*. [Online; accessed 2025-08-25]. May 2025. URL: https://www.forbes.com/sites/daveywinder/2025/05/23/184162718-passwords-and-logins-leaked---apple-facebook-snapchat/.

[97] *WSTG - Latest | OWASP Foundation*. [Online; accessed 2025-08-11]. URL: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/03-Identity_Management_Testing/04-Testing_for_Account_Enumeration_and_Guessable_User_Account.

[98] Tarun Kumar Yadav et al. “A Security and Usability Analysis of Local Attacks Against FIDO2”. In: *Proceedings 2024 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. San Diego, CA, USA: Internet Society, 2024. ISBN: 978-1-891562-93-8. DOI: 10.14722/ndss.2024.24327. URL: https://www.ndss-symposium.org/wp-content/uploads/2024-327-paper.pdf (visited on 08/26/2025).

[99] Yubico. *Discover YubiKey 5 | Authentication for Secure Login | Yubico*. [Online; accessed 2025-04-02]. URL: https://www.yubico.com/products/yubikey-5-overview/.

A Appendix

Table 3 shows why some websites were excluded from our analysis of passkey implementations. Many used shared login systems, which we only needed to analyze once. For example, we found 38 Gitea [41] instances and 134 domains using the Norwegian aID broker [4]. Other domains were excluded because account creation was restricted (such as banking, insurance, or education) or required paid access. Some also asked for information we could not provide, including payment details, business email addresses, or social security numbers.

Exclusion Reason	# Domains
Broker with shared login system	178
Restricted or paid access	170
No passkeys available (only 2FA via TOTP)	56
Duplicates missed by the merger	46
Website down or unreachable	33
Offers “Passkeys as a Service”	28
Foreign phone required	22
Mobile app required	21
Business email required	19
Unknown error during registration	18
Platform (i.e., OS, Browser)	17
No login found	13
Payment method required	10
Social security number required	10
Personal identification required	9
Geolocation restricted (even with VPN)	6
Hardware authenticators only (U2F)	5
Selfhosted app requires installation	2
Chrome extension required	1
Σ	664

Table 3: Reasons for Excluding Websites From Our Analysis.