A Framework for Abusability Analysis: The Case of Passkeys in Interpersonal Threat Models

Alaa Daffalla and Arkaprabha Bhattacharya, *Cornell University;* Jacob Wilder, *Independent Researcher;* Rahul Chatterjee, *University of Wisconsin—Madison;* Nicola Dell, *Cornell Tech;* Rosanna Bellini, *New York University;* Thomas Ristenpart, *Cornell Tech*

https://www.usenix.org/conference/usenixsecurity25/presentation/daffalla

This paper is included in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

A Framework for Abusability Analysis: The Case of Passkeys in Interpersonal Threat Models

Alaa Daffalla
Cornell University

Arkaprabha Bhattacharya
Cornell University

Jacob Wilder
Independent Researcher

Rahul Chatterjee
University of Wisconsin–Madison

Nicola Dell
Cornell Tech

Rosanna Bellini
New York University

Thomas Ristenpart
Cornell Tech

Abstract

The recent rollout of passkeys by hundreds of web services online is the largest attempt yet to achieve the goal of passwordless authentication. However, new authentication mechanisms can often overlook the unique threats faced by at-risk users, such as survivors of intimate partner violence, human trafficking, and elder abuse. Such users face interpersonal threats: adversaries who routinely have physical access to devices and either know or can compel disclosure of passwords or PINs. The extent to which passkeys enable or mitigate such interpersonal threats has not yet been explored.

We perform the first analysis of passkeys in interpersonal threat models. To do so, we introduce an abusability analysis framework to help practitioners and researchers identify ways in which new features can be exploited in interpersonal threat models. We then apply our framework to the setting of passkeys, ultimately investigating 19 passkey-supporting services. We identify a variety of abuse vectors that allow adversaries to use passkeys to cause harm in interpersonal settings. In the most egregious cases, flawed implementations of major passkey-supporting services allow ongoing illicit adversarial access with no way for a victim to restore security of their account. We also discover abuse vectors that prevent users from accessing their accounts or that help attackers emotionally manipulate (gaslight) users.

1 Introduction

Password-based authentication remains the most widely used mechanism for user authentication on the web. Recently, however, there has been a push to adopt cryptographic, phishing-resistant credentials to achieve passwordless authentication via the FIDO2 standard [1]. Unlike FIDO2's prior use in second-factor authentication (2FA), passkeys are cryptographic credentials that can be used for primary authentication. Passkeys are, by design, portable: they can be exported to be used from another device, or synchronized across devices via passkey providers such as Apple, Google, or password managers. Passkeys have seen rapid uptake in the last four years, with Google reporting that, as of May 2024, 400 million accounts have enabled passkeys [2].

Despite growing research on passkeys' availability [3], usability [4–9], and security [10–13], there has been no investigation into their role in interpersonal threat models. Adversaries in such settings may have periodic physical access to (unlocked) devices and the ability to guess or compel disclosure of authentication information from highly vulnerable user populations, including children [14, 15], young adults [16, 17], disabled persons [18], or survivors of intimate partner violence (IPV) [19, 20], human trafficking [21], and elder abuse [22, 23], among others [24, 25].

We therefore initiate exploration of passkeys in interpersonal threat models. To do so, we introduce a new abusability analysis framework that generalizes one used by Bellini et al. [26] to assess tech abuse in the context of financial apps. The goal is to identify abuse vectors, or sequences of steps a malicious user can perform with a digital product to enact harm against a target. Our six-stage framework provides structure for an analyst team to identify threat models, assess functionality, hypothesize abuse vectors, design and execute stepthrough protocols to test viability of the hypothesized abuse vectors, and summarize the findings in easy-to-understand abuse scenarios.

We exercise our framework by applying it to recent passkey implementations using interpersonal threat models gleaned from the academic literature. Our threat models capture an interpersonal adversary that either seeks unauthorized access to their victim's account, to deny the victim access to their accounts, or to gaslight (emotionally manipulate or harm) the victim via passkey features. To do so, the adversary can either periodically use a victim's unlocked device, or be able to log in remotely from an adversarial device, and uses either standard user interfaces in typical ways or has expertise sufficient for reconfiguring software settings.

To understand the current state of passkey implementations, we explore their common functions through a functionality assessment of the UI affordances of six online web services.

Interacting with UIs as typical end users, we examine how passkeys may be registered, authenticated, managed, exported, and (if possible) revoked. In doing so, we uncover a range of buggy implementations, inconsistent design patterns around passkey management, and subtleties that confused even expert team members.

We then worked to hypothesize seven possible abuse vectors, three related to unauthorized access, and two each related to denial of service and gaslighting. For example, one is an adversary using local access to add their biometrics to a victim device in order to retain illicit access to an account. Another uses a remote device to login to a victim's account and revoke their passkeys to perform denial of service.

To evaluate the viability of these abuse vectors, we designed stepthrough protocols in which an analyst alternatively plays the role of simulated victim and adversary, to assess whether (and in what exact way) abuse vectors can be enacted and, if so, whether victims can detect and/or recover from the harm. We apply our stepthrough protocols to four client-side platforms (iOS, MacOS, Windows, Android) and 15 popular services that support passkeys.

Unfortunately, the abuse vectors are overwhelmingly successful: anyone with sufficient knowledge and a basic level of adversarial access can leverage passkeys to cause harm, with those harms often exacerbated by lack of victim visibility and recoverability. For example, we show how an adversary adding their biometric to a victim's device does not trigger any clear UI-visible notifications and allows ongoing access to all victim accounts (even if a victim routinely logs out of accounts). We also discover that passkey portability features allow passkey cloning: an attacker with one-time access to a phone and knowledge of a victim's PIN can obtain passkey-based access from another device. For some services (CVS, Porkbun, TikTok) this access is *impossible to detect* and we could determine *no way to revoke access*.

Summary. Our key contributions are as follows:

- We detail a new framework for abusability analysis to help practitioners and researchers identify abuse vectors.
- We apply our framework to passkey deployments across a variety of clients and 19 online services.
- We identify seven abuse vectors that can be exploited in interpersonal threat models to cause harms such as unauthorized access, denial of service, and gaslighting. In some cases services provide the user no visibility into abusive actions or ability to recover from them.

While attacks due to adversaries with high levels of access may sometimes be inevitable, our work suggests concrete mitigations. Enhancing abuse prevention through improved notifications, visibility into passkey use on UIs, and integrating our auditing techniques into feature reviews could help mitigate these threats. Moreover, our abusability analysis framework can be adopted by testing teams to discover problems and explore possible mitigations *before* deployment.

2 Background and Related Work

FIDO2. The Fast IDentity Online (FIDO) Alliance [27] is an open industry association that develops and promotes alternate authentication standards to passwords. FIDO has published three specifications for cryptographic authentication: FIDO Universal Authentication Framework (FIDO UAF), FIDO Universal Second Factor (FIDO U2F), and the FIDO protocols 1.0, 2.0 (FIDO, FIDO2) [1].

The original FIDO protocol (aka FIDO 1.0) was tied to hardware devices (e.g., hardware tokens, biometric scanners), called authenticators, making them both difficult to standardize and manage at a software level. In response, the World Wide Web Consortium (W3C) working group has standardized the Web Authentication protocol (WebAuthn) [28] for allowing primary or second factor authentication via platform-managed cryptographic credentials (FIDO2).

FIDO2 specifies challenge-response interactions between an authenticator, client, and a web service known as the relying party (RP). An authenticator is a hardware or software component on the client side that manages the generation and use of credentials. Credentials, consisting of a public-private key pair, are unique to each RP, user account, and authenticator. Roaming authenticators do not synchronize credentials, requiring users to physically transfer a hardware key. Platform vendors have also implemented their own authenticators and support for syncing credentials across multiple user devices, giving rise to the notion of *passkeys*—cryptographic credentials that are portable and can be used instead of passwords. The unsynchronizable FIDO 1.0 credentials may also be referred to as passkeys.

Challenges with FIDO2 uptake. The FIDO2 protocol involves a complex ecosystem of RPs, client applications, and authenticators, making widespread deployment of passkeys reliant on adoption by major browsers, operating systems (OSes), and online services. The protocol is currently supported by popular browsers (Chrome, Firefox, Edge, and Safari) and OSes (MacOS, iOS, Android, Windows) [29]. Many services have also started deploying passkeys to end users. Google introduced passkeys across all its accounts on major platforms in May 2023. Although only five services supported FIDO2 for passwordless authentication in a 2022 analysis of Tranco's 100K [30], this has increased to 145 services[1].

Despite such a rapid roll-out across web services, multiple research studies have highlighted that FIDO2's WebAuthn and CTAP protocols are not without their flaws [10–13]. For instance, Jubur et al. [31] discovered a timing attack that exploits push notifications. Furthermore, Kuchhal et al. [30] identified client-side attacks enabled by uncertified authenticators and lack of mitigations on the server side. FIDO2 is also not immune to social engineering attacks; such as down-

[1]According to 1Passwords's Passkeys directory at the time of writing: https://passkeys.directory/

grading to weaker 2FA alternatives [32], or tricking victims into authenticating the actions of an attacker [30].

Since the release of FIDO2's U2F specification for using security keys for 2FA in 2017, studies have also highlighted usability and security concerns [4–9]. Ciolino et al. [9] and Owens et al. [4] found users faced difficulties finding and following setup instructions for security keys, especially when pairing keys or roaming authenticators. Account recovery was also highlighted by participants as a significant concern, particularly when a security key is lost or stolen [7].

Interpersonal threats. A major advantage of device-bound credentials, such as passkeys, is that they mitigate the risk of phishing attacks and compromise [33]. While these are important to mitigate, a drawback of such an approach is that device compromise grants an adversary access to all credentials on the device. Credential security is thus reduced to device security [3, 34]. This places additional stressors on users who share devices or authentication information [33] and those targeted by interpersonal threats.

Interpersonal threat models are contexts where an adversary has a pre-existing social relationship with the target, and exploits this relationship for proximity to a victim's physical and/or digital assets. Proximity to a victim could be physical, such as living together or frequently being in the same location (e.g., occupational), or logical, such as via device/account sharing. Traditional security measures often fail to mitigate these types of threats because adversaries employ highly targeted attacks and are driven by complex, social objectives like social control, surveillance, and intimidation [26, 35].

Such factors of interpersonal threats make device and account compromise extremely common for users targeted by such adversaries. For instance, many at-risk users—such as teenagers [16, 17], children [14, 15], people with visual or mental impairments [18], and older adults [22, 23, 36]—may appoint a trusted individual to help configure their account security, necessitating device access. In cases involving abuse, such as survivors of human trafficking [21] and intimate partner violence [19, 37], victims may have no choice other than to provide an adversary with access to their devices, either through threat of coercion or social manipulation.

Auditing for abuse. A small amount of prior work has introduced ways to conduct audits of systems for abuse [26, 38, 39]. Working in the context of IoT devices, Slupska et al. [38] apply threat modeling—which is mostly used to identify technical risks inherent to systems—to instead identify risks to people. Similarly, Bellini et al.'s work [26] introduces a methodological approach to elicit digital-safety concerns in consumer financial applications in the context of IPV. Their work utilizes UI stepthroughs—an experimental investigation into the UI features of the technology being audited—as a fundamental part of their approach. Our framework builds off and generalizes Bellini et al.'s [26] auditing approach.

3 Abusability Analysis Framework

In this section, we contribute a generalized framework for conducting abusability testing on technology products and features, and discuss how we applied it to passkeys. Our framework's goal is to enable researchers, designers, and engineering teams to proactively identify exploitable *abuse vectors*—steps a malicious user might take to enact harm against a target—in a feature or product. Ideally, such review would occur before new features are released, enabling design to be revisited to add friction or fully prevent 'abusability' (a portmanteau of abuse and usability) of the feature at hand.

To prepare for an abusability analysis, teams and organizations should assign a main point-of-contact person (point person) to oversee the design and delivery of such an audit. If such an analysis is intended to be used inside an organization, technology companies may already have assigned a point person to coordinate specialized training for other types of system abuse (i.e., spotting unusual activity or unauthorized access). Due to its reliance on pre-existing usability methodologies, a team may wish to seriously consider the incorporation of a team member experienced in UX or service design. We anticipate that our abusability analysis (as described below) would also work as an extension of targeted training programs that already address some system vulnerabilities.

Our abusability analysis framework involves a structured sequence of six auditing steps, the output of which is a set of tested *abuse vectors*, along with which ones were exploitable, detailed exploit descriptions, and *abuse scenarios* that summarize the potential harms arising from exploitation. We explain each of the steps below. Throughout, we refer to the people conducting the abusability analysis as *analysts*.

(1) Identify threat models. Threat model identification is the process of defining who the adversary is in relationship to the target, what their goals are, and what their capabilities are. We suggest using succinct one to two sentence descriptions for threat models when performing abusability analysis, and avoiding jargon that may be unfamiliar to analysts or other stakeholders. For example, a threat model we consider later is an interpersonal attacker personally known to the victim, who seeks to maintain the ability to log into the victim's account from a different device, and knows the victim's password.

Most often, an analysis should consider multiple, related threat models to increase coverage of potential abuse issues. A good starting point is to consider the common qualities of at-risk populations [24] or an area of acute risk for these communities, such as Bellini et al.'s [26] focus on digital financial abuse. Some teams will already be familiar with relevant threat models for the context and can rely on their expertise. Teams might complement existing expertise with a rapid review [40]—a process for quickly reviewing high-quality, authoritative resources (such as academic scholarship). Teams that lack expertise could opt for more rigorous reviews that prioritize coverage [41]. In performing such re-

#	Stage	Description	Output
1	*Identify threat models*	Determine a set of threat models, optionally aided by literature review	Set of threat models
2	*Assess functionality*	Open-ended exploration of system features	User pathways; screen captures; inventory of functionality
3	*Hypothesize abuse vectors*	Develop set of hypothetical abuse vectors by reviewing system functionality in light of threat models	List of hypothesized abuse vectors
4	*Design stepthrough protocols*	Develop set of stepthrough protocols to guide testing hypothesized abuse vectors, detection, and cleanup	List of step-by-step protocols to test abuse vectors
5	*Test abuse vectors*	Conduct stepthroughs to test abuse vectors	Evaluate viability of abuse vectors; stepthrough transcripts
6	*Summarize with scenarios*	Distill successful abuse vectors into short scenarios	Abuse scenarios

Figure 1: A summary of our six-stage abusability analysis framework.

views, an analyst should draw out documented adversarial goals and capabilities until saturation is reached, meaning no new information is gleaned from published works.

In many cases, one may be able to generalize population-specific threat models without substantively affecting the subsequent steps of the analysis. For example, an interpersonal attacker could be an intimate partner or someone else with similar goals (family member, friend, co-worker).

Given our focus on abuse, the adversarial capabilities we consider tend towards UI-bound adversaries [19] that interact via standard UIs with systems configured in standard (non-adversarial) ways. We also consider *reconfiguration adversaries* that modify software settings, including installing readily available applications. For example, an abuser that installs a VPN and uses built-in browser tools to modify user agent strings in order to obfuscate account access via spoofing [42] is an example of a reconfiguration adversary. Our framework may not be as effective for threat models with technical capabilities beyond reconfiguration or UI-bound adversaries (e.g., the ability to deploy software exploits such as privilege escalation or remote code injection).

Stage (1) for passkeys: Our team collectively has decades of specialized knowledge in computer security, authentication protocols, and interpersonal abuse, encompassing contexts such as IPV, human trafficking, and sex work. As such, we felt comfortable relying on our existing knowledge of the literature and practical realities of tech abuse to help us identify relevant threat models, and did not perform a rapid review.

In our considered threat models, we presume an adversary that targets a specific individual (rather than a group) and that adversary is within the same social circles as the target. For example, the adversary may be an intimate partner, other family member, co-worker, or other close acquaintance.

We consider several goals from the literature. The first is *unauthorized access* where the adversary seeks to obtain one time or ongoing privileged access to a victim's account. The second is *denial of service* where the adversary seeks to lock-out, restrict, or disrupt a victim's account access. Third, we consider *gaslighting*, where the adversary seeks to emotionally manipulate the victim into doubting their digital safety.

Our threat models also assume the adversary has either: (1) one-time or periodic *physical access* to the victim's device and the ability to unlock it (e.g., with a PIN); and (2) the one-time ability to login to the victim's account from a separate adversarial device which we call *remote login*. Such scenarios have been reported by security and privacy scholars as arising often in interpersonal abuse contexts [17–20, 43].

(2) Assess functionality. A functionality assessment is a focused survey of system behavior. In this context, the analyst seeks to produce a full inventory of the breadth of the system's functions from a user's perspective. In this phase, UI stepthroughs are used as a primary investigative method. This method is grounded in the cognitive walkthrough frameworks used to evaluate the usability of websites [44] but we adapt this, following [25], to evaluate areas of potential abusability.

During this phase, the analyst interacts with the system using standard UI tools and features like a regular (non-malicious) user, documenting UI details, user journeys, and capturing screenshots. They may also gather web traffic and relevant source files (i.e., from web pages) to understand the system and its behavior. This phase is considered more open-ended exploration thus the analyst is not required to follow a specific protocol. However, to structure the stepthrough (and ensure consistency between like technologies) they could identify a set of tasks representative of the system behavior.

Stage (2) for passkeys: Section 4 describes the results of this stage in detail. Briefly, we first enumerated a variety of services that deploy passkeys and identified a set of tasks associated with the passkey lifecycle: (a) configuring a passkey for primary authentication; (b) logging into services with the passkey; (c) reviewing the information visible to users about the passkey post-setup/use; (d) exporting or transferring the passkey to another device; and (e) disabling the passkey for service access. We then performed UI stepthroughs for these tasks for each service. These stepthroughs already surfaced several functionality problems, and pointed towards some problems in our chosen threat models.

(3) Hypothesize abuse vectors. An abuse vector refers to the steps an attacker takes to cause harm. For example, an

Adversarial goal	Capability	Abuse Vector	Harm
Unauthorized Access	Periodic physical access	Adversarial biometrics	Ongoing local login
	One-time physical access	Passkey cloning	Ongoing remote login
	One-time remote login	Adversarial passkeys	Ongoing remote login
Denial of Service	One-time physical access	Local passkey deletion (device & OS level)	Deny victim's login
	One-time remote login	Remote passkey revocation (service)	Deny victim's login
Gaslighting	Periodic remote login	Spoofing information on the service (ASIs)	Mislead/harass victim
	Periodic physical access	Spoofing information locally (device & OS)	Mislead/harass victim

Figure 2: Adversarial goals and capabilities identified in Stage (1) mapped to abuse vectors hypothesized in Stage (3). Additionally, for each abuse vector we show the resulting harm (e.g., attacker obtains ongoing unauthorized local login to victim's account).

abuse vector could be: use a known PIN to unlock a target phone, add an adversarial biometric to the device, and then use that biometric to access the device later. We recommend using clear, prescriptive language to describe these actions.

In the third stage of our framework, the analysts brainstorm a set of hypothetical abuse vectors, each of which is consistent with one of the threat models identified in the prior stage. To do so, the analysts hold a series of meetings with relevant members of expertise in computer security, and trust and safety. During these meetings, the team can review the notes and recordings from the functionality assessment phase in light of the threat models to identify potential abuse vectors.

Meetings may raise questions about new system operations and environments, prompting analysts to perform additional functionality assessments. These continue until the team identifies no new hypothetical abuse vectors. The outcome of this phase is a set of to-be-tested abuse vectors.

Stage (3) for passkeys: The team performed a sequence of six meetings in which members who performed functionality assessments presented their discoveries to the broader group, and we did active brainstorming to identify potential abuse vectors within our previously identified threat models (unauthorized access, denial of service, and gaslighting each with physical access and remote login adversarial capabilities). We then hypothesized how an interpersonal adversary might use the observed passkey functionality to enact their goal. We summarize the resulting abuse vectors in Figure 2, and clarify the capabilities required to enact each abuse vector when we discuss our testing results in Section 5. As we will see, each hypothesized abuse vector was viable on at least one service.

Although the abuse vectors we discover do not cover every possible issue, they effectively highlight key vulnerabilities and risks related to the deployment of passkeys.

(4) Design stepthrough protocols. Stepthrough protocols are detailed step-by-step guides for testing hypothesized abuse vectors. A protocol provides a series of steps for an analyst to follow that simulate both victim and adversary actions related to an abuse vector. A good stepthrough protocol should cover not only a way to test the viability of the abuse vector (i.e., carrying out the attack), but also whether the victim can detect the abuse and, when relevant, recover back to a safe state.

As example, a protocol for the hypothesized adversarial biometric abuse vector could be: (V) setup a simulated victim device with PIN; (A) unlock the device with known PIN; (A) add a new biometric to unlock the device; (A) lock the device; (A) unlock the device with biometric; (V) check device settings to see if a biometric is added; (V) attempt to remove the added biometric. Here the parenthetical letters indicate whether the victim (V) or adversary (A) are being simulated.

As in this example, stepthroughs will tend to be semi-structured thus allowing the analyst some flexibility in the exact sequence of UI steps and navigation paths required to implement each step. A stepthrough protocol should also come with description of expected testing environments, i.e., the type and number of devices, the OS and application software used. In short, the stepthrough protocol should provide an experimentation plan that an analyst can easily follow.

In general, an individual stepthrough protocol might be used to test multiple abuse vectors, and the next stage benefits from identifying the smallest feasible set of stepthrough protocols that cover all the hypothesized abuse vectors.

Stage (4) for passkeys: We developed five stepthrough protocols corresponding to the first five abuse vectors in Figure 2. Stepthrough protocols for passkey cloning and adversarial passkeys were also used to test the gaslighting vectors (see Figure 5). Each protocol consists of a set of tasks simulating victim and adversarial actions to carry out the attack. We also provide a summary of the stepthrough protocols for two of our abuse vectors in Figure 8. A stepthrough protocol covers the realization of the abuse vector, the detectability of the attack (i.e., can the victim detect the attack through system provided UI affordances), and the recoverability of the attack (i.e., can the victim recover from the attack and prevent adversarial access through system provided UI features).

(5) Test abuse vectors. In this phase, an analyst conducts stepthrough protocols to test the hypothesized abuse vectors. For each stepthrough protocol, the analyst: (1) prepares the stepthrough testing environment; (2) determines and documents the precise UI paths the analyst follows; and (3) concludes whether the abuse vector succeeded, along with the extent to which the victim can detect and recover from it.

Teams should consider whether they should perform each stepthrough protocol multiple times. For example, if determining the precise UI steps is sensitive to analyst back-

Scenario A: *Alex and Billy have been dating. Billy saw that Alex uses TikTok, logging in just using the device's pin (i.e., a passkey). Billy convinces Alex to share the 6-digit pin to their phone. When Alex is in the bathroom, Billy unlocks Alex's device and exports the TikTok passkey to Billy's iPhone using AirDrop. Billy logs into Alex's account using the exported passkey. Billy is now able to monitor Alex's TikTok messages and interactions. When Alex becomes suspicious due to comments Billy made, Alex resets their password and sets up a new passkey. Unbeknownst to Alex, this doesn't work: Billy still has remote, covert access to the account.*

Scenario B: *Andy and Mark are coworkers. Andy uses Yahoo Mail for personal use and has a passkey registered on a Yubico security key. Mark is interested in learning whether Andy landed a job with another company. He sees that whenever Andy logs into his Yahoo mail, he inserts the security key and taps the sensor on the key. One day, Andy leaves his computer unlocked with the security key inserted. Mark accesses the computer, navigates to Yahoo, and attempts to login using the browser autofill feature that populates user information. Mark re-inserts the security key and taps the sensor on the key. He can now access Andy's Yahoo mail account.*

Scenario C: *Mona and Adam are teenagers who attend the same school. They are active users on Roblox. Mona has gone viral with hundreds of people playing games she created. Adam's jealousy leads him to shoulder surf to learn Mona's Roblox password. He uses it to log into Mona's Roblox account from his computer, revokes all her passkeys on the account, and then resets the password. Mona was locked out of her account and could not find any way to recover it.*

Figure 3: Vignettes describing plausible abuse scenarios that result from our analysis, and that correspond to a subset of the transcripts generated in Stage 5. Scenario A and B describe how an interpersonal attacker obtains unauthorized access due to passkey cloning and improper user verification respectively. Scenario C describes a peer locking a victim out of their account.

ground, then having multiple analysts independently run the stepthrough protocol could be warranted. Or if the studied systems exhibit non-deterministic behavior in response to user actions (e.g., due to bugs), then an analyst may need to execute a protocol multiple times. Also, it will often be the case that one will want to execute stepthrough protocols for different environments (e.g., client software configurations).

At the end of this stage, the analyst team should determine which hypothesized abuse vectors are viable, and have a detailed transcript of the steps taken during the stepthrough protocol. The latter is analogous to a proof-of-concept exploit in software security, and will help in reproducing the abuse vector, e.g., during disclosure processes. Relatedly, the transcript will also be useful to relevant product teams to help them understand the design features implicated in the abuse.

Stage (5) for passkeys: Two authors executed all five stepthrough protocols for each of the 15 services. They used test accounts with synthetic user information and dedicated email addresses and phone numbers for research purposes only. Each protocol required testing with multiple client-side environments (MacOS, iOS, Android, and Windows). In total, we performed 87 stepthroughs. We discuss the experimental setups and stepthrough results in Section 5.

(6) Summarize with scenarios. The final step is to craft abuse scenarios, which contextualize an abuse vector's harms. The scenarios should concretize via example one or more abuse vectors. We suggest a distilled scenario that consists of a concise, high-level narrative overview of the sequence of events leading to harm. These scenarios serve the purpose of documenting identified abuse vectors in a contextualized way and also communicating them in an easy-to-understand way to to relevant stakeholders, including product managers, executives, engineers, lawyers, and others who may have the power to develop and deploy abuse mitigation [26].

Stage (6) for passkeys. We provide vignettes for plausible abuse scenarios in Figure 3.

4 Passkey Functionality

In this section we describe our passkey functionality assessment in greater detail. We assess functionality across six different services, covering a range of company types: technology companies (Google, Amazon, Paypal), retailers (CVS, Target), a domain registrar (Porkbun). Although not comprehensive, these services were selected as an ample cross-section of major services that support passkeys.

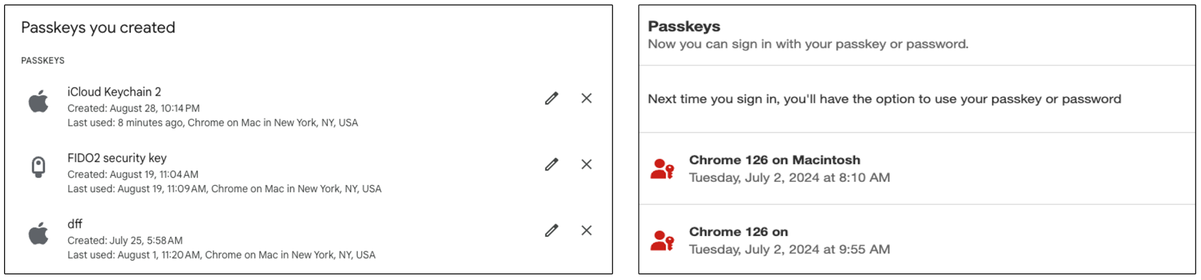
Our functionality assessment stepthroughs were conducted between Feb.–Apr. 2024 using a MacBook Pro (MacOS Sonoma v.14.5) using the Chrome browser (v.124.0.6367.118/.119) and the default authenticator provided by Apple (Touch ID) to store passkeys. We used Safari (v.17.6, 19618.3.11.11.5) for two services (CVS, Paypal) where passkey registration was not supported by Chrome.

(a) Configuring a passkey. We identify two primary pathways to configuring a passkey: during initial sign-up or configuration following login. The latter may require, or not, passing an additional authentication challenge.

Google, Amazon, Paypal, CVS, and Target require password-based authentication during account creation prior to creating a passkey. To set up a passkey, users must first authenticate with non-passkey credentials, navigate to the account settings and security pages—called account security interfaces (ASIs) [42]—and configure the passkey on this interface. In our testing, Target and CVS did not require re-authentication through an additional challenge (e.g., via one-time password, or OTP) prior to creating the passkey, whereas the other services required passing the extra check.

Porkbun is the only service we identified that allows passkey-only authentication immediately at account creation. We identify that common errors (e.g., an improperly formatted field) introduced during sign up form submission can generate extraneous, unusable passkeys for the same username and email, despite unsuccessful registration.

CVS enables passkey creation following password authentication on the account. Once a user enters their password


Passkeys you created

PASSKEYS

iCloud Keychain 2
Created: August 28, 10:14 PM
Last used: 8 minutes ago, Chrome on Mac in New York, NY, USA

FIDO2 security key
Created: August 19, 11:04 AM
Last used: August 19, 11:09 AM, Chrome on Mac in New York, NY, USA

dff
Created: July 25, 5:58 AM
Last used: August 1, 11:20 AM, Chrome on Mac in New York, NY, USA

Passkeys
Now you can sign in with your passkey or password.

Next time you sign in, you'll have the option to use your passkey or password

Chrome 126 on Macintosh
Tuesday, July 2, 2024 at 8:10 AM

Chrome 126 on
Tuesday, July 2, 2024 at 9:55 AM

Figure 4: Left to right, Screenshots of **(left)** Google's and **(right)** Target's ASIs for passkey management.

and it is verified they are presented with a prompt to enroll in passwordless authentication. However, during this initial passkey registration, the FIDO2 service-selected credential identifier [45] associated with the passkey is stored in the browser's local storage. This stored value is required during subsequent authentications so clearing local storage disables the CVS passkey, even though it remains available in the authenticator. Our testing also revealed that delays or switching windows can result in registration failure, with the browser displaying a generic "request timed out" error (see Figure 10).

All services, except Porkbun, support multiple passkeys per account, if they are associated with different cloud providers. We identify that Google, Paypal, Target, and Amazon consistently send an email notification each time a passkey credential is added, while Porkbun and CVS do not.

(b) Authentication with passkeys. We identify that passkeys can serve as primary authentication on all six services. Typically, the login process begins with a username and/or email address submission to the service. We observed varying flows: some prompt the user for their email address or username, followed by biometric authentication, granting access to the account. Other setups may require tapping a pressure sensor, entering a PIN, or no additional action if verification is absent.

Following the inspection of WebAuthn message transcripts, we found that PayPal and Porkbun leak passkey credentials associated with a user account to clients before authentication. On the login page, submitting the username and email in an HTTP POST request triggers the service to send a response message containing a WebAuthn field labeled $allowCredentials$ [46], listing a set of credential identifiers linked to the account. For PayPal, this list also includes device descriptions (model, OS, and browser). The WebAuthn specification identifies this as a privacy issue and suggests countermeasures to limit such leakage [28].

(c) Passkey information visible to users. A key complexity of passkeys, unlike other authentication protocols like passwords, is that credential management is split among the service, client-side software, and passkey provider. Our assessments highlight the importance of ASIs to enable passkey management for users. Two example ASIs appear in Figure 4. We identify that ASIs provided by the service (service ASIs), and ASIs offered by the client (client ASIs) are both relevant for passkey management. We refer to both OS-level and browser-level ASIs as client ASIs. See Figure 10 in the Appendix for examples.

(c.1) Service ASIs. Google, Amazon, PayPal, and Target offer service ASIs (Figure 4) that allow users to manage their passkeys. Users can view previously registered passkeys and configure new ones. However, these ASIs vary in the information provided to users. Many services have dedicated ASIs for passkeys and other ASIs for listing devices with account access or session lists. These other ASIs tend to have more granular information about access. None of the six services in our study clearly link passkeys to specific devices or sessions.

Our analysis found inconsistencies in how passkeys are represented in ASIs. Target labels all passkeys as "Chrome 126 on Macintosh" or "Chrome 126 on" regardless of the passkey provider (see Figure 4). Furthermore, our experiments also revealed that Amazon's ASI labels security keys as "Other".

Motivated by the lack of clear language on such important tools to users, we also conducted experiments to examine if such ASIs might be reconfigured, such as via spoofing. Many passkey entries could be manipulated by very basic attacks including changing the user agent (UA) string (c.f., [42]) to alter the browser or OS model displayed, using a virtual private network (VPN) to appear in a different location, and the ability to modify a given passkey label.

(c.2) Client ASIs. Passkeys are also managed by the client device, which we examined using macOS (Sonoma v.14.5). The relevant ASI is located in the settings app under *Passwords* reflecting an effort by designers to treat passwords and passkeys similarly. Accessing this ASI requires re-authentication via biometric or local device password.

The MacOS-based ASI lists passkeys managed by the platform authenticator, alongside options including *Security Recommendations*, *Password Options*, and *Shared Groups*. We identified that clicking on a passkey entry reveals service URL, account username, passkey last modification date, passkey creation date, and a field allowing a password to be stored for that service. Users can edit the passkey account username and delete the passkey. Finally, there is an export option (AirDrop) to copy passkeys to other Apple devices.

Task	Adversarial Biometrics	Passkey Cloning	Adversarial Passkeys
Precondition	**(1-V)** Set up online account **(2-V)** Set up passkey on account	**(1-V)** Set up online account **(2-V)** Set up passkey on account	**(1-V)** Set up online account
Set up access	**(3-A)** Unlock victim's device **(4-A)** Add adversarial biometric	**(3-A)** Unlock victim's device **(4-A)** Reconfigure to spoof passkey label **(5-A)** Clone passkey to adversarial device	**(2-A)** Reconfigure to spoof user agent **(3-A)** Log in to victim's account **(4-A)** Register new passkey on account & logout
Unauthorized access	**(5-A)** Local login to victim's account **(6-A)** Use adversarial biometric to authenticate	**(6-A)** Remote login to victim's account **(7-A)** Use cloned passkey to authenticate	**(5-A)** Remote login to victim's account **(6-A)** Use adversarial passkey to authenticate
Evidence of access	**(7-V)** Check service ASIs for adversarial access **(8-V)** Check device ASIs for biometric manipulation	**(8-V)** Check service ASIs for adversarial access **(9-V)** Check device ASIs for passkey manipulation	**(7-V)** Check service ASIs for adversarial access **(8-V)** Check service ASIs for gaslighting
Revoke access	**(9-V)** Prevent access through biometrics	**(10-V)** Revoke cloned passkey	**(9-V)** Revoke adversarial passkey **(10-V)** Reset password and invalidate sessions
Check access	**(10-A)** Login to victim account on service **(11-A)** Use adversarial biometric to authenticate	**(11-A)** Remote login to victim account on service **(12-A)** Use cloned passkey to authenticate	**(11-A)** Remote login to victim's account on service **(12-A)** Use adversarial passkey to authenticate

Figure 5: Stepthrough protocols for testing unauthorized access for the adversarial biometrics, passkey cloning, and adversarial passkeys abuse vectors. For each step we indicate whether it is simulating a victim (e.g., 1-V) or abuser (e.g., 1-A) action. The latter two protocols also test the two gaslighting abuse vectors (via steps 4-A, 9-V and 2-A, 8-V, respectively).

(d) Passkey export and sharing. We tested Apple's export functionality by copying passkeys to another device (an iPhone 11 running iOS 17.5.1) and attempted to login with these passkeys. Our attempts were successful for five services (Google, Amazon, Paypal, Target, and Porkbun). The CVS passkey implementation, as mentioned earlier, relies on credential identifiers stored in the browser, so copying the credential to another device alone is insufficient[2] to authenticate. In addition to manual export, Apple iCloud keychain automatically syncs passkeys across Apple devices logged into the same iCloud account. We were unable to sync or share Apple passkeys with other non-Apple devices.

(e) Passkey revocation and deletion. Passkey management is divided between the client and service, offering two methods to prevent login via a passkey. Deletion involves removing the passkey secret from client storage using a client ASI. Revocation uses a service ASI to remove a registered passkey public key, so it can no longer verify a WebAuthn challenge.

Across all six services, once all passkey copies are deleted, passkey authentication no longer works (as expected). The services still list the passkey(s) as registered (they have not been revoked). Google, Amazon, Paypal, and Target all support revoking individual passkeys. However, only Google sends consistent email notifications upon revocation. While Paypal sends notifications, it only does so if the device is deemed 'untrusted', yet what consitutes a trusted device is unclear. Revoking a passkey on Google and Target does not immediately terminate active sessions authenticated by that passkey; detailed analysis of the time it takes to terminate the session upon revocation is shown in Figure 7. Amazon warns users that revoking a passkey does not delete it on clients, advising them to manually delete it from their cloud service account.

Neither CVS nor Porkbun provides service ASIs supporting revocation, meaning there is no way for users to revoke passkeys. Porkbun does offer a passkey toggle button in the *Account Security* ASI for enabling or disabling passkey use. However, disabling this toggle does not revoke a previously registered passkey, and, the passkey remains usable for authentication despite passkeys being "disabled".

[2]Copying the browser state does enable use of the copied passkey.

Whether FIDO2 should ensure server/client consistency with revocation and deletion linked remains an open question. Currently, services are not notified of deletion, and clients are not informed about revocation. We suspect this will confuse users, for example one might erroneously believe deleting one copy of a passkey suffices to secure an account.

5 Testing Abuse Vectors

Our threat models and functionality assessment enabled us to develop a set of hypothesized abuse vectors (see Figure 2 in Section 3). Then we developed a detailed set of five stepthrough protocols that test all the abuse vectors. Figure 5 details three stepthrough protocols for unauthorized access; these also test the two gaslighting abuse vectors. Figure 8 in Appendix C details the two other protocols.

For our stepthroughs we use four client-side platforms (iOS, MacOS, Android, Windows) and considered 15 passkey-supporting services (Adobe, Amazon, Apple, Ebay, Github, Google, Intuit, LinkedIn, Microsoft, Roblox, Samsung, TikTok, Twitter, WhatsApp, Yahoo) from the top 200 Tranco list [47]. We focused on web versions of services, but for TikTok, Twitter, and WhatsApp we used the iOS apps.

We could not perform our stepthroughs for Apple, WhatsApp, and Yahoo. Apple supports passkeys, but only provisions them automatically; users cannot manually generate passkeys. WhatsApp allowed passkey creation but not authentication, while Yahoo only presented a passkey configuration option sporadically. Thus we performed one or more stepthroughs on each of twelve services.

Appendices A and B provide more details about service selection and experimental setups.

5.1 Adversarial Biometrics

In interpersonal threat models, attackers can gain access to victim devices through physical proximity and shared credentials [19, 23, 26, 48]. Our first abuse vector capitalizes on this, with an attacker adding their biometrics to a device to provide persistent access even if the device password changes—an adversarial method also documented in prior research [26].

In our context, an adversarial biometric also grants ongoing access to all passkey-protected accounts on the device. The simplicity of this attack contrasts sharply with the difficulty of remediating it, which typically involves deleting and re-registering all biometrics. Sometimes, this is made worse by a lack of any ASIs to help identify malicious access. For each client, we registered passkeys on services that have passkey support for the given client configuration. After registering, we logged out of each of these services. We then added an adversarial fingerprint to the unlocked device, and attempted to log into the services. See Figure 5 detailed protocol steps.

Executing the stepthrough protocol. We ran our adversarial biometric protocol for 10 services and three client configurations: MacOS, Windows, and Android. Two services, TikTok and Twitter only support passkeys on iOS, so we skip them here. This meant we performed a total of 30 stepthroughs to test the adversarial biometric abuse vector.

Adding an adversarial biometric. On MacOS, an attacker adds their fingerprint using the device's settings app. On Android, biometrics are configured under *Pixel Imprint* in Android's *Security* settings. On Windows, biometrics are added on the sign-in options page in the settings app, under "*Fingerprint recognition (Windows Hello)*". All three platforms require an adversary to input the victim's device credentials to add a new biometric.

All of these OSes limit the number of biometric configurations (e.g., fingerprints) on a single device or on a single account on the device. MacOS allows a maximum of three fingerprints per account, Android allows up to five fingerprints on the device, and on Windows we were able to configure six fingerprints[3] on a single device account. In our stepthroughs, the victim did not have more than one biometric setup, so this step always succeeded, but if the victim has reached the maximum biometrics capacity, then the attacker would have to replace one, possibly hindering covertness.

Unlocking with adversarial biometrics. Adding adversarial biometrics succeeded for all services, demonstrating that *any* registered biometric suffices to unlock *any* passkey. By design, FIDO2 does not support binding passkey operations to specific user biometrics. This means that the user biometrically authenticating with a passkey is not verified as the user that registered it. In theory, systems could somehow bind FIDO2 credentials to specific biometrics, but this may hamper usability (e.g., one could not use any of their fingerprints to perform user verification). We conjecture that users may be confused by these subtleties, assuming that using their fingerprint ensures that only they can perform service logins.

[3]We did not confirm a maximum, though online sources suggest it varies based on hardware model.

Detecting adversarial access and manipulation. We find that detecting adversarial biometrics is difficult or impossible using available ASIs (if any). On both MacOS and Android, fingerprints are labelled and enumerated as "*Finger 1*", "*Finger 2*", etc. On Windows 10, the UI does not show any information about the number or nature of registered fingerprints. Additionally, none of the OSes inform users of any changes to their biometric configurations through notifications.

For most services, adversarial accesses are only distinguishable from victim accesses by the day and time a passkey was used. Figure 6 lists the service ASIs that we discovered through our stepthrough protocols. For completeness, we also list services explored via our functionality assessments. We indicate which information attributes are available on each interface to help users understand access, and whether they are fully controllable by reconfiguration (i.e., spoofable). Most information attributes (browser, OS, time, etc.) and their spoofability were also considered in prior work [42].

Passkey labels are newly considered by our work here: these are short text descriptions of individual passkeys listed on a service ASI. The nature of these labels vary widely, sometimes seeming to indicate the passkey storage location (e.g., iCloud keychain, Google Account), sometimes labels are generic names assigned by the service (e.g., "Passkey 1"), and in some cases are directly user-editable on the ASI interface (making spoofing trivial).

We conjecture that users will find it exceedingly difficult to use service ASIs to distinguish adversarial accesses from their own (which, after all, are from the same device), particularly in the presence of spoofing.

Preventing ongoing adversarial access. To prevent ongoing access, the analyst attempted to remove the adversarial biometric. On MacOS and Android, the interface for removing a biometric is the same as for configuration. Given the lack of visual indicators observed when detecting adversarial access, users must remove all biometrics on the device and re-register legitimate ones. On Windows, deletion of individual fingerprints is not supported. In order to recover from a configured adversarial biometric, the victim would have to first deactivate the use of biometric authentication at "*Remove this sign-in option*" in the device account settings. This removes all registered biometrics. They would then either reactivate it and configure a new fingerprint, or just deactivate it entirely and rely on pin verification.

None of the above guarantees that access to accounts is revoked, as the attacker could have used the access from this attack to setup other forms of remote access. Thus, the user may need to go through securing all of their accounts as well.

Service	ASI	Passkey Label	Device Model	OS	Browser	Location/IP Address	Date Added/Removed	Date Used
Adobe	Passkeys	○	○	○	○			
	Active Sessions			○	○	○		
Amazon	Passkey	●					●	
	Account data access attempt*		○	○		○		
	Set up a passkey*							
Apple	Devices		●	●				
CVS	None							
Ebay	Sign-in Activity			●	○	○		
	New Device*			●	○	○		
Github	Passkeys	○					●	●
	Security log	○	○	○	○	○	●	●
	Sessions			○	○	○	●	
	Passkey added/removed*				○	○		
Google	Passkeys & security keys	○	○		○	○	●	●
	Recent Security Activity		○			○	●	●
	Security alert (added/deleted)*							
Intuit	Where you're logged in		●	○	○	●	●	
	Passkeys	●					●	●
	Passkey added/deleted	●					●	
LinkedIn	Passkeys	●		○	○	○	●	
	Where you're signed in			○	○	○	●	
	Here's your pin			○	○	○		
Microsoft	Security	○					●	●
	Sign-in activity			○	○	○	●	
	Account info added/deleted*							
Paypal	Passkey	○		○	○		●	
	Manage your logins		○	○	○			●
	Changed login settings*						●	
	Passkey removed*						●	
Porkbun	Recent login history			○	○	○	●	
	Login using WebAuthn key*					○		
Roblox	Manage your passkeys	○						
	Where you're logged in			○	○	○		●
Samsung	Passkeys	●					●	●
	Recent Account Activity				●	○		●
	New Sign*				●	○		●
Target	Passkeys	○	○		○		●	
	Where you're signed in		○			○	●	
	Added passkeys*		○	○				
TikTok	Manage devices†				○		●	
Twitter	Sessions†						●	
WhatsApp	Passkeys†	●						
Yahoo	Recent activity	●					●	
	Connected devices			○	○	○	●	●
	device auth de/activated*							

Figure 6: Summary of services considered in this paper. Boldfaced services are the 12 for which we performed one or more stepthrough protocol tests. We list individual ASIs and the information attributes that are non-spoofable (●), spoofable (○), or not displayed (blank). A "∗" indicates an email notification and † denotes a native iOS app UI; all others are web UIs.

5.2 Passkey Cloning

Passkey cloning involves an attacker using temporary access to a victim's device to copy a passkey registered with a service. The attacker can then use the cloned passkey on a different device, upgrading one-time access to an unlocked device to ongoing remote login to the victim's account.

There are two ways to clone a passkey, both of which exploit credential portability functionality. An attacker can either export the passkey (iOS, MacOS) or use a synchronization service to have it added to their device (iCloud Keychain, Apple Password Groups, Google Password Manager).

The victim device is first set up with a registered passkey on services. Then, the analyst attempts to reconfigure client ASIs to spoof passkey labels before cloning each passkey from the victim's device to an adversarial device. We then checked each client ASI for evidence of the spoofing and cloning. Using the adversarial device, we then attempted to log into each web service using the cloned passkeys, before examining the service ASIs for evidence of adversarial access. Finally, we tried to revoke any passkeys that were successfully cloned. The detailed protocols steps appear in Figure 5.

Executing the stepthrough protocol. We ran our passkey cloning stepthrough protocol for all 12 services in the Apple ecosystem and for nine services in the Google ecosystem (Google sign-in is required for sync, and TikTok and Twitter only offer passkey support in iOS). We used multiple client configurations for cloning as described in Appendix B. Overall, we performed 21 stepthroughs to test passkey cloning. We present our results for the Apple ecosystem here; the Google ecosystem results appear in Appendix D.

Cloning in the Apple ecosystem. In the Apple setup the analyst discovered three distinct routes for passkey cloning.

The first is AirDrop export on MacOS and iOS [49]. To export via AirDrop, the abuser and victim must be in each others' trusted contacts list and AirDrop must be enabled on both devices. The attacker can add themselves to the contacts list on the victim's unlocked device without any re-authentication. Then, the attacker (while on the victim device) navigates to the passkey ASI on the *Passwords* panel in the Settings app. Opening this ASI requires the attacker to re-authenticate. After this, the attacker expands on the entry for the passkey to be exported and edits the passkey to spoof the username. Following this, they click the export icon on this same passkey (which again asks for re-authentication). At this stage, the attacker is presented with an AirDrop dialogue to export their desired passkey to the adversarial device. Once export is completed the passkey ASI on the victim device displayed a sentence in very small font, saying *"last shared with AirDrop [Date_of_Sharing]"*.

The second cloning route leverages Apple's shared password groups feature [50]. This is enabled whenever Apple's *Passwords and Keychain* are synced using iCloud in the Apple

ID settings window; this is the default setting assuming the victim is using iCloud. The attacker proceeds by first creating a new shared password group on the adversarial device and adds the victim's Apple ID account to that group. Then, they use temporary physical access to the victim's device to accept the invite. Finally, they add the victim's passkeys to the shared group. After this, a group icon is visible on each passkey's entry in the passkey client ASI.

The third cloning route is via logging into Apple iCloud on the adversarial device, given the attacker knows the victim's iCloud credentials. Signing into iCloud then sends 2FA challenges to the victim device, but we assume the attacker has access to this already. When the attacker logs into the victim's iCloud account, the adversarial device will automatically sync by copying all of the victim's passkeys locally. Once that is complete, the attacker can log out of iCloud—a local copy of the victim's credentials remains on the adversarial device if they choose *"Keep a copy of Keychain data on this device"* on the iCloud sign out prompt (Figure 9 in Appendix C).

Authenticating using cloned passkeys. All three passkey cloning mechanisms allowed analysts to log into the twelve services that had functioning passkey implementations. Cloning via AirDrop sometimes failed, however analysts were able to authenticate when the export was successful.

Detecting authentication via cloned passkeys. Unlike the adversarial biometric abuse vector, in which the same device is used to access the account, here a distinct device is used. Evidence of access via passkey cloning varies based on service-provided ASIs and notifications. On Amazon, Ebay, TikTok, and Twitter there are no visible differences on *any* of the ASIs indicating that an adversarial login via the cloned passkey took place. For Adobe, Intuit, LinkedIn, Roblox, and Samsung, we find that cloning is possibly detectable as the victim can infer unrecognized logins on non-passkey ASIs, but these do not include information that allows identifying the time of login or (non-spoofable) device details (Figure 6).

On Github, Google, and Microsoft we find that passkey cloning is plausibly detectable as the three services include a "*Last used*" indicator on their *Security*, *Passkeys and security keys*, *Password and authentication* ASIs, respectively, as per Figure 6. This indicator is included for each passkey entry indicating when the passkey was last used for sign-in, so a victim could potentially spot an unexplained access time. Google additionally includes the device and location information of last use. Of course, prior work [42] has explored how tricky these ASIs are to confirm adversarial access, and we expect many users will have difficulty doing so.

Detecting passkey cloning. Detecting cloning via client ASIs can be difficult—each route for performing passkey cloning gives rise to distinct types of warnings. A victim can in theory detect cloning via: the previously mentioned last shared text displayed after AirDrop; the new presence of group icon displayed on a passkey after being added to a shared passwords group; a warning that displays to a victim that a credential was previously shared to a password group (Figure 9 in Appendix C); and iCloud sign-in notifications sent to the victim's device. We note that for AirDrop, the textual description will show even if the export fails. We surmise that many users facing interpersonal abuse may not understand the importance of these indicators. Additionally, it is impossible to detect the reconfiguration of passkey labels on Apple's *Passwords* ASI.

If an attacker enrolls their own device in the victim's iCloud account then signs out, they can retain a previously-synchronized copy of the victim's passkeys without any notice on the victim's device beyond the initial enrollment. If the attacker does not sign out, then their device appears in a device list enumerating trusted devices on the victim's iCloud account, but no additional indicators describe which devices have a copy of the passkeys and which do not.

Revoking cloned passkeys. For passkeys cloned via export, deleting a passkey on the victim device does not disable the copy on the adversarial device from working to log into a service. Similarly, for passkeys cloned via iCloud synchronization, there is no way for the victim to delete those copies even if they log out the adversarial device. In these cases, the user must instead revoke the passkey at the service.

Figure 7 lists relevant service ASI management features for 17 of the services. (Apple and CVS do not have passkey service ASIs.) An entry in the re-authentication column indicates that during testing the analyst had to re-authenticate somehow to access the interface or to create a passkey. The second column indicates whether changing a password automatically revokes registered passkeys in our testing—only Ebay does so and in fact that is, unintuitively, the *only* way to revoke a cloned passkey. Password change flows did not include reminders suggesting a user revoke passkeys.

Only four services included options to disable use of passkeys on the account, whereas most services allow revoking individual passkeys. Doing so most often did not immediately prevent ongoing adversarial access, i.e., the passkey-authenticated session on the adversary's device could still access the account. We performed additional tests to check if a passkey-authenticated session on the adversary's device was invalidated by 5, 15, 30, or 60 minutes after revoking the passkey. The results are shown in the fourth column of Figure 7. For several services (Adobe, Github, LinkedIn, Samsung) sessions remained valid even an hour later (denoted by '>60')—we do not know when these sessions are ended. For comparison, we repeated this experiment but instead starting with a password change—the results in many cases are different with invalidation happening faster. Note that we only performed these tests twice per service; the exact timing/behavior might fluctuate for a variety of reasons.

We are unsure what explains this inconsistency in session management, but view it as a potential vulnerability. Combined with lack of re-authentication (Adobe, Github, Intuit, Microsoft, Roblox, Target), an adversary might be able to

Service	Re-authentication	PW change → passkeys revoked	Disable passkeys	Revoke individual passkeys	Passkey revoke → session end	PW reset → session end	End particular session	End all sessions
Adobe				◇	*>60*	*5*	◇	◇
Amazon	◇			◇	*5*	*15*		◇
Ebay		◇				*5*		
Github				◇	*>60*	*5*	◇	
Google	◇			◇	*30*	*5*	◇	
Intuit				◇	15	30		
LinkedIn	◇		◇	◇	*>60*	*5*	◇	◇
Microsoft				◇	*30*	*5*		
PayPal	◇			◇	*30*	*5*	◇	
Porkbun			◇			5		◇
Roblox				◇	*60*	*5*	◇	◇
Samsung	◇			◇	*>60*	*60*		
Target				◇	*30*	*15*	◇	◇
TikTok						*5*	◇	
Twitter			◇			*5*	◇	◇
WhatsApp	–	n/a		◇	–	n/a	◇	
Yahoo	–	–	◇	–	–	*5*	◇	

Figure 7: Available ASI management options on 17 services (Apple and CVS do not have service ASIs for passkeys). Numbers indicate the maximum amount of time (in minutes) for which sessions authenticated by passkey (resp. password) remain valid following passkey (password) revocation. We could not test functionality marked with a dash (–) due to service limitations.

recover access by changing security configurations within the time period their session remains active. A victim may know to use features for explicitly ending sessions, but these were not available on Ebay, Intuit, Microsoft, Samsung.

We could not determine any way to prevent ongoing access with a cloned passkey on TikTok. While we didn't perform stepthrough tests for CVS and Porkbun, the same situation seems true based on our functionality assessment.

For cloning via Apple's shared passwords group feature, the account that adds a credential to the group is the only one authorized to remove it. In our stepthrough, this account is the victim's, and they can thus remove the passkey from the group—the adversarial device loses access to the passkey. If instead an abuser compels the victim to initially setup a passkey for a service on a device logged in using the abuser's Apple ID, and then shares that passkey to the victim's Apple ID, there would be no way for the victim to delete the attacker's copy of the passkey using the feature. The victim would instead have to use service ASIs to revoke the passkey.

5.3 Adversarial Passkeys

Next, we examine adding an attacker's passkey to a victim's account, referred to as an adversarial passkey.

In our stepthrough, we first reconfigure the adversary's client to spoof ASI entries, then log into each victim account from the adversarial device, and then register a new passkey as the primary authentication mechanism. Then, we check the victim's device for indicators (e.g., notifications) of any adversarial access or passkeys. Next, we would log onto each web service from the adversarial device using the adversarial passkeys. We then attempt to secure each victim account by changing the password, revoking adversarial passkeys, or disabling passkey support. The stepthrough protocol is summarized in Figure 5).

Executing the stepthrough protocol. We ran our adversarial passkey protocol for all 12 services. We performed a single stepthrough for each service. Details about the client setup appear in Appendix A.

Registering adversarial passkeys. We find that adding a remote adversarial passkey is successful on Adobe, Amazon, Github, Google, Ebay, Intuit, LinkedIn, Microsoft, Roblox, and Samsung. For two services (TikTok, Twitter) only a single active passkey is permitted on the account, foiling the attacker[4] if the victim already registered a passkey.

On Adobe, Ebay, Github, Intuit, Microsoft, and Roblox adding a passkey when logged in does not require any re-authentication (Figure 7). Others require varying levels of re-authentication before passkey registration: Amazon sends users a one-time-password (OTP) to their email, Google requires re-entering the victim's password, and LinkedIn requires both entering the victim's password and a OTP to email. This adds friction for the attacker since they would need to be able to pass these challenges.

The analyst also realized that a variant of the attack works with one-time access to the victim's unlocked device and assuming an active session on the target service. The attacker again performs a passkey registration on the victim's account, but instead of using the local platform authenticator, chooses to store the passkey on a roaming authenticator such as a security key, or the attacker's mobile phone (often possible by just scanning a QR code during passkey registration c.f., [51]).

Detecting adversarial access. Adding a passkey to the victim's account triggers an email notification for Amazon, Github, Google, Intuit, Microsoft, and Roblox. Besides Ebay—which does not have a passkey ASI—the new passkey is listed on service passkey ASIs for all services for which the attack succeeded (Figure 6). Determining which are the adversarial passkeys is more difficult due to spoofing.

Authenticating with adversarial passkeys. Login with the adversarial passkey worked as expected for all of the services

[4]One could revoke the victim's passkey, but this bars login with it, making it less useful for ongoing, covert monitoring.

(beyond those that only supported a single passkey). The victim visibility into accesses via service ASIs is the same as in the cloned passkey case.

Revoking adversarial passkeys. Passkey revocation works for all ten services for which the attack succeeds, albeit using different mechanisms (Ebay via password reset; the rest via service ASIs). Only Amazon invalidates the passkey-authenticated adversarial session immediately (within 5 minutes) upon revocation; as described before other services take much longer to terminate adversarial sessions. This is problematic as the abuser can possibly use the still active session to set up another adversarial passkey, or perform a fuller account takeover (revoke legitimate passkeys, reset the password, etc.).

5.4 Passkey Deletion

The passkey deletion abuse vector has an abuser leverage physical access to a victim's device to delete stored passkey credentials, thereby denying a victim access to their accounts via them. A detailed stepthrough protocol for deletion is shown in Figure 8 in the appendix.

Executing the stepthrough protocol. We ran the protocol for all 12 services in the Apple ecosystem and nine services in the Google ecosystem using a single client configuration (MacOS). This meant we performed a total of 21 stepthroughs to test the passkey deletion abuse vector. We discuss the results for the Google ecosystem in Appendix D.

Results. The analyst found that for passkeys stored in Apple's iCloud Keychain, local deletion can be done using the *Passwords* ASI within the settings app. The adversary may have to re-authenticate to the device to access this ASI. They can then expand on each credential listed on the ASI to take actions including deleting it. Upon doing so, the passkey is moved to the *Recently deleted folder* on the same *Passwords* ASI. This prevents the passkey from use; the adversary can delete it from the *Recently deleted folder* to render it unrecoverable.

Next, the analyst logged into each of the twelve services on the same device to simulate a victim logging into their accounts. For all services, authentication was denied using the (now deleted) passkey, as expected. Most services in our client configuration show a QR code to the user when requesting to log in via passkey (i.e., it does not detect a passkey on the device and instead offers the user the option to use a passkey from another device).

Users are not informed via any platform or email notifications about the credential deletion, but in theory could discover it by seeing fewer-than-expected passkeys on a client ASI list or because login fails. For all services studied, a password is required on the account before setting up passkeys, so absent further adversarial actions, a victim can recover from the denial of service by logging in via password and creating a new passkey to replace the deleted one.

5.5 Passkey Revocation

The passkey revocation abuse vector has an attacker leveraging remote access to a victim's service account (e.g., by learning the password) to deny the victim access via passkeys. The attacker strictly relies on service ASIs to perform the revocation; a detailed stepthrough protocol is available in Figure 8 in the appendix.

Executing the stepthrough protocol. We ran our protocol once for each of 12 services using MacOS and iOS devices to simulate the victim and abuser, respectively.

Results. Adversarial passkey revocation succeeded for eleven services (see Figure 6). For Ebay the only way to successfully revoke is through password reset. TikTok does not have any ASIs that enable revocation.

On all services, login via a revoked credential fails. Further, we find that none of the services provide an in-app notification for passkey revocation, and only five out of the eleven services inform users about passkey revocation through email notifications. The design of these notifications varies across services. Information attributes on these notifications and their vulnerability to reconfiguration are shown in Figure 6.

For services that lack email notifications, a victim would have to refer to non-passkey ASIs on the service to find information about logins and account activity (e.g., session logs, which were available on all eleven services).

5.6 Gaslighting

The spoofability [42] of service ASIs can enable abusers to gaslight their victims—emotional manipulation or harm enabled by reconfiguring information on security-relevant ASIs. The abuser might do this to mislead the victim into believing that they have control over their credentials or accounts. We tested gaslighting via our stepthrough protocols for passkey cloning and adversarial passkeys (Figure 5).

Results. Gaslighting can be achieved when the adversary controls information shown to the victim in security-relevant ASIs. The analyst was able to do so for service ASIs by reconfiguring the client device to: (1) modify the HTTP user agent string (which affects device model, browser, or operating system on service ASIs); (2) by using a VPN (to change location shown on service ASIs); and (3) by changing passkey labels in client ASIs that end up displayed on service ASIs. Figure 6 summarizes spoofability of service ASIs.

As for client ASIs, on MacOS and iOS, for the *Passwords* ASI each passkey entry has a username associated with it; this is set by the service initially, but a user with local device access can change it. Google's password manager similarly allows editing of the username associated with passkeys. Changing a username does not impact use of the passkey. Windows does not allow modifying usernames.

6 Discussion

Implementation bugs versus protocol weaknesses. Like prior work on FIDO2 and WebAuthn [10–13], our work surfaced a variety of implementation bugs. These included a lack of functionality for revocation of passkeys (CVS, Ebay, TikTok), relying on ephemeral client-side state to use registered passkeys (CVS, Yahoo), leaking credential identifiers to unauthenticated clients (Paypal, Porkbun), creating stale credentials upon registration failure (Porkbun), inconsistent offering of passkey creation (Yahoo), and improperly formatted passkey labels (Target, Amazon). This suggests a lack of maturity and testing in current deployments.

More fundamental was problematic patterns in how services implement passkeys and communicate it to users. No services list which passkey was used to authenticate a session (or whether another authentication mechanism was used); this hinders compromise detection. Moreover, both service and client ASIs varied in design across services and client configurations, hindering usability (reinforcing results offered by [4, 9]). Many services allow customization of passkey labels, making labels exploitable for adversarial behavior. We believe credential revocation will confuse users: only Ebay revoked passkeys upon password reset, and no services prompted users about revoking passkeys during password reset flows. Moreover, even when revoking passkeys, sessions that were authenticated are not invalidated immediately, leaving a window of opportunity for interpersonal adversaries to retain access or reassert control over the account.

Finally, our results raise questions about seemingly inherent issues with passkey protocols as currently standardized. For example, the lack of binding between OS-controlled biometrics and passkeys gives rise to abuse vectors such as adversarial passkeys. One potential approach to rectify this would be to extend FIDO2 protocols to allow binding each passkey to a specific set of biometrics, but this would seem to hinder usability. Instead, we suggest improving notifications regarding biometric and passkey configuration changes, which might require protocol changes as both the client and service ASIs should be informed about configuration changes including new biometrics. Similarly, the way passkey sharing or exporting works currently needs to be revisited in light of abuse.

From password to passkey problems. Passkeys have been advertised as a fix-all to the problems with passwords [3]. However, our findings identify weaknesses that are passkey-specific and do not apply to other authentication mechanisms like passwords. For example, unlike passwords, an account can have multiple active passkeys, which enables adversarial passkeys. A major concern is the conflation of passkeys with passwords in both implementations and documentation. Many passkey ASIs are found under password management panels, which may confuse developers and users due to differences between the two. For instance, passkeys can be added without revoking old ones, unlike passwords. We need better ways to communicate these differences, perhaps emphasizing passkeys as “keys” that can be copied.

In terms of near-term mitigations, to prevent passkey cloning, developers may look to Google’s enforcement of stricter controls on passkey export (disallowing it entirely), and how logging out of a device removes access to synchronized passkeys. To address adversarial biometrics, OSes could implement an ASI with an immutable log to track changes to security-critical settings (e.g., biometric additions) and pair it with account notifications at the OS level. Finally, adversarial passkeys may be mitigated by adding re-authentication steps and notifying users of changes to authentication configuration. Many of these mitigations will also help with the denial-of-service abuse vectors. As noted by Daffalla et al. [42] spoofing is difficult to mitigate due to competing privacy concerns, though recent work suggests potential cryptographic approaches [52]. Finally, we anticipate the need for further research in designing and evaluating new wizards to help users secure their passkey-enabled accounts, with a particular focus on addressing the needs of at-risk users.

On abusability analyses. Our new six-stage framework for abusability analysis provides a structured way for researchers and practitioners to test for potential ways in which products will or can be abused. Future work will be needed to test the applicability of our framework within other contexts, develop tooling to assist analysts applying the framework, perform user studies to gauge the framework’s usability, and potentially suggest refinements.

7 Conclusion

We propose a new framework for abusability analysis and apply it to the quickly evolving passkey ecosystem. We investigated in total 19 services and four passkey providers. We identified a number of abuse vectors and showed, via stepthroughs, how adversaries can enact them to cause harms such as unauthorized access, denial of service, and gaslighting. While some abuse may not be preventable completely in our threat models, our analysis suggest important mitigations such as better notifications, clearer management mechanisms and documentation, and more. As such, our work provides new directions for how passkey deployments can be improved to help users, particularly those targeted by interpersonal abuse.

Acknowledgments

We thank the reviewers and the shepherd of our paper for their insightful comments and suggestions. The research is supported in part by the Baldwin Wisconsin Idea Grant, NSF award #2339679, and the Google Cyber NYC Program.

Ethical Considerations

This research did not involve human subjects or personal data; only dedicated research accounts were used. We employed lab devices solely for analyzing passkey and WebAuthn capabilities and adhered to each service's terms and conditions. Synthetic names, emails, or phone numbers were used, and we avoided imposing unusual resource burdens on the services. When passkey issues were identified, we refrained from contacting customer service.

Our findings reveal vulnerabilities in current passkey deployments. We are cognizant of the risks of adversarial readers of this work learning low-level details of these attacks and using them to inflict harm in interpersonal threat models. To make progress on discovering and mitigating such abuse vectors before they can be exploited, we believe it important to document our procedures and work with companies to add mitigations. In the cases of the eight services with implementation bugs (see Section 6), we contacted the relevant teams at each service. We provided a copy of our manuscript, a detailed description of how to reproduce our findings. For the remaining services, we focused our report on demonstrating the inherent weaknesses of common implementations of the passkey protocol. We offered all services the ability to meet with the research team to answer any questions and suggestions for near- and longer-term mitigations.

Open Science Policy

We offer detailed, separate artifacts describing the results of our research. This includes copies of the stepthrough procedures, detailed notes about identifying abuse vectors, and results from our stepthrough protocol testing. All data pertaining to this work is available online at https://doi.org/10.5281/zenodo.14745290.

References

[1] *FIDO Alliance Overview - Changing the Nature of Authentication*. https://fidoalliance.org/overview/. (Accessed on 08/10/2024).

[2] Heather Adkins. *Passkeys, Cross-Account Protection and new ways we're protecting your accounts*. https://blog.google/technology/safety-security/google-passkeys-update-april-2024/. 2024.

[3] Jenny Blessing et al. *SoK: Web Authentication in the Age of End-to-End Encryption*. arXiv:2406.18226 [cs]. June 2024. DOI: 10.48550/arXiv.2406.18226. URL: http://arxiv.org/abs/2406.18226 (visited on 09/03/2024).

[4] Kentrell Owens et al. "User perceptions of the usability and security of smartphones as {FIDO2} roaming authenticators". In: *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*. 2021, pp. 57–76.

[5] Sanchari Das, Andrew Dingman, and L Jean Camp. "Why Johnny doesn't use two factor a two-phase usability study of the FIDO U2F security key". In: *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22*. Springer. 2018, pp. 160–179.

[6] Wataru Oogami et al. "Observation study on usability challenges for fingerprint authentication using WebAuthn-enabled android smartphones". In: *Age* 20 (2020), p. 29.

[7] Sanam Ghorbani Lyastani et al. "Is FIDO2 the kingslayer of user authentication? A comparative usability study of FIDO2 passwordless authentication". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 268–285.

[8] Joshua Reynolds et al. "A tale of two studies: The best and worst of yubikey usability". In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 872–888.

[9] Stéphane Ciolino, Simon Parkin, and Paul Dunphy. "Of two minds about {Two-Factor}: Understanding everyday {FIDO}{U2F} usability through device comparison and experience sampling". In: *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*. 2019, pp. 339–356.

[10] Manuel Barbosa et al. "Provable security analysis of FIDO2". In: *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part III 41*. Springer. 2021, pp. 125–156.

[11] Nina Bindel, Cas Cremers, and Mang Zhao. "FIDO2, CTAP 2.1, and WebAuthn 2: Provable security and post-quantum instantiation". In: *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2023, pp. 1471–1490.

[12] Jingjing Guan et al. "A formal analysis of the FIDO2 protocols". In: *European Symposium on Research in Computer Security*. Springer. 2022, pp. 3–21.

[13] Charlie Jacomme and Steve Kremer. "An extensive formal analysis of multi-factor authentication protocols". In: *ACM Transactions on Privacy and Security (TOPS)* 24.2 (2021), pp. 1–34.

[14] Arup Kumar Ghosh et al. "Safety vs. surveillance: what children have to say about mobile apps for parental control". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 2018, pp. 1–14.

[15] Priya C Kumar et al. "Privacy and security considerations for digital technology use in elementary schools". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 2019, pp. 1–13.

[16] Pamela Wisniewski et al. "Parental control vs. teen self-regulation: Is there a middle ground for mobile online safety?" In: *Proceedings of the 2017 ACM conference on computer supported cooperative work and social computing*. 2017, pp. 51–69.

[17] Arup Kumar Ghosh, Charles E Hughes, and Pamela J Wisniewski. "Circle of trust: a new approach to mobile online safety for families". In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 2020, pp. 1–14.

[18] Jordan Hayes et al. "Cooperative privacy and security: Learning from people with visual impairments and their allies". In: *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*. 2019, pp. 1–20.

[19] Diana Freed et al. ""A Stalker's Paradise" How Intimate Partner Abusers Exploit Technology". In: *Proceedings of the 2018 CHI conference on human factors in computing systems*. 2018, pp. 1–13.

[20] Tara Matthews et al. "Stories from survivors: Privacy & security practices when coping with intimate partner abuse". In: *Proceedings of the 2017 CHI conference on human factors in computing systems*. 2017.

[21] Christine Chen, Nicola Dell, and Franziska Roesner. "Computer security and privacy in the interactions between victim service providers and human trafficking survivors". In: *28th USENIX Security Symposium (USENIX Security 19)*. 2019, pp. 89–104.

[22] Jodi Halpern Clara Berridge and Karen Levy. "Cameras on beds: The ethics of surveillance in nursing home rooms". In: *AJOB Empirical Bioethics* 10.1 (2019). PMID: 30794112, pp. 55–62. DOI: 10.1080/23294515.2019.1568320. eprint: https://doi.org/10.1080/23294515.2019.1568320. URL: https://doi.org/10.1080/23294515.2019.1568320.

[23] Nora McDonald et al. "Realizing choice: Online safeguards for couples adapting to cognitive challenges". In: *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. 2020, pp. 99–110.

[24] Noel Warford et al. "SoK: A framework for unifying at-risk user research". In: *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2022, pp. 2344–2360.

[25] Rosanna Bellini et al. "SoK: Safer Digital-Safety Research Involving At-Risk Users". In: *arXiv preprint arXiv:2309.00735* (2023).

[26] Rosanna Bellini et al. "The {Digital-Safety} Risks of Financial Technologies for Survivors of Intimate Partner Violence". In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 87–104.

[27] *FIDO Alliance - Open Authentication Standards More Secure than Passwords*. URL: https://fidoalliance.org/.

[28] *Web Authentication: An API for accessing Public Key Credentials - Level 2*. https://www.w3.org/TR/webauthn-2/#sctn-credential-id-privacy-leak. (Accessed on 08/08/2024).

[29] *FIDO2: Web Authentication (WebAuthn) - FIDO Alliance*. https://fidoalliance.org/fido2-2/fido2-web-authentication-webauthn/. (Accessed on 08/10/2024).

[30] 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*. 2023, pp. 2381–2395.

[31] Mohammed Jubur et al. "Bypassing push-based second factor and passwordless authentication with human-indistinguishable notifications". In: *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 2021, pp. 447–461.

[32] Enis Ulqinaku et al. "Is real-time phishing eliminated with {FIDO}? social engineering downgrade attacks against {FIDO} protocols". In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 3811–3828.

[33] Leon Würsching et al. "FIDO2 the Rescue? Platform vs. Roaming Authentication on Smartphones". In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. CHI '23. Hamburg, Germany: Association for Computing Machinery, 2023. ISBN: 9781450394215. DOI: 10.1145/3544548.3580993. URL: https://doi.org/10.1145/3544548.3580993.

[34] *How FIDO Addresses a Full Range of Use Cases*. URL: https://fidoalliance.org/wp-content/uploads/2022/03/How-FIDO-Addresses-a-Full-Range-of-Use-Cases-March24.pdf.

[35] Kurt Thomas et al. "SoK: Hate, Harassment, and the Changing Landscape of Online Abuse". In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021. DOI: 10.1109/SP40001.2021.00028.

[36] Celine Latulipe, Ronnie Dsouza, and Murray Cumbers. "Unofficial Proxies: How Close Others Help Older Adults with Banking". en. In: *CHI Conference on Human Factors in Computing Systems*. 2022. ISBN: 978-1-4503-9157-3. DOI: 10.1145/3491102.3501845. URL: https://dl.acm.org/doi/10.1145/3491102.3501845 (visited on 08/04/2023).

[37] Diana Freed et al. "Digital technologies and intimate partner violence: A qualitative analysis with multiple stakeholders". In: *Proceedings of the ACM on human-computer interaction* 1.CSCW (2017), pp. 1–22.

[38] Julia Slupska and Leonie Maria Tanczer. "Threat modeling intimate partner violence: Tech abuse as a cybersecurity challenge in the internet of things". In: *The Emerald international handbook of technology-facilitated violence and abuse*. Emerald Publishing Limited, 2021, pp. 663–688.

[39] Ashkan Soltani. "Abusability testing: Considering the ways your technology might be used for harm". In: *Enigma 2019 (Enigma 2019)*. 2019.

[40] Philip Moons, Eva Goossens, and David R Thompson. "Rapid reviews: the pros and cons of an accelerated review process". In: *European Journal of Cardiovascular Nursing* 20.5 (2021), pp. 515–519.

[41] Zachary Munn et al. "What kind of systematic review should I conduct? A proposed typology and guidance for systematic reviewers in the medical and health sciences". In: *BMC medical research methodology* 18 (2018), pp. 1–9.

[42] Alaa Daffalla et al. "Account Security Interfaces: Important, Unintuitive, and Untrustworthy". In: *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 3601–3618. ISBN: 978-1-939133-37-3. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/daffalla.

[43] Lucy Simko et al. "Computer security and privacy for refugees in the United States". In: *2018 IEEE symposium on security and privacy (SP)*. IEEE. 2018, pp. 409–423.

[44] Thomas Mahatody, Mouldi Sagar, and Christophe Kolski. "State of the Art on the Cognitive Walkthrough Method, Its Variants and Evolutions". In: *Int. J. Hum. Comput. Interaction* 26 (July 2010), pp. 741–785. DOI: 10.1080/10447311003781409.

[45] *Web Authentication: An API for accessing Public Key Credentials - Level 2*. https://www.w3.org/TR/webauthn-2/#credential-id. (Accessed on 08/29/2024).

[46] *Web Authentication: An API for accessing Public Key Credentials - Level 2*. https://www.w3.org/TR/webauthn-2/#dom-publickeycredentialrequestoptions-allowcredentials. (Accessed on 08/29/2024).

[47] Victor Le Pochat et al. "Tranco: A research-oriented top sites ranking hardened against manipulation". In: *arXiv preprint arXiv:1806.01156* (2018).

[48] Sophie Stephenson et al. "Abuse Vectors: A Framework for Conceptualizing {IoT-Enabled} Interpersonal Abuse". In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 69–86.

[49] *How to use AirDrop on your iPhone or iPad - Apple Support*. https://support.apple.com/en-us/119857. (Accessed on 08/19/2024).

[50] *Share passwords or passkeys with people you trust on iPhone - Apple Support*. https://support.apple.com/guide/iphone/share-passwords-iphe6b2b7043/ios. (Accessed on 08/19/2024).

[51] *Terms | passkeys.dev*. https://passkeys.dev/docs/reference/terms/#cross-device-authentication-cda. (Accessed on 09/01/2024).

[52] Carolina Ortega Pérez, Alaa Daffalla, and Thomas Ristenpart. "Encrypted Access Logging for Online Accounts: Device Attributions without Device Tracking". In: *USENIX Security Symposium*. 2025.

[53] *Passkeys.directory*. https://passkeys.directory/. (Accessed on 08/07/2024).

[54] *How do I use Passkeys on Snapchat? – Snapchat Support*. https://help.snapchat.com/hc/en-us/articles/27317385968532-How-do-I-use-Passkeys-on-Snapchat. (Accessed on 08/20/2024).

[55] *Passkeys : r/mintuit*. https://www.reddit.com/r/mintuit/comments/17pwenf/passkeys/. (Accessed on 08/20/2024).

[56] *Sign in & sync in Chrome - Android - Google Chrome Help*. https://support.google.com/chrome/answer/185277?hl=en&co=GENIE.Platform%3DAndroid. (Accessed on 08/30/2024).

A Details about Service Selection

For the selection of 15 services we used the Tranco list. Tranco provides a ranking of popular websites developed by researchers [47]; it is used frequently for web security studies and internet measurements. We did a fresh pull of the top 200 services on the default Tranco list generated on July 10th 2024. As our study focuses on the abusability of passkeys in interpersonal settings, we only include consumer-facing services, excluding business, enterprise, advertising and analytics services. After filtering through all 200 services, we were left with 73 services.

Following this filtering process, we removed duplicate URLs that lead to the same service. For example, WhatsApp showed up twice, once as `whatsapp.net` and another as `whatsapp.com` and both redirect to the same site. Additionally, we considered services connected through shared accounts as duplicates because they share a common authentication infrastructure and thus passkey behavior is going to be similar. For example, both `google.com` and `googlevideo.com` require a Google account for authentication. This also applies to the Microsoft suite of products and services, e.g., `live.com`, `office.com`, and `skype.com` all rely on a Microsoft account for using the service. After removing duplicates we were left with 59 services.

To learn which services supported passkeys we cross-referenced our list of 59 services with 1Password's passkey directory—a publicly available directory of apps and services that have adopted passkeys for sign-in and MFA [53]. We paired this with a manual search on Google's search engine to confirm the deployment of passkey support on the services. The keyword search terms we used included the name of the service coupled with passkeys, for example *snapchat passkeys*. Of the 59 platforms, only 14 platforms were mentioned in the passkey directory and four more (Snapchat, Samsung, Adobe, and Intuit) were discovered through our manual search process. For Snapchat and Intuit, while the search surfaced the availability of passkeys on both platforms in [54] and [55] respectively, we were not able to register a passkey on either using either a MacOS or an iOS device; we therefore exclude them from study. Finally, we excluded non-English services (e.g., VK), leaving us with 15 services for the study.

B Experimental Setups

We chose to perform abusability tests across multiple client configurations:

- MacOS (Macbook Pro 14inch, MacOS Sonoma v.14.5 using Chrome (v127.0.6533.120),
- Macbook Pro 16inch, MacOS Sonoma v.14.5) using Chrome (v131.0.6778.110),
- iOS (iPhone 11, iOS v.17.5.1),
- Android (Google Pixel, v.10), and
- Windows (Dell Inspiron, Windows v.11).

For our tests involving security keys, we used Yubico YubiKey (Security Key C NFC).

To test adversarial biometrics (Section 5.1), we experimented with three client victim configurations (MacOS, Windows, Android). To test passkey cloning (Section 5.2), and denial of service (Sections 5.4 and 5.5) we used a MacBook Pro and Chrome as the victim and alternated between iPhone 11 and Google Pixel as the adversary for testing in the Apple and Google ecosystem, respectively. For adversarial passkeys (Section 5.3), we used a Macbook Pro using Chrome as the victim's device, while the adversarial device was an iOS device for one of the authors and a MacOS one for the other author conducting the tests.

For passkey deletion Section 5.4, we used a Macbook Pro using Chrome as the victim's device for both Apple and Google ecosystems. The victim's device is the same device on which the abuser performs deletion. For passkey revocation Section 5.5 we used a Macbook Pro using Chrome as the victim's device and an iOS device as the abuser's.

C Additional Figures

Figures that could not fit in the body.

D Passkey Cloning & Deletion in the Google Ecosystem

Passkey cloning in the Google ecosystem. For Android client devices, there is no feature for exporting or sharing a passkey with another device, leaving the only route to passkey cloning as synchronization. Google passkey synchronization is similar to iCloud: the victim and attacker device should both be signed-in to the same Google account and that account should have sync enabled [56]. This requires the attacker's knowledge of the victim's Google password (2FA is irrelevant as we assume device access). Then, the attacker device is automatically given copies of all the victim's passkeys. Unlike iCloud, however, there is no option to retain synchronized items at sign-out, so the attacker must stay logged into the account.

To spoof the passkey label in the Google ecosystem, the attacker navigates to the Google Password Manager associated with the victim's Google account. This can be accessed in Chrome's *Passwords & Autofill* setting. Once there, the attacker can expand the passkey entry and edit the passkey to spoof the username.

Authenticating using synchronized passkeys. Login using the synchronized passkey was successful for Microsoft, Github, Amazon, Roblox, LinkedIn, Intuit, Ebay, and Adobe. On Samsung, authentication with a cloned credential fails for unclear reasons. Roblox downgrades user verification after syncing: logging in with the synchronized credential on the adversarial device did not require user verification (i.e., the attacker's biometric) despite requiring this on the original victim device. This may be because Roblox sets the service-requested authenticator property userVerification to discouraged.

Detecting passkey synchronizations. The fact that the attacker device is logged into the victim's Google account is visible on Google's ASIs. However, there is no indication on this ASI that passkeys have been synchronized; that fact is implicitly inferred from having the sync enabled setting. There is also no

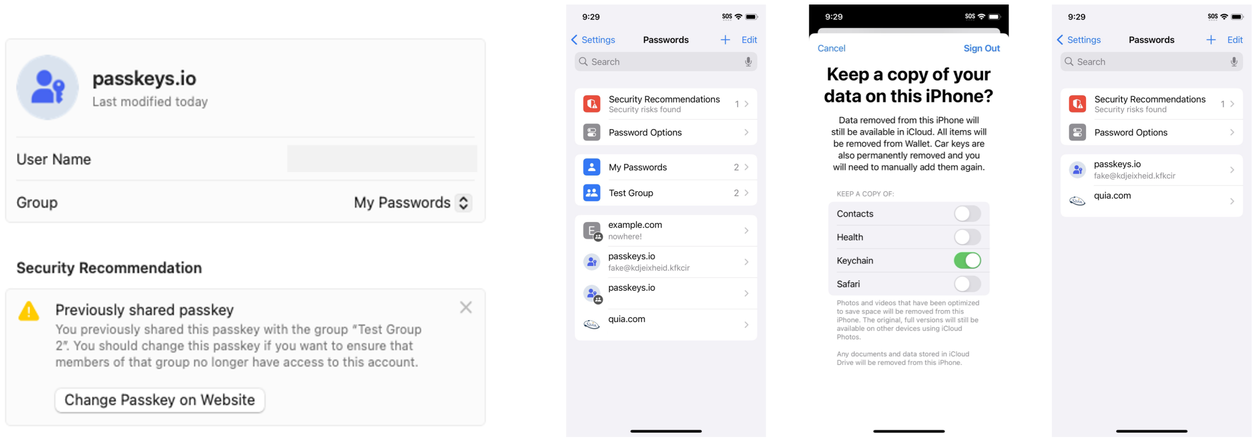
Task	Passkey Deletion	Passkey Revocation
Precondition	**(1-V)** Set up online account **(2-V)** Set up passkey on account [Apple] **(3-V)** Set up passkey on account [Google]	**(1-V)** Set up online account **(2-V)** Set up passkey on account
Locate passkey	**(4-A)** Unlock victim's device **(5-A)** Locate passkey ASI [Apple] **(6-A)** Locate passkey ASI [Google]	**(3-A)** Login to online account **(4-A)** Locate passkey ASI on service
Delete passkey	**(7-A)** Delete service passkey [Apple] **(8-A)** Delete service passkey [Google]	**(5-A)** Revoke service passkey on account
Deny access	**(9-V)** Authenticate using passkey [Apple] **(10-V)** Authenticate using passkey [Google]	**(6-V)** Authenticate using service passkey
Evidence of deletion	**(11-V)** Check device ASI for deletion [Apple] **(12-V)** Check browser ASI for deletion [Apple]	**(7-V)** Check service ASIs for revocation

Figure 8: Stepthrough protocols for the passkey deltion and passkey revocation abuse vectors. For each step we indicate whether simulates a victim action (e.g., 1-V) or an abuser action (e.g., 1-A). Additionally, we add [Apple] and [Google] for steps only performed in the Apple and Google ecosystem, respectively.

way to tell that the passkey label was spoofed. Service ASIs show varying amounts of login information from the attacker device; Figure 6 shows a breakdown of this information.

Revoking synchronized passkeys. The victim can log out the attacker device using Google's "*Your devices*" ASI. When the adversarial device is (remotely) signed out, the Chrome profile on the abuser device shows a *Sync is paused* text indicator and icon. This prevents the adversarial device from subsequently logging back into victim service accounts. Even if sync is paused however, existing active sessions on the adversarial device still remain active for all services. This is expected as session management on individual services is independent from sync management on Google, but nevertheless problematic from an abuse perspective. Similar to Apple, the victim can also revoke passkeys on individual services.

Passkey deletion in the Google ecosystem. In the Google ecosystem the passkey deletion stepthrough protocol (Section 5.4) was successful for ten services. TikTok and Twitter are only supported on iOS; thus, we could not create a passkey stored in Google's Password Manager. To perform a local deletion the analyst navigates to the Chrome settings menu, then *Passwords and Autofill*, and then *Google Password Manager*. The latter lists both passwords and passkeys. Similar to the iCloud Keychain, a user can edit each passkey credential to perform actions such as reconfiguring the credential's username or deleting the credential. The analyst deletes individual passkeys. Once deleted, they attempt to log into the service. As expected, access is denied for all nine services.



passkeys.io
Last modified today

User Name

Group: My Passwords

Security Recommendation

Previously shared passkey
You previously shared this passkey with the group "Test Group 2". You should change this passkey if you want to ensure that members of that group no longer have access to this account.

Change Passkey on Website

9:29
Settings | Passwords | Edit
Search
Security Recommendations — Security risks found — 1
Password Options
My Passwords — 2
Test Group — 2
example.com — nowhere!
passkeys.io — fake@kdjeixheid.kfkcir
passkeys.io
quia.com

9:29
Cancel | Sign Out

Keep a copy of your data on this iPhone?

Data removed from this iPhone will still be available in iCloud. All items will be removed from Wallet. Car keys are also permanently removed and you will need to manually add them again.

KEEP A COPY OF:

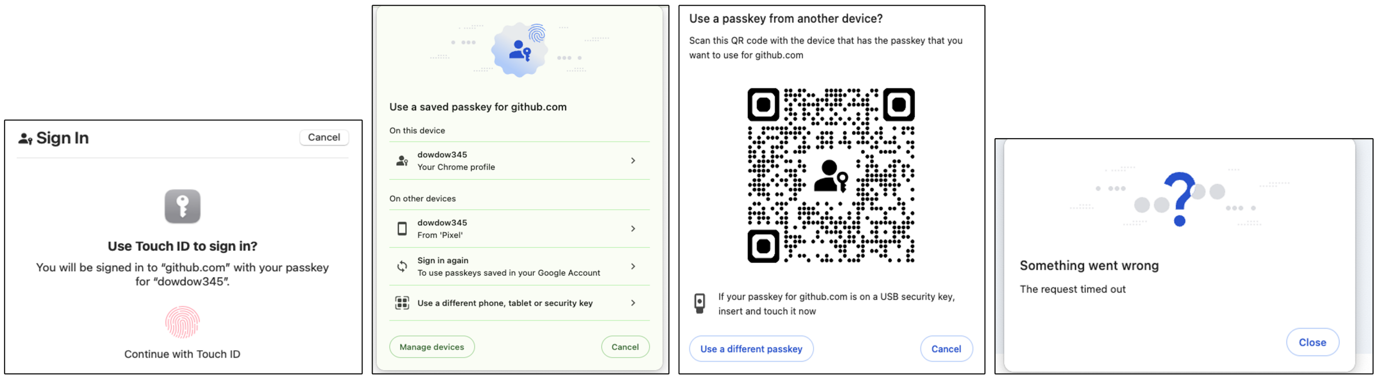
Contacts	Off
Health	Off
Keychain	On
Safari	Off

Photos and videos that have been optimized to save space will be removed from this iPhone. The original, full versions will still be available on other devices using iCloud Photos.

Any documents and data stored in iCloud Drive will be removed from this iPhone.

9:29
Settings | Passwords | Edit
Search
Security Recommendations — Security risks found — 1
Password Options
passkeys.io — fake@kdjeixheid.kfkcir
quia.com

Figure 9: **(Left image)** A security recommendation and warning on Apple's *Passwords* ASI informing users that a passkey was shared in a password group. **(Remaining images)** A series of screenshots depicting how iCloud allows a synced passkey to persist on the device. From left to right, the first screenshot shows the initial state of an attacker's "Passwords" page on an iPhone once they sign into the victim's iCloud account. The second screenshot shows the prompt to save Keychain information to this iPhone when signing out of the victim's iCloud account. The third and screenshot demonstrate how the set of credentials remain on the attacker's local device after iCloud sign out occurs. Note that shared credentials are removed upon sign-out but the non-shared credentials remain.



Sign In | Cancel

Use Touch ID to sign in?
You will be signed in to "github.com" with your passkey for "dowdow345".

Continue with Touch ID

Use a saved passkey for github.com

On this device
dowdow345 — Your Chrome profile

On other devices
dowdow345 — From 'Pixel'
Sign in again — To use passkeys saved in your Google Account
Use a different phone, tablet or security key

Manage devices | Cancel

Use a passkey from another device?
Scan this QR code with the device that has the passkey that you want to use for github.com

If your passkey for github.com is on a USB security key, insert and touch it now

Use a different passkey | Cancel

Something went wrong
The request timed out

Close

Figure 10: Series of browser and OS UIs when authenticating using a passkey with a service. First screenshot (1) shows the native default MacOS biometric authentication popup windows, second is the (2) Chrome popup that shows when a user cancels on (1), and third screenshot is the the QR code option when a user chooses the *different phone* option in (2). Finally, a browser popup when the passkey registration or authentication request times out.