Moving From Passwords to Authenticators

Alex Takakuwa

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2019

Reading Committee:

Tadayoshi Kohno, Chair

Alexei Czeskis

Franziska Roesner

Program Authorized to Offer Degree:
Paul G. Allen School of Computer Science & Engineering

University of Washington

Abstract

Moving From Passwords to Authenticators

Alex Takakuwa

Chair of the Supervisory Committee:
Professor Tadayoshi Kohno
Paul G. Allen School of Computer Science & Engineering

Humans have used passwords for access control since ancient times. Upon the advent of the internet, passwords naturally transitioned to the web and have since become the standard mode of web authentication. However, over the last 25 years, password authentication has shown persistent and unavoidable security and usability problems. Many within the computer security industry believe that we can improve the state of the art in both security and usability by utilizing asymmetric challenge-response protocols for authentication. For example, the FIDO Alliance, a group of industry and academic partners working together to bring secure and usable authentication protocols to the web, utilize such asymmetric cryptographic protocols to help strengthen the authentication flow. Nevertheless, despite industry and academic desire to improve web authentication, passwords remain the status quo for users. In this dissertation, I present the landscape of authentication protocols and propose solutions allowing users to upgrade devices and recover from device loss — two of the remaining technical challenges that prevent modern authentication schemes from supplanting passwords as the dominant method of web authentication.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to acknowledge God for creating the bar with which we measure ourselves, and Vlade Divac for setting it. This thesis represents the combined work, effort, and especially patience of an innumerable number of people, starting with my advisor, Yoshi Kohno, who believed in me enough to bring me to UW and tolerated me enough to see me submit this document. I would also like thank Alexei Czeskis, without whom the entire ecosystem on which this dissertation is based may not exist, and without whose advice, support, and patience, neither would this document. Also for suffering through my conversational writing style enough to fix it — a true Sisyphean effort.

I would like to thank my colleagues at Google and the FIDO Alliance who have helped brainstorm, develop, and implement the ideas contained within this document and who have helped push these ideas toward standardization. In particular, I'd like to thank Arnar Birgisson, Alexei Czeskis, and Hideo Nishimura who have supported this push since the beginning. I'd also like to acknowledge the collective efforts of Brett McDowell, Christina Hulka, Laura Dorsey, and Keith Lutsch for their multi-year collective uphill effort to get me into the FIDO Alliance as a contributor.

I would like to acknowledge the additional members of the security lab throughout my time for our discussions and arguments that helped me shape my points of view, and for the technical support they have given me throughout my time at UW (Karl Koscher, Miro Enev, Tamara Denning, Franziska Roesner, Temitope Oluwafemi, Ada Lerner, Paul Vines, Camille Cobb, Peter Ney, Ian Smith, Eric Zeng, Anna Kornfield Simpson, Kiron Lebeck, Lucy Simko, Ivan Evtimov, Christine Geeng, Christine Chen, Kimberly Ruth, Shrirang Mare, Earlence Fernandes, and Yutaka Matsubara)

I'd like to thank the "front office family", Tracy Erbeck, Sophie Ostlund, Lindsay Michimoto, Elise deGoede Dorough, Rebecca DeGaris, Rebekah Hansen, Alexander LeFort for feeding me and putting up with the many lost hours of productivity I've caused them, but even more so for the invaluable advice and motivation they've given me. I'd like to thank Melody Kadenko, the CTF team, and Alexei Czeskis for taking me to my first computer science conference at DEFCON and showing me the ropes, and especially Melody for helping me at any hour to make sure the bills got paid.

I would also like to thank my family for supporting my decision to remain in academia for an unreasonably long period of time. I'd like to thank my friends from all over who have tried their best to keep me sane over such an extended period. Despite being futile, your valiant efforts are appreciated immensely and will not be forgotten. I'd like to thank my teammates from all of my soccer/basketball/softball teams for providing the last bit of exercise before my body completely deteriorates. Lastly I'd like to thank all of my roommates over the years who've kept me company and brought me so much joy and rent.

This dissertation and the other works from my graduate career not appearing here were funded by the National Science Foundation (61-1117 NSF CNS-1565252, 62-7161 NSF CNS-1329751), DARPA (62-3975, DARPA FA8750-12-2-0107), the University of Washington, and the Anne Dinning-Michael Wolf Endowed Regental Fellowship.

DEDICATION

To my grandparents, who persevered through tough immigration and internment to provide the privilege that allowed me to complete this degree.

Chapter 1

INTRODUCTION

Passwords are the status quo in web authentication. From [28], *web authentication* is "the process by which one entity (e.g., a server) identifies another entity (e.g., a user) remotely over the web". The security and usability of this process is of the utmost importance to both the server (Relying Party/RP) and the user. But despite decades of effort to improve password schemes to make them more usable and secure (see Section 2.4), problems persist [8, 31, 80]. As a result, industry and academic researchers have begun to turn to alternative authentication schemes based on asymmetric cryptography, which can provide much stronger security guarantees.

The FIDO alliance [5] is a collection of hundreds of companies who share the goal of secure and usable authentication. These companies represent a significant industry push to move away from passwords toward more secure modern authentication based on asymmetric cryptography. Though previous proposals (Universal Authentication Framework (UAF) and Universal Second Factor (U2F)) failed to truly replace the password ecosystem, FIDO continues to forge ahead in the quest to replace passwords by attempting to unify the use cases from both UAF and U2F into one protocol — FIDO 2.0. FIDO 2.0 consists of two components: 1) a specification for browsers and 2) a specification for everything else. The W3C published the former as a formal W3C recommendation called the WebAuthn protocol [14]. FIDO published the latter as the Client to Authenticator Protocol (CTAP) [4]. Together, these specifications enable a completely password-less user experience, significantly improving the usability of the authentication flow. In such a specification, users will be able to leverage existing devices such as phones or laptops as cryptographic keys to easily and securely log in to websites, protecting users from the vast majority of existing attacks, such as phishing,

and simplifying the user experience by removing the requirement that users type usernames and passwords. For example, in FIDO 2.0 a user can navigate to a banking website on a PC, receive a notification on her phone, and scan a fingerprint via the phone's fingerprint sensor to log in. This user experience can be tailored to each person's unique preferences and constraints while maintaining a level of security far exceeding that which passwords offer.

However, despite the industry and academic push toward such schemes, there are outstanding barriers to adoption. For example, in addition to creating more secure and more usable protocols, industry and academic researchers must find ways to convince users to switch to modern authentication. In some environments, such as within corporate networks, companies can force employees to adopt modern authentication. But in the rest of the world, the new authentication schemes will have to provide users a better experience *and* convince users that the ecosystem is more secure. The FIDO Alliance has made significant progress on both fronts and has pushed us considerably closer to the adoption of modern authentication schemes, but there are still challenges to solve before mass-scale adoption is possible. The contributions in this thesis focus primarily on solving two of these remaining challenges: *upgrading devices* and *recovering from device loss* in the WebAuthn ecosystem.

1.1 Problems with Device Upgrade and Recovery in WebAuthn

1.1.1 Device Upgrade

In the password ecosystem, when a user gets a new phone she does not need to update any existing passwords. She can simply navigate to the sites with which she already has accounts and authenticate normally. But in the WebAuthn system, when a user gets a new device, she needs to register that new device for each of her online accounts. Even worse, she will likely need to use password-based authentication first in order to start each registration. Should she forget her password (which she likely wouldn't have used for authentications), she even may have to run an account recovery first. So while WebAuthn can improve the security and usability of the standard authentication flow, falling back on passwords for common scenarios

such as device upgrade both leave weak links in the chain and dampen the usability gains WebAuthn provides.

1.1.2 Recovering from Lost Devices

Recovering from a lost device is a similar, but more difficult problem. In the event that a user still has a password, the recovery proceeds very similar to that described above. However, in an ecosystem where users no longer have passwords associated with their accounts, recovery is far more difficult, if not impossible. Without access to a registered device or a username and password, the user may need to enter into a site-specific recovery process for every single online account or may lose access to the account altogether.

1.2 Contributions

This dissertation makes the following contributions toward solving problems related to *Device Upgrade* and *Recovering from Device Loss* in WebAuthn:

1.2.1 Discussion of Solution Space

Chapter 3 provides an investigation of the many different options for solving these two problems, including an analysis of costs and benefits of each approach. As a result of this analysis, this dissertation identifies a form of credential binding as the method with the most promise. Credential binding allows for creating chains from trusted credentials to new credentials that the server can trust as strongly as it does the original credentials. Most importantly, credential binding does not require copying authentication material from one device to another, which would degrade the security guarantees of WebAuthn. As shown in subsequent chapters, there are ways to implement protocols based on credential binding that enable upgrading devices and recovering from lost devices without sacrificing any of the security or privacy properties afforded by WebAuthn. Further, the proposals in Chapters 4, 5, 6 show how to do so while minimizing user burden.

1.2.2 The Transfer Access Protocol

Chapter 4 presents one of those proposals. The Transfer Access Protocol uses credential binding to allow users to upgrade devices in FIDO protocols without sacrificing security or privacy at any step. This means it employs the same digital signatures used in the asymmetric challenge response protocols of WebAuthn authentications to transfer access from the old, trusted credentials to the fresh ones created on the new device — without changing any of the web APIs. Further, it does not add any additional steps for users and scales gracefully to large numbers of credentials and large numbers of relying party sites. From the user's perspective, authentication proceeds exactly as it did on the old device, without a single difference. However, the server sees the transfer access message and can check each link in the credential chain to ensure that it trusts every device through which access has transferred. Among other contributions, this section summarizes a security analysis of each step of the protocol and provides an implementation based on Google's public U2F implementation. Although this implementation uses the U2F framework, many of the core principles and insight still apply to the WebAuthn ecosystem with minor adjustments. I presented this work to members of the FIDO Alliance at an official FIDO Plenary in 2017 (Vancouver). This chapter includes a summary of feedback from that session.

1.2.3 Pre-Emptively Syncing Keys

Chapter 5 presents a potential solution that allows users to recover from device loss by pre-emptively syncing keys to a backup device. This solution builds upon the insights of the *Transfer Access Protocol* using similar chains of signatures from trusted credentials on trusted devices to new credentials on new devices. As discussed in Chapter 3, the FIDO Alliance recommends that users register multiple devices to prevent being locked out of an account [42]. However, from a user's perspective, this means that users need to have two devices available during any registration. Given that users may not be able to predict when they need to create online accounts, the *Pre-Emptively Syncing Keys* solution aims to reduce

user burden while still allowing the user to effectively register multiple keys with a single device. For users, there is one additional step during setup of any new device: they must register a backup device, but need not do so for each ensuing registration of that device. Crucially, the *Pre-Emptively Syncing Keys* solution does not require the copying of any key material, and does not sacrifice any of the security or privacy properties of WebAuthn, nor does it require changes to the WebAuthn web APIs. However, there are tradeoffs associated with this solution. Namely, it requires a storage overhead on the device that is to be used as an authenticator. Some authenticators are small, low-powered devices that have limited computation and storage. Further, such a protocol presents some issues when used with usernameless flows. Chapter 5 contributes a discussion of these tradeoffs. I presented this solution to multiple members of the FIDO alliance and incorporate the feedback from those presentations in this chapter.

1.2.4 Online Recovery Storage

The *Online Recovery Storage* solution presented in Chapter 6 attempts to provide an alternative without the above tradeoffs at the cost of each user employing some highly available recovery storage platform. Like the *Pre-Emptively Synced Keys* solution, it builds on the *Transfer Access Protocol* and uses Credential Binding to ensure the server trusts the new devices and credentials. It does not require copying of any key material, sacrifice any of the security or privacy benefits of WebAuthn, or changes to the WebAuthn web API. However, it does require the user to perform two additional steps during the initial setup. First, the user must set up some sort of online recovery storage to store encrypted metadata. This could be a cloud storage provider, an email attachment, or some other form of simple online storage. Second, like the *Pre-Emptively Synced Keys* solution, the user must link a backup device and store it for future devices. Although an *Online Recovery Storage* platform is a third party, it can be implemented as dumb storage which can obtain no information about credentials, registrations, or authentications. Because online storage is cheap, this solution offloads the minimal storage overhead from end user devices. And by keeping more metadata, such a

solution works well with usernameless flows. I presented a previous version of this work at an industry Summit hosted at the University of Washington in 2018. The proposal in this dissertation incorporates the feedback from that presentation and summarizes the progress made during the summit.

Chapter 2
BACKGROUND AND RELATED WORK

This chapter presents information, concepts, and terminology useful for understanding the work and proposals to follow in this and later sections. We present attacks and define our threat model in Section 2.2. In Section 2.3 we explore some of the vast volume of work that analyzes and identifies strengths and weaknesses of the current password ecosystem. We briefly introduce some existing proposals to improve or replace the status quo in Section 2.4. Finally, we explore necessary details of the WebAuthn [14] protocol to provide the background for the preliminary and proposed work in the rest of the paper.

2.1 Ecosystem and Description of Terms

In the pieces of this work discussing WebAuthn and other asymmetric challenge-response protocols, we use certain terms to refer to key entities within the system.

In these schemes, there are two main actions a user can perform:

- **Registration** occurs when the user wants to create an account or register a new key to an existing account.

- **Authentication** occurs when a user wants to access an existing account and needs to provide evidence of account ownership in order to be authorized to access those resources.

In either case, a user communicates with a *relying party* (or RP) through a browser or mobile application (client). This relying party serves the web page or provides the mobile app (e.g. www.google.com or the endpoint for the Facebook app).

To perform the cryptographic operations necessary for these actions, a user uses an *authenticator*, which is a device that provides the necessary hardware and software functions to perform all cryptographic computation and storage the authentication scheme requires. Though modern smartphones can perform many more functions than an authenticator, in most of our preliminary and proposed work we focus on their ability to act as authenticators and use the terms "phone" and "authenticator" interchangeably.

2.2 Attacks/Threat Model

Throughout this work, we will reference attackers and specific types of attacks. In this section, we introduce common attacks and present a threat model. As a result of these attacks, we also give examples of goals for future authentication schemes. We present three classes of attacks as done by Alexei Czeskis in his dissertation on the subject [28]. As in that work, we consider software compromise of the key components (the authenticator, the browser, and the relying party server) as out of scope.

Attacks fall into three main categories: *Web Attackers*, *Network Attackers*, and *Related Site Attackers*.

- ***Web Attackers*** operate malicious websites that can be used to steal credentials. The most common type of web attack is *phishing*. Phishing has become a standard method for compromising account credentials, with tens of thousands of active phishing websites targeting hundreds of companies [12]. According to the Anti-Phishing Working Group [12], "Phishing is a criminal mechanism employing both *social engineering* and *technical subterfuge* to steal consumers' personal identity data and financial account credentials." This can include using spoofed e-mails or counterfeit websites designed to steal data, such as usernames and passwords. Though phishing is not a new attack, previous attempts to mitigate the effectiveness of such attacks proved unsuccessful [8]. A number of high-profile breaches involving phishing [48] helped push leading industry members toward phishing-resistant solutions, for example, FIDO's UAF and U2F [5]

and Google's Advanced Protection [44].

- ***Network Attackers*** are able to observe and potentially alter web traffic between a client and a relying party. The classic description of this set of attackers is the *Man-in-the-Middle* (MITM) attacker, who can range in power and capability from a Wi-Fi router which can alter and read un-encrypted HTTP requests, to a state sponsored network attacker who can potentially decrypt TLS connections [1, 92]. Like successful Web Attackers, Network Attackers can use their capabilities to extract credentials from online communications.

- ***Related Site Attackers*** use attacks on one site to attack another. This attack is successful because users will often make security concessions in order to ease the burden of managing passwords [19, 38, 51, 56, 67, 86, 109, 110]. For example, if a user selects the password `'passwordForExampleSite'` at `www.examplesite.com` and `'passwordForRelatedSite'` on `www.relatedsite.com`, attackers can use the similarities to leverage the compromise of credentials at one site to compromise credentials at other sites.

Although this work focuses on situations where users have fully patched software and are not subject to an active zero-day attack on the components of the authentication system, we note that there is evidence that attackers are using software attacks to compromise some percentage of vulnerable authentication systems. For example, in the EuroGrabber attack, miscreants installed trojans on users' phones to intercept two-factor codes and authorize the transfer of nearly $50 million USD [63]. There is also evidence that attackers have success using keyloggers to compromise account credentials of unwitting users [104]. We place this out of scope simply because designing systems that can resist active adversaries within the system is a separate, far more difficult problem from the common authentication case we discuss in this work. However, to the extent possible, we aim to design our systems so as to resist as many software attacks as possible without compromising security or usability.

In response to the attacks presented in this section, we would like to move toward authentication protocols which exhibit the following properties, adapted from Lang et al. [66]:

- Prevent Phishing: The protocol should not be phishable, nor should the resulting credentials.

- Defend Against MITM: Attackers who Man-In-The-Middle the connection, for example between the browser and relying party server, should not gain an advantage by attacking during any step in the authentication protocol.

- Prevent Session-Duplication: The protocol should not allow attackers to steal credentials that would result in a duplicate, attacker-controlled session, for example, by exposing long-term cookies or passwords.

- Prevent Session Riding: The protocol should not allow an adversary to gain access to an existing session or to a future existing session.

- Trusted Hardware: The protocol should allow the relying party to verify that it trusts any authentication specific hardware, such as authenticators (see Section 2.1).

- Non-Linkability: Credentials should be site-specific by default so that colluding relying parties can not link credentials to users across sites.

2.3 Evaluating the Password Ecosystem

In 2012, Bonneau et al. produced a seminal work in web authentication which helped explain why the world was having so much trouble moving away from passwords [20]. The authors developed three categories by which to judge web authentication schemes: *Security, Usability, and Deployability*. For simplicity and consistency, we will use their terminology and framework throughout this section to discuss existing work.

2.3.1 Deployability

Bonneau et al. [20] identified Deployability as one of the key components of an authentication scheme. They state that passwords are:

- *Accessible:* Disabilities or other physical conditions do not prohibit usage of passwords.
- *Negligible-Cost-per-User:* Costs for the relying party and users are negligible, indicating that even start-ups with no revenue can plausibly implement such a scheme.
- *Server-and-Browser-Compatible:* Users and relying parties don't have to change existing authentication infrastructure in order to support passwords, as they are already the status quo.
- *Mature:* As the incumbent web authentication scheme for over 20 years, passwords are widely deployed and implemented across the web.
- *Non-Proprietary:* The core mechanisms used in password authentication are not protected by patents or trade secrets and are available to anyone to implement.

2.3.2 Usability

Bonneau et al. [20] show that passwords also have numerous advantageous usability properties:

- *Nothing-to-Carry:* Because users can remember passwords, they do not need to carry any extra device or object to use the authentication scheme. Although users prefer different usability qualities, most do not report preferring systems that require users to carry devices that help them authenticate [60, 74].
- *Easy-to-Learn:* Users already understand how to create passwords and accounts, use passwords to authenticate, and recover accounts and can do so passably [74], though doing it securely is another matter entirely [19, 38, 51, 56, 67, 86, 109, 110].

- *Easy-to-Recover:* The most common way to recover account access today is to register a "backup email", which the user can use to reset access to an account. This approach usually requires a user to register said email at each relying party and verify possession of the email address. In many cases, the user's registered email address is the account identifier, raising some security and privacy concerns [53, 77].

However, passwords do suffer from some usability drawbacks. For example, password schemes are subject to relatively frequent errors (according to Mare et al., false rejections occur around 12% of the time [74]). Password schemes also require users to *Remember Secrets*, *Physically Type Passwords*, and tend to *Scale Poorly* for users who have many accounts [20].

2.3.3 Security

Continuing with the definitions from Bonneau et al. [20], passwords have the following security benefits:

- *Resilient-to-Theft:* Given that password authentication schemes do not require users to carry object or device, attackers have nothing to physically steal.

- *Not-Reliant-on-Trusted-Third-Parties:* The standard implementation occurs between the user's web browser and the relying party servers and does not require interaction with any other party. There is no other party to attack that could compromise a user's security or privacy. However, in practice, *Related Site Attackers* (see Section 2.2) are able to compromise credentials at one site and use those credentials to compromise a user's security and privacy with a different relying party.

- *Explicitly-Require-Consent:* Users must physically type the password, indicating they consent to being authenticated. Forcing user consent helps clarify user intent and prevents attacks that would, for example, charge a user's account without consent.

- *Unlinkable:* Passwords and accounts can be made unique per relying party. In practice, users tend to ignore this to reduce burden, re-using email addresses and passwords across many relying parties [19, 38, 51, 56, 67, 86, 104, 109, 110].

In the above framework, of the three categories (*Deployability, Usability, Security*) passwords receive the lowest grade in security, despite the benefits mentioned above. This is because passwords suffer from a number of security pitfalls listed below:

Copyable: Passwords can be *Copied by External Observation* (ex: observing users as they type) by *Observing Internally* (ex: from a key logger which records key strokes) or by *Phishing* (see Section 2.2). Despite improvements in browser warnings and phishing detection, users seem as susceptible as ever to phishing attacks [8, 12] and attackers can keep pace with minimal updates to phishing toolkits [104].

Guessable: As a result of users' poor security practices for creating and managing passwords [25, 34, 40, 48, 81] and servers' poor implementations [80, 104], both *Throttled and Unthrottled Guessing* have proven effective strategies for exposing credentials.

Leakable: Further, because users recover accounts using other (email) accounts, and because they re-use passwords across many relying parties, password schemes are vulnerable to *Leaks-from-Other-Verifiers*. For example, an attacker who compromises a user's backup email can simply reset the password for an account and receive the new credentials via the backup email.

Impersonate-able: In practice, sites often use personal information as recovery codes [55]. Users will also often use personal information like birthdays as pieces of passwords [67]. This opens up password systems to *Targeted Impersonation*, where an attacker acquires personal information about a subject and leverages that information to obtain account credentials.

Attacks against the password ecosystem have begun to proliferate into the public conscience. A number of high-profile phishing attacks [25, 34, 40, 48, 81] and data dumps [33, 35, 39, 41, 47, 57, 65, 102] have forced companies to create more secure ways of authenticating

users [9, 44, 46, 76, 100]. Many new schemes have been proposed, from trying to increase awareness to forcing users to create better passwords, but numerous studies show that the password ecosystem as a whole fails to resist these types of attacks [8, 109]. For example, as a lower-bound, from March 2016 to March 2017 phishing and data dumps alone accounted for *at least* hundreds of millions of stolen active credentials [104].

2.4 Improving Upon the Password Ecosystem

Despite the growing evidence that the security of password authentication on the web was insufficient for many uses and evidence that users would benefit from more usable schemes, studies find that users are relatively comfortable with passwords [56]. They also showed that users were not willing to sacrifice usability for security, often making the opposite trade-off [27, 49, 52, 60, 61, 68, 74, 111]. To convince users to learn a new, more secure way to authenticate, a proposal would have to match or improve upon the usability of passwords for a critical mass of users. This has proven very difficult. In this section, we detail some of the attempts to improve web authentication. These attempts generally fall into two categories: 1. Augmenting the existing password ecosystem and 2. Replacing the password ecosystem with asymmetric cryptographic schemes.

2.4.1 Augmenting the existing password ecosystem

There has been an incredible volume of work in this space including studying user behavior [19, 38, 51, 56, 67, 74, 86, 110, 111], attempting to improve the strength of passwords [22, 109], reducing burden on users [15, 37, 52, 60, 68, 72, 75, 85, 96], and changing the way servers store authentication data [21, 80, 96, 97].

In this subsection I briefly cover two of the main categories for improving the existing password ecosystem: Two Factor Authentication (2FA/TFA) and Password Managers.

Two Factor Authentication Two-Factor Authentication aims to thwart attacks by requiring a signal unique to the user from two of the following three categories:

1. Something you are (e.g. biometrics)

2. Something you have (e.g. your phone or a physical token)

3. Something you know (e.g. password)

In order to breach a system using 2FA, an attacker would need to obtain two of three factors, ostensibly a more difficult task.

Following initial proposals for 2FA [5, 7, 23, 32, 73, 107], most of the large online players added two factor options for their users [9, 27, 44, 46, 76, 88, 100]. However, attackers can still thwart many of these schemes with common attacks. For example, in schemes combining a knowledge factor (password) with a possession factor (phone to receive a two-factor code) attackers can still phish the user-entered two-factor code. Further, attackers can use standard toolkits to obtain other sensitive data that helps hijack accounts [31, 104]. Some two-factor schemes, however, like the FIDO U2F [5] protocol we use in our implementation in Section 4.1, have been shown to be secure against attacks when implemented correctly [87], but do not improve upon the usability of passwords.

Password Managers Password managers help ease the burden of creating and remembering strong unique passwords for each RP by storing credentials until the user wants to authenticate. Password managers have many different approaches. Some help the user generate strong passwords [85, 96], others focus on usability [15, 60], while others focus on defending against active adversarial attacks [37, 75].

As an example, LastPass [85] allows users to have a single password with which they can encrypt all their other passwords. It then generates strong passwords for all the user's accounts and can provide the passwords to the user when necessary. All passwords are encrypted in a browser plugin on the user's machine and stored on the LastPass servers. When a user wants to use a password, he grabs a blob of his encrypted passwords from LastPass and decrypts it on his phone or computer via the supplied application or browser

plugin. LastPass software running in the browser determines if the origin is correct, and if so, it fills the password form for the user. Many users like password managers such as Last Pass for their security and usability. For example, one user from [60] stated that they liked LastPass because it was "easy and [there was] no need to carry anything". However, a password manager that allows use across devices with a single master password introduces a single point of failure that, as others noted, might make it "more vulnerable towards the attacks from cyber criminals" [60]. LostPass, for example, is a phishing attack targeted specifically at the LastPass browser extension meant to trick users into providing their master password [24].

2.4.2 Replacing the Password Ecosystem

There have been numerous proposals for improved authentication schemes that help to solve some of the outstanding problems with password authentication. Of these, the majority use asymmetric cryptography, as the challenge-response capabilities of such a system help prevent *Phishing*. Further, the ability to perform challenge-response (where the relying party issues a random challenge unique to each authentication, and each response depends on said challenge) helps prevent *Replay Attacks*, which may result in *Session Duplication*. In this section, I provide an overview of some of these schemes. I also present two password-based authentication schemes, Sphinx and the Secret Sharing Password Manager, which change the client side behavior significantly to harden the password ecosystem.

Sphinx [96] uses a phone and client computer together to create strong passwords. The user authenticates himself to the client computer with a master password. Via a secure connection, the phone mixes its own stored information with that of the client to send a password to the server. The scheme ensures the credentials can't be phished by sending a credential that depends on the origin. Sphinx requires a master password at the client, which is potentially phishable [40]. However, an attacker would need to both eavesdrop on local communications between the phone and the client *and* phish the master password in order to get valid credentials to sites. *Related Site* attacks may still be effective against Sphinx in

the case where the user registers a compromisable email account as backup.

The Secret Sharing Password Manager [37] generates strong passwords and splits them across three resources using secret sharing: a personal server, a phone, and a browser. In order to reconstruct a password, the user must access two of the three resources. Further, if the user loses access to one of the three resources, he can use the remaining two to restore the third. Setup and registration for online accounts is rather exhausting, requiring the user to contact all three resources for each account with each relying party. When shares of a specific resource are *stolen*, revocation requires visiting each account and re-registering.

Passphone [89] is a two-factor authentication system using a smartphone to perform asymmetric cryptographic operations. Passphone outsources user verification to a trusted third party, while preserving anonymity and unlinkablity for users. It achieves these properties without sacrificing security guarantees by using nonces and hashing to blind the prover's identity, but requires a trusted third party.

MPAuth [72] is an authentication scheme that aims to defend against phishing and log-ins from untrusted PCs. It requires that the server send a challenge, which gets forwarded to the phone along with its origin information and public key. The phone checks the public key of the website, takes a password input from the user, and encrypts that information with the public key so that only the website can read it. This scheme still requires passwords and doesn't improve the usability for users at all. Further, such a scheme is susceptible to phishing if an attacker can convince the user to enter credentials for a different site.

Phoolproof [84] is an authentication system that is, in many respects, very similar to WebAuthn (see Section 2.5). It uses a phone to generate and store asymmetric key pairs and uses those for authentication. They suggest using phone calls for recovery and revocation because users already understand how to make phone calls. However, Parno et al. [84], were not likely considering applying Phoolproof authentication for every type of online account. We view their setup and recovery/revocation flows as far too burdensome for all user accounts on the modern web.

PhoneAuth [29], like Phoolproof is very similar to WebAuthn (see Section 2.5). It also

uses asymmetric cryptography in a challenge response scheme meant to prevent standard attacks such as phishing, man-in-the-middle, and related site attacks. In PhoneAuth, the phone and PC communicate over a Bluetooth connection without user interaction and without initial pairing using RFCOMM connections. However, PhoneAuth still requires a username and password log-in to set up this connection. Because of this, PhoneAuth is strictly less usable than a traditional password scheme for users, but allows the relying party to opportunistically upgrade to a more secure authentication. Improving usability of the initial Bluetooth pairing between a phone and computer for a scheme that does not incorporate any username and password is an open problem that, if solved, would benefit WebAuthn greatly.

2.5 FIDO Alliance and WebAuthn

The related work most relevant to this dissertation is the work done by the FIDO Alliance standardizing protocols like the U2F protocol [5] and FIDO 2.0 [18], which led to the Web Authentication standard (WebAuthn) at the W3C [14].

The FIDO Alliance [5] envisions a world without passwords, providing the tools to revolutionize the way users authenticate on the web. The current WebAuthn protocol provides secure standards that promise to improve online account security and simplify the experience for internet connected users. This ecosystem allows users to sign into web services through authenticators (for example, a smartphone or dedicated token) that perform user authentication using an asymmetric cryptographic signature that is resistant to phishing attacks and provides two-factor authentication. Similar to the iPhone's TouchID, users on many platforms will have devices, such as phones, that can serve as FIDO authenticators. For example, imagine that a user is using a phone as an authenticator. This phone has an app that allows the user to view and manage keys. It also allows the user to log-in to websites using FIDO authentication. When the user goes to example.com and selects "log-in with authenticator", the phone alerts the user to scan a fingerprint. The user complies and the server and authenticator app negotiate in a cryptographic protocol to ensure that the user is safely authenticated and consents to the log-in.

This is the environment which frames the work in this dissertation. Rather than go into technical detail about these standards (which change over time and which will hopefully change in time to incorporate solutions to the problems raised in this work) this section simply describes the pieces of the current WebAuthn standard relevant to this dissertation. Because later sections propose changes to the spec, this section will not rigidly follow the existing specifications. However all proposals contained within this document try to preserve the benefits of the current specification and as a result try to leave as much of the existing specification as possible unchanged.

In order to use the WebAuthn protocol to access a Relying Party's resources, a user must have a *client* and an *authenticator*. Examples of *clients* are a browser or a mobile application. An *authenticator* is a device that performs the cryptographic operations necessary for the WebAuthn protocol. Although modern phones can perform many more functions than an authenticator, in most of our preliminary and proposed work we focus on their ability to act as authenticators and use the terms "phone" and "authenticator" interchangeably, ignoring for simplicity the fact that a phone can also act as a *client device* which houses multiple client applications.

There are two core actions a user can do with an authenticator: Registration/Enrollment and Authentication. We describe the user flow for each.

2.5.1 The WebAuthn Registration Flow

Registration occurs the first time a user registers an authenticator with a Relying Party (RP). This can happen during the account creation upon first visiting the site/app, or as per [14] a user can log-in to an existing account and add an authenticator. In the following flow, a user registers a phone as an authenticator. The registration proceeds in the following steps:

1. The user navigates to the RP site (or app) and initiates account creation or registration of a new authenticator. Note that if the user navigates to the RP on a different device

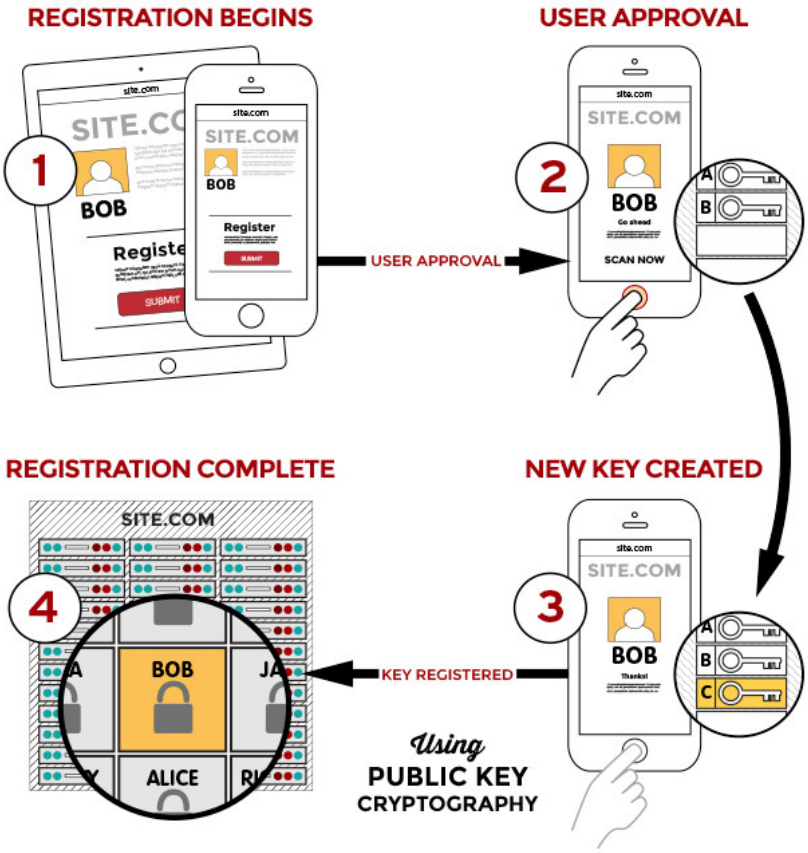


Figure 2.1: FIDO Registration. Taken from [5]

(for example, their PC), they will have to pair that device with their phone.

2. The phone asks for the user's permission to register the device with the RP.

3. The user grants permission.

4. The phone asks the user to provide an *authorization gesture*. This can be a simple tap to confirm user presence or a second factor, as per Section 2.4.1. Note that the possession of the authenticator (which provides the cryptographic credential) is the first factor, so the second factor will have to be either (a) something the user *knows*

(ex: PIN) or (b) something the user *is* (ex: biometric).

5. The user provides the authorization gesture and the registration is complete.

6. The phone can also ask the user to provide a user-readable name for the credential when stored locally, or this can be provided by the RP.

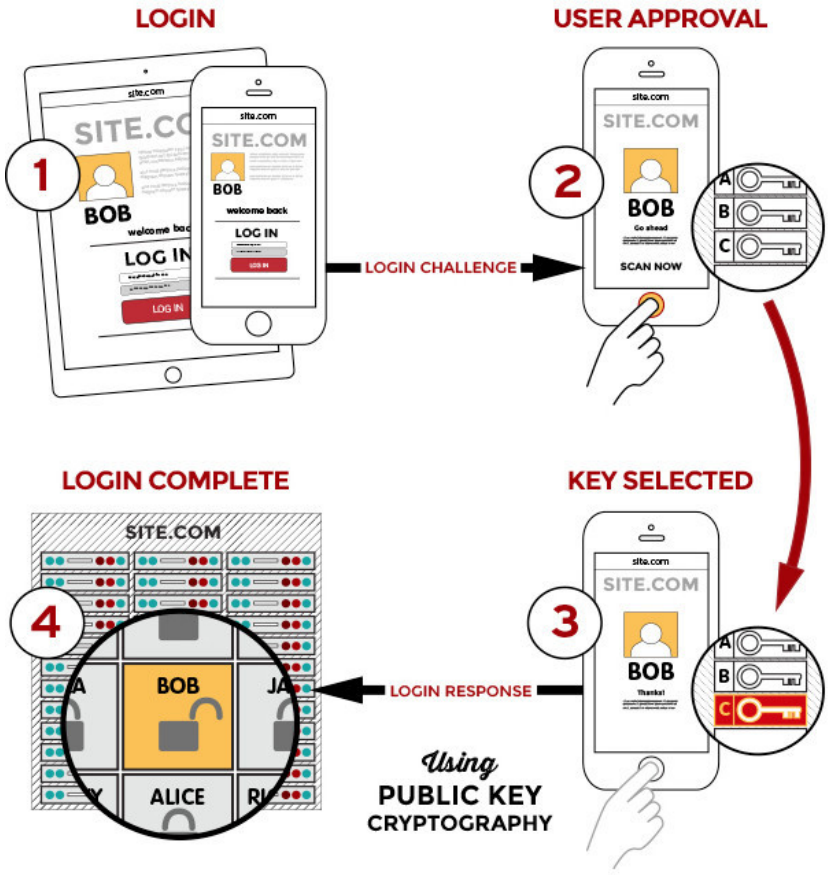


Figure 2.2: FIDO Authentication. Taken from [5]

2.5.2 The WebAuthn Authentication Flow

Authentication occurs after an authenticator has already been registered with an account. This proceeds as follows:

1. User navigates to the RP and selects the option “Sign in with phone”.

2. The browser informs the user to complete the authentication via the phone.

3. The phone sees a notification of a log-in request from the RP and displays all stored credentials matching that site.

4. The user selects the identity he would like to use to log-in to the RP (This can be skipped if the user only has one account with the RP).

5. The phone asks the user to provide an authorization gesture (ex: biometric or PIN).

6. The user provides the authorization gesture and the web page shows the user as successfully logged-in.

2.5.3 Deployability of the WebAuthn Scheme

WebAuthn has the following usability benefits:

- *Accessible:* Because users have a variety of choices of authenticator form factors and second factors, the WebAuthn scheme can be made accessible for users with unique physical conditions.

- *Negligible-Cost-per-User:* Unfortunately, users will have to purchase authenticators at cost. However, if users already own a compatible phone, it can act as an authenticator without extra cost. For example, as of April, 2019, phones running Android 7.0+ can act as valid FIDO2/WebAuthn authenticators [101].

- *Server-and-Browser-Compatible:* As of March 2019, WebAuthn is already well deployed in browsers. Windows 10, Android 7.0+, Chrome, Firefox, Edge, Safari (preview) and Opera all support various parts of WebAuthn [78, 79, 108]. Because this is a relatively

recent development, server compatibility is lagging behind that of browsers, as users can only log in to a select few sites using authenticators [99].

- *Mature:* Asymmetric cryptographic authentication has been in use for many years to authenticate servers to browsers. Using adapted schemes for client authentication is also fairly mature for remote shell authentication, among other uses.

- *Non-Proprietary:* Anyone can implement the WebAuthn spec. Note that companies who care about official certification can pay the FIDO Alliance to certify products [3], but certification is not necessary to implement the protocols.

As mentioned in Chapter 1, the FIDO Alliance [5] is a collection of hundreds of companies who share the goal of secure and usable authentication. One of the great strengths of the FIDO Alliance is the abundance of industry support for a movement to an improved authentication scheme. Boasting support from hundreds of companies from around the world, including many of the largest and most influential from tech and finance [6], the FIDO Alliance has the position, technical capabilities, and financial support to push standards like the WebAuthn protocol and affect things like *server and browser compatibility* directly.

2.5.4 Usability of the WebAuthn Scheme

We note a few usability aspects of this scheme:

- *Memorywise-Effortless:* The user has some choice about what type of second factor to use. Some prefer a PIN, while others will prefer a biometric. If the user chooses a biometric, he need not remember any secret.

- *Scalable-for-Users:* Because users can re-use the same phone and same second factor (PIN, fingerprint, etc) across all relying parties, this authentication scheme scales as well as most password managers and scales much better than do current schemes based on passwords alone.

- *Nothing-to-Carry:* If the user already carries a phone (as is increasingly the case), there is generally nothing else to carry.

- *Physically-Effortless:* This is a trade-off between usability and security that the WebAuthn scheme can leave up to the users and relying parties. If the user is required to give explicit consent for authentications, he will likely be required to physically interact with the authenticator. If user and relying party agree not to require physical effort, the user will likely concede authentications without explicit consent. That said, we can still reduce the physical effort required in a way that users prefer. When given multiple options for authentications, there is evidence that users tend to prefer the easiest option, and thus prefer the biometric in most authentication cases [49, 52, 61, 74, 111].

- *Easy-to-Learn-and-Use:* Thanks to the increased inclusion of biometrics and PIN unlock in popular devices [10, 11, 91], users are becoming more familiar with the use of these systems for authentication, reducing the learning curve and alleviating concerns about ease of use.

- *Infrequent-Errors:* Reliability of the log-in scheme is important for users, and biometrics are getting very good when compared to passwords [20, 27, 61, 74, 111]. In fact, false rejects (when the authentication system rejects the correct person) were highest for password website log-ins and password laptop unlock — up to four times as high for websites than fingerprints or physical keys [74].

Unfortunately, the WebAuthn scheme does have some usability drawbacks. In particular it is:

- ***Difficult-to-Upgrade*** Currently, the WebAuthn scheme makes upgrading to new authenticators arduous for users. For each relying party where a user has registered an old authenticator, the user will need to log in, delete the old authenticator and register a new one manually. Clearly, this doesn't scale well, but the experience of replacing

keys at each relying party can be different and difficult on its own. As a result, it seems unlikely users will go through the burden at all sites each time they upgrade devices, leaving passwords as the fallback or requiring account recovery for each account.

- ***Difficult-to-Recover:*** The WebAuthn scheme also makes it very difficult to recover from lost devices. For a user with only one authenticator and no password, the WebAuthn specifications do not give the user a way to recover account access.

2.5.5 Security of the WebAuthn Scheme

The WebAuthn protocol as specified [14] is more secure than passwords across the board [20, 66]. The protocol can provide all of the security benefits provided by password schemes (see Section 2.3.3) and further satisfies all the remaining criteria posed by Bonneau et al. [20]. Like passwords, a WebAuthn scheme is:

- *Resilient-to-Theft:* Although the user has a physical authenticator which is subject to theft, a thief would also have to obtain the second factor in order to compromise accounts. Attackers can feasibly obtain certain second factors, but it is up to the user to determine the level of security with which he is comfortable [36]. Because users, who often have misconceptions about the security [56], have the ability to choose factors with different levels of security and usability, researchers will likely have to monitor the ecosystem to ensure that widespread misconceptions don't lead to security breaches. Further, allowing users to easily revoke access to lost or stolen authenticators helps harden the system's resistance to theft.

- *Not-Reliant-on-Trusted-Third-Parties:* As with passwords, this scheme depends only on the prover (authenticator/phone) and the verifier (relying party server).

- *Explicitly-Require-Consent:* As mentioned above, this is a trade-off between security and usability. Users and servers can require explicit consent for authentications.

- *Unlinkable:* Authenticators should generate strong random keys for each relying party site, making credentials unlinkable by default.

However, the WebAuthn spec also exhibits the following additional properties.

- *Cannot be Copied by External Observation:* Given that the authenticator is performing the cryptographic operations out of view of the user, an external observer gains no advantage by observing a well implemented authenticator. Some attackers may be able to perform side-channel attacks to extract keys from poorly implemented authenticators [71].

- *Cannot be Copied by Internal Observation:* A well-implemented authenticator performs all operations in hardware or protected by a Trusted-Execution-Environment to protect credentials from internal observation.

- *Phishing Resistant:* The challenge-response nature of the authentication keeps each response fresh to prevent replay attacks. The browser and authenticator validate the origin and certificate of the relying party before sending credentials, thus preventing MITM attacks. The purpose of this scheme is to solve Phishing once and for all.

- *Resilient to Throttled/Unthrottled Guessing:* Because the authenticator randomly generates credentials from a large key space, they are resilient to even unthrottled guessing.

- *Unaffected by Leaks from Other Verifiers:* Because the authenticator automatically generates different credentials for each account, no two identifiers will be the same. Further, as discussed in Section 3, there are potentially recovery schemes that don't depend on other accounts, isolating each relationship with a relying party.

- *Resilient to Targeted Impersonation:* Credentials, by default, do not rely on the user's personal information. Even if the user decides to use personal information as a second

factor, an attacker would need to physically acquire the authenticator before using the credentials.

2.5.6 Summary

If the WebAuthn scheme is to match or exceed passwords in every aspect of Security, Usability, and Deployability, we as researchers and industry leaders must work to improve it in a few key places. In this section we have identified areas which must be improved in order to replace passwords with a modern authentication protocol such as WebAuthn. Those areas are as follows:

- *Server-and-Browser-Compatibility:* Achieving ubiquitous compatibility is very difficult because there will always be users who do not possess the necessary devices for the WebAuthn protocol, but the FIDO Alliance has made major strides in browser compatibility and is making strides to bring servers up to par. We think they are in a much better position to continue this push than we are as researchers.

- *Easy-to-Upgrade:* Upgrade in WebAuthn is arduous for users and the experience can differ greatly from site to site.

- *Easy-to-Recover:* The WebAuthn protocol does not provide any easy way for users to recover accounts or to migrate from one authenticator to another in general.

- *Pairing-Phone-and-Computer:* The current spec does not indicate *how* a relying party would initiate a password-less log-in notification on the user's phone. For existing computer-device pairings, the computer can notify the phone over an existing channel (ex: paired Bluetooth channel). However, for new computer-device pairings, the current specifications do not have a solution that allows the relying party to connect with a user's phone without: (a) requiring the user to execute a task or (b) leaking information about the log-in to potential attackers. Because there are methods for pairing

that already exist which will suffice at limited user burden, we do not believe this issue will prevent the adoption of WebAuthn. Given that FIDO has made progress on a more permanent solution to this problem [64], we think they are in a better position to continue this push at this time.

- *Security-of-Second-Factors:* Lastly, the WebAuthn specification allows for innovating and changing second factors. While users can explore the marketplace to select satisfactorily usable second factors, researchers will need to continue to provide constant guidance on the security of those factors. Like the previous issue, *pairing-phone-and-computer*, we believe that the increased usability and decreased attack interface of most available second factors likely indicates this will also not hinder adoption of WebAuthn. As such, we also propose this as future work, outside the scope of this Dissertation.

Based on the above analysis, I identify three remaining barriers to WebAuthn adoption. They are *Server-and-Browser-Compatibility*, *Easy-to-Upgrade*, and *Easy-to-Recover*. The FIDO Alliance and the W3C have made significant progress on server and browser compatibility, providing the WebAuthn API in both Windows 10 and Android 7.0+ and major browsers Chrome, Firefox, Edge, Safari (preview) and Opera [78, 79, 108]. They have also been working with relying parties to implement server-side WebAuthn protocols on dedicated apps. This Dissertation tackles the two remaining barriers: *Easy-to-Upgrade* and *Easy-to-Recover*.

Chapter 3

SOLUTIONS TO DEVICE UPGRADE AND DEVICE LOSS

Upgrading to new devices requires no additional action for password users. When someone buys a new device, they can simply use their existing username and password on the new device to authenticate online. However, for password users to *recover* access to an account they must perform a number of steps. Traditionally, a user must register an email address some time before recovery (1). Should the user then lose access to the account, he can request the relying party send a recovery email to that email address (2), access the email (3), and recover the account by registering new credentials (4). Note these four steps must be repeated for each account.

While this user experience is well understood, it has a couple of drawbacks. First, it leaks information about which accounts a user owns. For example, say a user stores the email user-recovery-email@gmail.com as the recovery email for his Facebook account user-account@facebook.com. If the user loses access to the Facebook account, he can request a recovery email. However, now Facebook can link the user's Gmail account with his Facebook activity and Google can link a user's Facebook account with his activity on Google controlled resources. Further, should an attacker gain control of the Google account, that attacker could simply run the account recovery protocol for the Facebook account and gain access to that account.

In the current WebAuthn specification, even without passwords the situation is arguably equally bad. For example, assume a user authenticates to all online accounts with his phone, as per the spec [14]. This is a nice user experience for *Registration* and *Authentication* but when the user loses the phone how does he recover? He cannot ask for an email from Facebook to restore access to a new authenticator because he won't be able to log in to

Gmail either, having also lost credentials for that account. Additionally, given the security and privacy vulnerabilities associated with using email as a backup, a solution that does not make security concessions would avoid using backup emails for account recovery.

This, among other reasons, is why the FIDO Alliance recommends that users register multiple authenticators for each account [42]. That way, even if a user loses an authenticator, they can use their other authenticator to access the account. To upgrade a device, users need to authenticate with one of their existing authenticators for each account at each relying party and then register their new device.

The main problem with this solution is that it scales horribly. Users with many accounts will have to spend a significant amount of time authenticating and registering for many different online accounts. As such, the discussion in this chapter focuses on scalability when analysing the existing solution space and when analysing proposed solutions. This chapter organizes the potential and existing solution space in Section 3.1 and defines the goals for any potential solutions in Section 3.2. Section 3.3 discusses some of the limitations and trade-offs of potential solutions and Section 3.4 presents a discussion of some example of those example solutions analyzed briefly within the framework developed in the rest of this chapter.

3.1 Solution Classes

From a user's perspective, when trying to transfer access to a new device, solutions fall into four main classes:

1. **Non-Recoverable** *The user cannot recover access to the account.*

   This is usually the worst case scenario, but may be intentional in some cases. If access to an account is tied physically to the old device, the new device will never be able to authenticate with the account.

2. **Single additional action per account per authenticator**: *The user has to perform an action for each account to transfer access to that account to the new device.*

This is the status quo when users get locked out of their password managers or forget passwords for individual accounts. Usually the user needs to run a recovery procedure for each account that uses some of the following techniques:

- Require answering personal knowledge questions, tying the strength of the authentication to the user's knowledge of the account holder's identity.
- Send a code to another device, tying the strength of the authentication to the security of another device.
- Send a recovery email to another account, tying the strength of the authentication to the security of another account.
- Tie the authentication to the user's real world identity, for example, by requiring identity verification.
- Trust a federated proof, tying the strength of the authentication to the trust of another entity who can, for example, check the user's identity.
- Trust contacts, tying the security of the authentication to a connection of potentially many other accounts.
- Generate trusted backup codes the user can use to gain or reset access.

However, note that many of these recovery processes are still phishable or have the potential to introduce a weaker link into the WebAuthn ecosystem. Further, trusting other entities such as phone providers, email providers, identity providers, federating entities, contacts, etc provide information that can help relying parties or other attackers link between accounts and compromise the privacy protections of WebAuthn.

Therefore, the FIDO Alliance suggests preparing for recovery by registering multiple authenticators, keeping the security of any authentications tied to un-phishable cryptographic credentials on trusted devices which can register unique credentials at each relying party. However, such a recovery procedure still requires a single action per account [42].

3. **Single additional action required per authenticator**: *The user has to perform a single action to transfer access to a new device for all accounts*

   In the password world, to transfer to a new device the status quo for most password managers requires a simple one-time set up of the new device. That set up could be, for example, downloading an app or browser extension, authenticating and/or decrypting an encrypted blob, and giving the app the correct permissions to operate on the new device. Additionally, the user must set up any device-specific settings, including those for future recoveries. For many password managers, recovering a master password requires a single action to recover access or transfer access to all accounts. For example, Last Pass allows users to set up a number of recovery options including SMS, mobile recovery, trusted Administrator, or one-time recovery password [13, 45]. There are other cases where users can run a recovery or transfer procedure by copying over seed data for authenticator apps to transfer second factor information as well [105].

   Note that even though this is only a single action during the setup of a new device, the transfer or recovery can still be arduous for users if it requires too many steps or is hard to understand. For example, a recovery procedure that requires a user to manage multiple devices, tweak obscure settings, and follow multiple steps may cause too many errors to be an effective set up procedure.

4. **No additional action required**: *The user doesn't need to change any behavior to transfer access to a new device*

   In a password-based world, this is the case for device recovery and device upgrade. A user who replaces an old or lost device does not need to do anything extra to use it with online accounts. However, the net user burden may still be higher in a password-based system than the one-time cost of a recovery or transfer with the much simpler authentication flow of WebAuthn. As seen in Chapter 4, it may be possible for some users to leverage the initial setup process for certain new devices to make the

recovery/upgrade process effectively require no additional user action.

Ideally, solutions to *Device Upgrade* and *Device Loss* would not require any extra user action. However, as seen in 3.3, sometimes this is not possible and solutions must make trade-offs. In order to improve upon the password ecosystem, solutions presented in this dissertation either require a **single additional action per device**, or **no additional action**.

3.2 Goals

To the extent possible, any solutions allowing users to upgrade devices or recover from loss should preserve the benefits of the existing WebAuthn specification. As in Section 2.5, we break those benefits into Deployability, Usability, and Security.

Deployability In Section 2.5.3, we list the numerous usability benefits WebAuthn affords. Any solution should preserve these properties to the extent possible.

- **Accessible**: Users should still be able to use devices with many form factors and second factors, instead of being tied to a particular type of device.

- **Negligible Cost per User**: Solutions should not cost users significantly more than the existing WebAuthn ecosystem already costs. Ideally, a solution would add no cost.

- **Server and Browser Compatible**: Solutions would ideally be compatible with existing devices and code. However, in the event that specification changes are necessary, those changes should aim to minimize changes and additional hardware from end users and relying parties.

- **Mature**: Solutions should prefer mature technologies instead of introducing unvetted or untested systems.

- **Non-Proprietary**: Anyone should be able to implement any aspect of the recovery procedure.

Usability In Section 2.5.4, we list the numerous usability benefits WebAuthn affords. Any solution should preserve these properties to the extent possible.

- **Memorywise-Effortless**: Solutions should preserve the user's choice of second factors and retain the possibility of a diverse ecosystem of devices. Should users choose a biometric second factor, transfer and recovery would ideally not require them to remember anything.

- **Scalable**: Solutions should scale well to many different accounts on many relying parties.

- **Nothing to Carry**: Solutions should not require users to carry more devices than they already have to carry in the WebAuthn ecosystem. This means that if they already carry an authenticator for registrations and authentications, they should not have to carry another one to enable device upgrade or recovery from device loss.

- **Physically Effortless**: To the extent possible, users should retain the ability to use their selected level of physical effort.

- **Easy to Learn and Use**: Any solution should not be more difficult to learn or use than a standard WebAuthn authentication or registration.

- **Infrequent Errors**: Any solution should not cause more errors than a standard WebAuthn authentication or registration, and should not cause errors in any future authentications or registrations. Ideally, any errors that do arise should be easily and logically resolvable by the user, but errors would ideally not be allowed to occur.

Security As we mention in Section 2.5.5, the WebAuthn protocol is more secure than passwords in all the criteria posed by Bonneau et al. and also provides additional privacy protections that the password ecosystem doesn't provide [21]. Solutions to device upgrade and device loss should preserve these properties in all steps of the solution's protocol without weakening future or past authentications or registrations.

- **Resilient to Theft**: Solutions should still protect protocols with local authorization gestures that the users can select.

- **Not Reliant on Trusted Third Parties**: Ideal solutions would only have to trust the prover(s) and verifier for each account. As shown in Chapter 6, in some cases it may be acceptable to trust a third party for availability as long as entities do not have to trust the third party for any other reason. That is not to say that solutions allowing for third party contributions should be excluded. For example, it would be acceptable to have an option where users can trust a third party to provide recovery services if the third party provides value to the user, but users should not be forced to use that or any other entity.

- **Explicitly Require Consent**: Users should still have the ability to force reliance on their consent during the upgrade/recovery and during future authentications and registrations.

- **Unlinkable**: Solutions should not render future or past credentials linkable by any party.

- **Cannot be Copied by External Observation**: Solutions should not weaken the protections against copying by external observation for any attacker, nor should they present new opportunities for external observers.

- **Cannot be Copied by Internal Observation**: Solutions should be implementable within secure elements.

- **Phishing Resistant**: No part of the solution should be phishable, and future authentications or registrations should remain un-phishable as well.

- **Resilient to Throttled/Unthrottled Guessing**: Solutions should not rely on guessable parameters or keys that travel between devices.

- **Unaffected by Leaks from Other Verifiers**: Solutions should not allow leaks from any entity to affect the recovery or upgrade of any other entity.

- **Resilient to Targeted Impersonation**: Solutions should not rely on the user's personal information outside of using that information for a local authorization gesture, as in current WebAuthn registrations and authentications.

This document presents potential solutions which aim for **no additional action required** when possible, and a **single additional action required** in the worst case, while minimizing concessions to the existing deployability, usability, or security aspects of the underlying WebAuthn scheme.

3.3 Limitations and Trade-offs

Given these goals, this section explores some of the trade-offs and limitations that arise, preventing proposed solutions from satisfying all goals stated in Section 3.2.

3.3.1 Extra Device vs. Security

Because the WebAuthn ecosystem relies on authenticators for credential management, when a user loses an authenticator, he cannot simply restore access from his own memory. Instead, he will likely need to rely on one of two classes of recovery entity in order to restore access: an *extra device* or an *extra service*. As per the FIDO Alliance recommendations, a user can register two devices at every relying party [42]. This requires the user obtain and manage an extra device, but when the user loses one of the devices, he can still use the other to

log in and set up a new authenticator. Variations of this scheme that improve usability are possible, but all still require the user to acquire two devices.

Should the user prefer to only have a *single* authentication device, he will be left with no devices if he loses that lone authenticator. As such, he will likely need to rely on an extra service to help restore access to accounts. The natural place for such a service is the cloud, thanks to the reliability and availability the cloud provides. However, a user who stores recovery information in the cloud will have to authenticate with the cloud provider without an authentication device. As we cover in Chapter 2, authenticating without a device can be problematic, likely relying on less-secure authentication mechanisms like passwords. However, given that the users are not likely to need to recover nearly as often as they authenticate, they could be amenable to more stringent and secure mechanisms for authenticating to recovery services. Such stringent policies could mitigate the security risks of authenticating to an extra service without an authentication device. For example, if a user loses his phone and needs to restore from the cloud, he can either physically go to a notary who can authorize such a recovery attempt, or he can split key shares across many already-owned devices or associates who can help him authenticate with the cloud provider. However, as discussed in detail in Chapter 2, these authentication mechanisms are not likely to be as secure as WebAuthn.

3.3.2 Storage Overhead vs Usability

As discussed in Section 3.3.1, a user will likely have to rely on a recovery entity (either a recovery device or recovery service) to restore access to all of his accounts. We can segment our proposals into two categories: 1. Recovery solutions where the recovery entity only stores a single key and 2. Recovery solutions where the recovery entity contains a key for every account (similar to the Transfer Access approach from Section 4.1). We believe that the latter category suffers from a *Storage Overhead* versus *Usability* trade-off. The argument proceeds as follows:

Assume a user has a matching set of credentials on both his primary (and only) authen-

ticator and his backup entity. In this case we say there is no storage overhead because the number of credentials on the backup entity is exactly the same as the number of credentials on the authenticator. The next time the user registers a new account with a relying party, his authenticator will generate a new key pair. To maintain this 1:1 ratio and be able to help the user recover his account, the backup entity needs to be made aware of that registration. If this does not happen *immediately*, the user runs the risk of losing his authenticator before the backup entity learns about the new credentials, meaning he will lose access to the recently created account. If the user relies on a *recovery device* instead of a recovery service, this means that registrations now require an **additional action per registration**, adding a potentially prohibitive number of user actions to the existing flow and possibly forcing the user to carry two authenticators instead of one. Even in the event that a user shares a seed between recovery and primary authentication devices, the recovery device eventually must be made aware of each registration in order to recover, and we again risk account loss due un-synced credentials. Further, if the user must carry two authenticators to deal with registrations, the likelihood of losing both authenticators increases, undermining the value of a *recovery device*.

As an alternative, a user can have the backup device generate extra keys ahead of time and give them to the primary authenticator. That way, when the primary authenticator registers with relying parties in the future, it can also register a backup key. Chapter 5 is an example of such a solution. However, note that with this type of pre-loading, the backup device will still not know which of those pre-generated keys have been registered with a relying party unless it is contacted during or after a registration. If the primary authenticator gets lost before contacting the backup device, unregistered keys become a *Storage Overhead* - keys that must be kept around in case they were used, even though they weren't. Though a storage overhead may always be necessary in cases where the recovery entity is unaware of recent registrations, Chapter 6 presents a potential solution that can help mitigate the overhead and user burden by utilizing *Online Recovery Storage*.

3.3.3 Single Key vs Security

All proposals that use a single key violate at least one of the security goals of the existing WebAuthn scheme. A single key can be either a symmetric or asymmetric key. Section 3.4 shows how each of the example single-key proposals violates a security goal.

3.4 Example Solutions

This section briefly discusses four example classes of proposals that could allow users to recover from device loss. As stated in Section 3.3, there are trade-offs and limitations of each listed scheme. This discussion motivates the choices made in the solutions presented in Chapters 4, 5, and 6.

3.4.1 Example solution with a storage overhead

One example solution could utilize a backup device to store a backup credential for each of the user's online accounts. From a technical standpoint, the key insight is to use credential chaining similar to that of the Transfer Access Work explained in more detail in 4.3.2.

In this type of solution, a user can sync a backup device with each new primary authenticator during the setup phase for that authenticator. During this setup the backup device generates recovery key pairs so that, in future registrations, the primary authenticator can register both one of its own keys and a backup key. Unfortunately, for this to be possible, the primary authenticator must store those keys until it needs them, causing a storage overhead. If it did not store them, it would have to contact the backup device before each registration to get a backup key, effectively forcing the user to carry two devices.

This solution allows users to effectively register multiple authenticators for all accounts without having to carry multiple authenticators, at the cost of a storage and computation overhead on the authenticators themselves. Further, from Section 3.2, this solution adds complexity to the devices in the ecosystem and the relying parties who have to handle these messages. It also requires users to acquire and manage another recovery device, negatively

affecting the deployability and usability of the scheme. However, as discussed in more detail in Chapter 5, such a solution can be implemented without sacrificing security properties of the WebAuthn scheme and reduces effort for users to a **single additional action per device**.

3.4.2 Example solution that trusts a third party

If the user can trust a third party, he can implement recovery schemes similar to the "Last Pass" model [85]. In this type of scheme, a user relies on a third party to store credentials and simply authenticates with the third party to receive the credentials necessary for authentications. While this requires the user to frequently visit a third party, such a requirement is not likely prohibitive for those using the third party for *web* authentications. These schemes would allow for user experiences and trust models similar to that of existing schemes (for those users who use password managers).

Recovery in this ecosystem does not require the user to migrate keys for all accounts. Rather, the user only needs to ensure access to the trusted third party on the new device. The user can achieve this with or without another authenticator, but as discussed in Section 3.3.1, authenticating without a device can be problematic. If the user does use a device, he only needs to use that device for authentication with the trusted third party. The user now simply needs to manage recovery for a single domain, and can use either of the previous solutions in Section 3.4.1 or Section 3.4.3. Again, as mentioned in Section 3.3.3, the user has a single recovery key, but violates a security goal by relying on a trusted third party.

The main drawback of such a class of solutions is the reliance on a third party, violating one of the security goals from Section 3.2. However, it is possible to combine schemes to reduce the *level* of trust in such a third party and offload that trust onto devices. For example, Chapter 6 presents such a solution that only trusts the third party for *availability* by instead trusting authenticators and backup devices to preserve the security and privacy properties of WebAuthn.

3.4.3 Example solution that foregoes unlinkability

Another potential solution can remove the storage overhead by using a single key pair that authenticates the user for all accounts. In the simplest of scenarios, a user has *PrimaryAuthenticator1* and a backup device, each which only use a single key pair. The primary authenticator must be aware of the backup device and store the backup device's public key (and potentially attestation certificate, etc) in addition to its own. Whenever the primary authenticator registers an account, it uses its own single long-lived key and also registers the backup device's key. If the user loses his primary authenticator, he simply gets a new one, *PrimaryAuthenticator2* and runs a setup protocol with his backup device, which includes a delegation from the backup key to the long lived key on *PrimaryAuthenticator2*. Upon his next log-in, *PrimaryAuthenticator2* will serve this delegation certificate just as in the Transfer Access Protocol (see Section 4.1), instructing the server to remove access to *PrimaryAuthenticator1* and give access to *PrimaryAuthenticator2*.

Schemes can be made to support current authenticators and the current WebAuthn model by using long-lived unique attestation certificates rather than long-lived authentication credentials. However, I argue that since either type of solution still does not provide unlinkability, the system should be made as simple as possible (a single public key for each device, instead of a key for each account per device with a linkable attestation). This solution also requires users to acquire and manage a second device, introduces additional complexity for authenticators and relying parties, and likely requires users to type usernames to authenticate with one of many accounts at a single relying party. However, it also reduces effort for users to a **single additional action per device**, and reduces the number of credentials on each authenticator to a single key, potentially reducing the net complexity of the authenticators. Further, users can still achieve unlinkability by managing multiple authenticators, for example for different classes of accounts.

Because this does not preserve the privacy properties of WebAuthn, this dissertation does not explore further options foregoing unlinkability.

3.4.4 Example solution that copies keys

Solutions can potentially achieve zero storage overhead and preserve unlinkability without requiring a trusted third party. To do so, the WebAuthn ecosystem would need to make slight changes to the existing structure. As in Lang et al. [66], the FIDO U2F ecosystem allows for a key-wrapping infrastructure where the authenticator itself does not store its own credentials. Rather, it has a symmetric key stored in hardware that it can use to unwrap and use credentials served by the relying party.

When users register an account, the authenticator creates a key pair, registers the public key with the relying party, encrypts the entire pair with its own symmetric key, and serves that blob to the relying party for storage. When the user returns to the relying party to log-in, the relying party serves the encrypted blob to the authenticator which un-encrypts the key pair and uses it to perform a standard asymmetric challenge-response authentication.

If the entirety of the authentication ecosystem works on this model, all a user would need to do upon losing his authenticator is to restore the wrapping key from his old authenticator. This can be done via secret sharing [94], or from a backup entity that can restore access to other authenticators via a secure copy mechanism.

We note that again, the user only needs to manage one symmetric key as in Section 3.3.3. However, this scheme relies on the ability to extract secrets from authenticators and duplicate those secrets on new authenticators. Because attackers would also like to do the same, this violates the security goal of having credentials that cannot be copied. Revocation of access from previous authenticators is also potentially difficult without reliance on non-mature technologies.

Because this violates the security properties of WebAuthn, this dissertation does not further explore options that copy keys or data that can be used to reconstruct keys.

3.5 Summary

This chapter presents an overview of the current solution space, and many of the potential ways to solve problems related to device loss and device upgrade. It provides concrete goals for solutions and identifies some key trade-offs inherent in certain solution classes. Finally, it provides some high-level example solutions as thought experiments that exemplify some of those trade-offs. There are many potential trade-offs in usability, security, and deployability when designing and selecting protocols to enable device upgrade and recovery from device loss. However, the WebAuthn protocols derive their security from the strong security properties of asymmetric encryption schemes. As such, the proposals presented in the remainder of this document seek to provide solutions to *Device Upgrade* and *Device Loss* that preserve WebAuthn's security by leveraging the properties of asymmetric encryption and credential binding. Relying on other types of recovery schemes that subvert the cryptographic keys trusted by the server introduce potentially weaker links into the system.

Chapter 4

DEVICE UPGRADE: TRANSFERRING ACCESS IN THE FIDO ECOSYSTEM

This chapter presents the *Transfer Access Protocol* — our proposal that would allow users to upgrade devices in the FIDO U2F ecosystem. Although implemented on the u2f protocol, much of the concepts here apply to the current WebAuthn standard, as seen in Chapters 5 and 6. The *Transfer Access Protocol* leverages credential binding — tying trust in new credentials to the trust in old credentials using asymmetric encryption. Device manufacturers and relying parties can implement the changes necessary for the *Transfer Access Protocol* on existing hardware to allow users to upgrade devices seamlessly with potentially no effect on their ability to log into any existing accounts. This concepts were originally described in [103], done in collaboration with Alexei Czeskis (Google) and Tadayoshi Kohno (UW).

4.1 Introduction

Users who employ phones as authenticators will inevitably *upgrade* phones. As a result, the following scenario is likely to occur regularly: A user is using a phone as an authenticator but wants to replace that device with a newer model, ceasing to use the old phone. The user, therefore, needs to set up the new phone as an authenticator and remove credentials from the old phone.

In the current password-based ecosystem, *upgrading* to a new device doesn't require anything in particular from a user, since they can simply re-log in with the new device. Some relying parties who rely on device fingerprinting signals may ask for another factor, such as a CAPTCHA or code, but the minimum burden to a user is simply re-typing the username and password.

We propose and implement a solution — the Transfer Access Protocol — based on the FIDO U2F standard, which does so without requiring *any* additional user action and while preserving all existing security and privacy benefits of the WebAuthn protocol. Though the discussion in this chapter is based on the FIDO U2F protocol, many aspects of the solution are applicable to WebAuthn and inform our discussion of recovering from device loss in Chapter 5 and Chapter 6.

4.2 Goals

The goal of this work is to transfer account access from the authenticator app on the old phone to the authenticator app on the new phone. However, we would like to do so while preserving desirable properties of the existing FIDO authentication scheme. Here we discuss our primary goals in detail and some of the challenges that arise. In short, we do not want the addition of a Transfer Access Protocol to affect the user experience or the security and privacy properties of the existing FIDO authentication scheme.

4.2.1 The User Experience

Ideally, when a user buys a new phone, transferring authenticator access should work seamlessly, not adding or changing steps for the user either when they set up the new phone or during the next log-in. For example, during the initial phone setup, one possible implementation could simply attach the Transfer Access Protocol to the Tap & Go [43] feature in Android 5.0+. In the current implementation of Tap & Go, when a user first boots up a new phone, they see the screen sequence from Fig. 4.1.

If desired, we could simply ask the user whether they would like to use the new device as an authenticator and remove access from the old phone. In that case, the user would see an extra screen as in Fig. 4.2. We stress that this is just an example implementation and is not necessary for the Transfer Access Protocol described in this chapter. One could, for example, choose to make the Transfer Access Protocol completely transparent during the Tap & Go procedure or make the process independent of Tap & Go by processing the Transfer Access

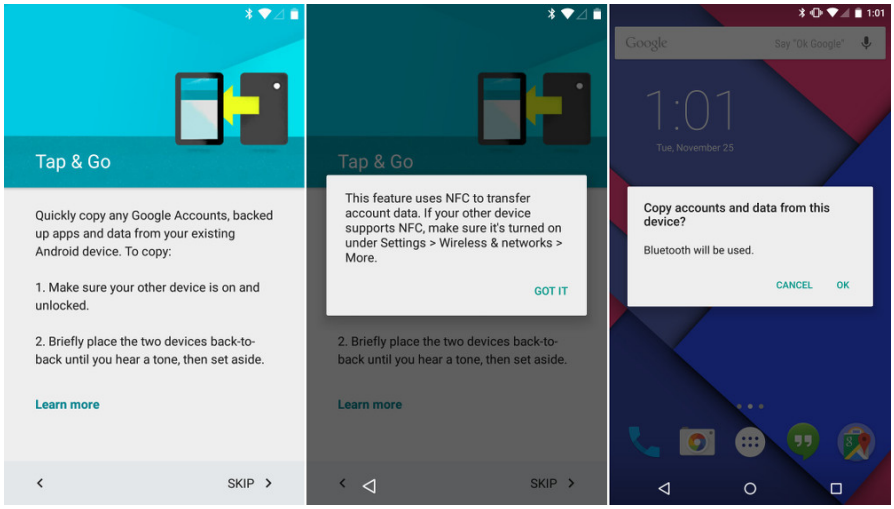


Figure 4.1: The stock Tap & Go implementation for Android 5.0 and up. This allows a user to quickly set up a new phone by transferring apps and data from the old one. Screenshots from Droid-Life [62].

within the authenticator app itself instead. In principle, none of the concepts discussed here *require* any extra steps.

After setting up the new phone, users will navigate to sites (or open each native application) as they did before. If a user is accustomed to seeing a log-in screen as in Fig. 4.3, we would not like to change that user experience. In fact, we envision that the Transfer Access Protocol would keep the cryptographic transfer transparent to the user as it does during normal FIDO authentication so the next time the user logs in on the new phone, the user experience does not change at all.

4.2.2 Security

For the Transfer Access Protocol, we seek to preserve the security and privacy properties of the existing FIDO authentication scheme. For example, FIDO authentication can prevent web and network attackers from phishing or copying credentials, defend against Man-In-The-Middle attacks, provide clone detection, allow relying parties to revoke access or prevent

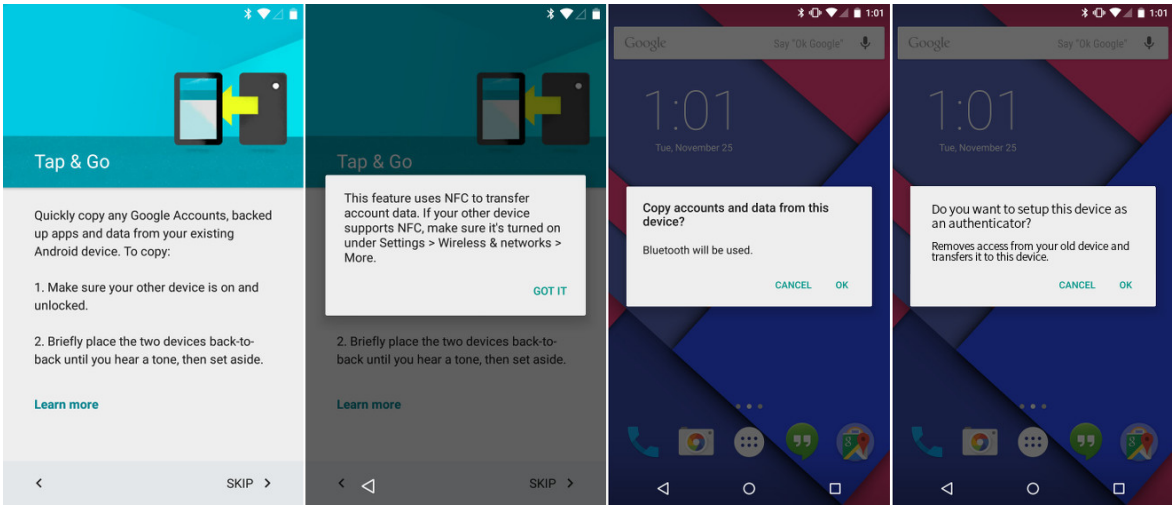
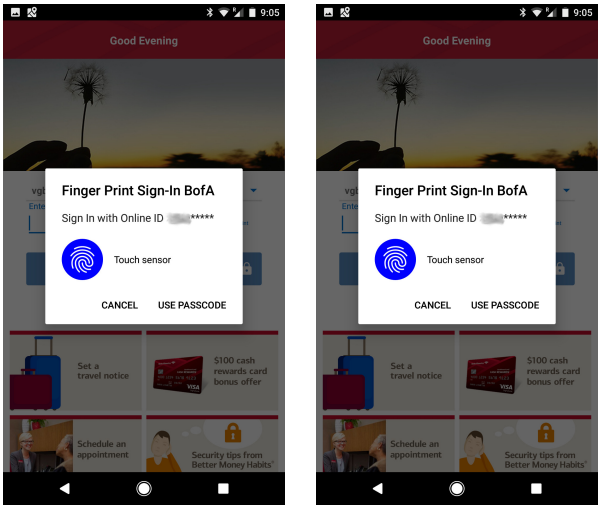


Figure 4.2: A potential user flow with the Transfer Access protocol added. Original screenshots from Droid-Life [62].

registration of untrusted hardware, and prevent relying parties from colluding to link user accounts. However, a number of attacks are still out of scope. For example, the FIDO authentication scheme does not explicitly protect against attackers who can simultaneously attack network traffic and the local wireless environment, attackers who can compromise a user's PC and personal device, nor would it protect against a malicious authenticator or operating system compromise. Because the Transfer Access Protocol transfers FIDO authenticator access, rather than performing an independent security and privacy analysis of each piece of this protocol, we aim to design a protocol that introduces no additional vulnerabilities. Throughout the description of this work, we will discuss some of the relevant and interesting decisions we make through the lens of concerns raised in previous papers on asymmetric authentication schemes. As in Section 2.2, we would like to uphold the following properties (adapted to the Transfer Access protocol):

- Phishing: The protocol should not be phishable, nor should the resulting credentials.
- Defend Against MITM: Attackers who Man-In-The-Middle the connection, for exam-



Before After

Figure 4.3: An example user experience for an app log-in. This screenshot is from the Bank of America app on a Google Pixel. This figure also appears in Chapter 5 as Figure 5.1

.

ple between the browser and relying party server, should not gain an advantage by attacking during any step in the Transfer Access Protocol.

- Session-Duplication: The protocol should not aid in the ability for stealing credentials to result in session-duplication, for example, by exposing long-term cookies or passwords.

- Prevent Session Riding: The protocol should not allow an adversary to gain access to an existing session or to a future existing session.

- Trusted Hardware: The protocol should allow the relying party to verify that it trusts the new hardware before allowing access.

- Non-Linkability: Credentials should be site-specific by default so that colluding relying parties can not link credentials to users across sites.

- Detecting Clones: The protocol should allow for the continued detection of potential authenticator clones by keeping a counter.

Threat Model To determine whether the additional steps in the Transfer Access Protocol uphold these goals, we analyze each step using a threat model based on previous works done in this space. For example, we will use the attackers mentioned in [66]:

- Web Attackers who can phish for credentials by setting up forged web pages, including correct TLS certificates for victim sites.

- Related-Site attackers where users may have reused the same credentials as the victim site.

- Network Attackers who can MITM connections with correct certificates or decrypt traffic.

- Malware Attackers who can install malicious applications or take over benign applications.

However, we will also consider other potential attackers through whom we can demonstrate some of the strong security and privacy properties of the FIDO authentication scheme. For example, Site Attackers who can dump logs and credentials from the victim site may be able to reveal user passwords, and adversaries who are able to gain physical control of devices at later or earlier times (OEM vs. repurchasing an old phone) can raise some interesting concerns. We also place certain attacks out-of-scope. For example, we do not consider protecting against a malicious FIDO application as the underlying FIDO scheme would not be secure anyway.

We believe that the addition of this work to the FIDO authentication scheme would help secure potential vulnerabilities that result when users transition to new devices.

4.2.3 Goal Conditions

Before diving into potential workable solutions for transferring access to a new authenticator, we discuss the assumptions and the properties constituting goal conditions for the Transfer Access Protocol. To start, we have the following assumptions:

Assumptions:

- The user has access to an old phone (A) and new phone (B)
- Phone A has keys and associated metadata, each associated with an account
- Phone B may or may not have existing keys
- Phone A and Phone B can create a "secure channel"
  — This secure channel is out of scope for the Transfer Access Protocol. We assume that this channel can only be set up by a legitimate user who explicitly allows the transfer of access from Phone A to Phone B. For the purposes of this paper, we assume this channel allows communication between the two phones that is resilient to all possible attacks, including eavesdropping and Man-In-The-Middle attacks.

  Though this is clearly a concern for FIDO/WebAuthn authentications, the current protocols do not mandate how devices communicate outside of core authentication operations between an authenticator, browser, and relying party. The protocols instead leave the trust in these implementations up to the relying party and end users via hardware attestations. For example, a third party (or the relying party itself) can vet the secure channel set up by android phones in the Tap & Go procedure to verify it can be trusted. We support the decision to leave this to relying parties and users and thus, do not try to secure this part of the protocol explicitly in this proposal.

Target Goal Conditions (for each transferred account): At the conclusion of this protocol, we expect the following properties to hold:

- Phone A has deleted the "transferred key".
- Phone B has the "transferred key"
- Phone B is logged in to the relying party.
- The relying party removes Phone A's access
- The relying party adds access for Phone B so that it will be able to authenticate in the future using standard FIDO authentication.
- **Security Goals:** Throughout each step of the procedure, we expect the Transfer Access protocol to give attackers no advantage in attacking the FIDO authentication scheme.

4.3 Solutions

The goal of this work is to transfer access from an old phone, Phone A, to a new phone, Phone B, while preserving the usability, security and privacy properties of the existing FIDO authentication scheme. In the current system, the solution requires the user to log-in to each site with Phone A, register a new set of keys for Phone B, remove Phone A's access at the relying party, and delete keys on Phone A (or factory reset the phone). Clearly, this adds multiple steps for the user for each existing account, but worse, the user may not have any indication as to how many or which sites require new credentials. This section builds from a simple solution, solving remaining problems until finally reaching the Transfer Access Protocol. This protocol satisfies the stated goal, allowing users to upgrade to new devices without requiring any additional steps for each account, changing the existing experience of authentications or registrations, or degrading the security and privacy properties of existing FIDO authentication schemes.

4.3.1 Simple Solution: Copying Keys

A straightforward solution that simplifies the experience of moving to new devices and eases user burden could merely copy the authenticator data from the old phone to the new phone. However, this violates the security properties of the FIDO protocol in that the relying party (example.com in the example above) would not have a chance to verify the new hardware. Further, if keys are stored in a secure element or trusted execution environment, the OS may not be able to copy them at all. If the OS could copy credentials, it stands to reason that malware could potentially extract keys as well.

4.3.2 Chain of Trust

As such, we propose a system that utilizes a secure channel between two phones (Tap & Go, for example, establishes a secure wireless channel between the new and old phones) to sign a new set of credentials with the old trusted credentials. This creates a chain of trust since the relying party already trusts the old private key.

To create such a set of credentials, Phone A will inform Phone B which accounts the user would like to transfer over the secure channel, at which point Phone B will generate fresh key pairs for each of those sites. Phone A also must send metadata uniquely identifying each key so that when Phone B sends back its new public keys Phone A knows which of its private keys to use to create signatures. Phone A then signs over each of the new public keys and sends those signatures back to Phone B.

Such a signature scheme solves a number of the problems above with the current and simple solutions. Namely, it can be done on initial setup without requiring a user to visit every site and it does not require copying credentials — a poor security practice for private keys. The scheme requires two stages. In Stage 1, as described, Phone A and Phone B communicate to exchange necessary information and generate the required signatures. In Stage 2, Phone B negotiates with the relying party to provide assertions that verify trust in the new credentials. Fig. 4.4 shows the two-stage nature of the Transfer Access Protocol, described

in detail in Section 4.4.1. However, a signature that simply delegates access from Phone A's public key to Phone B's new public key (even while providing hardware attestations for the new phone) still sacrifices a number of security properties provided by the existing FIDO protocol. In the following sections, we discuss how to mitigate these problems.

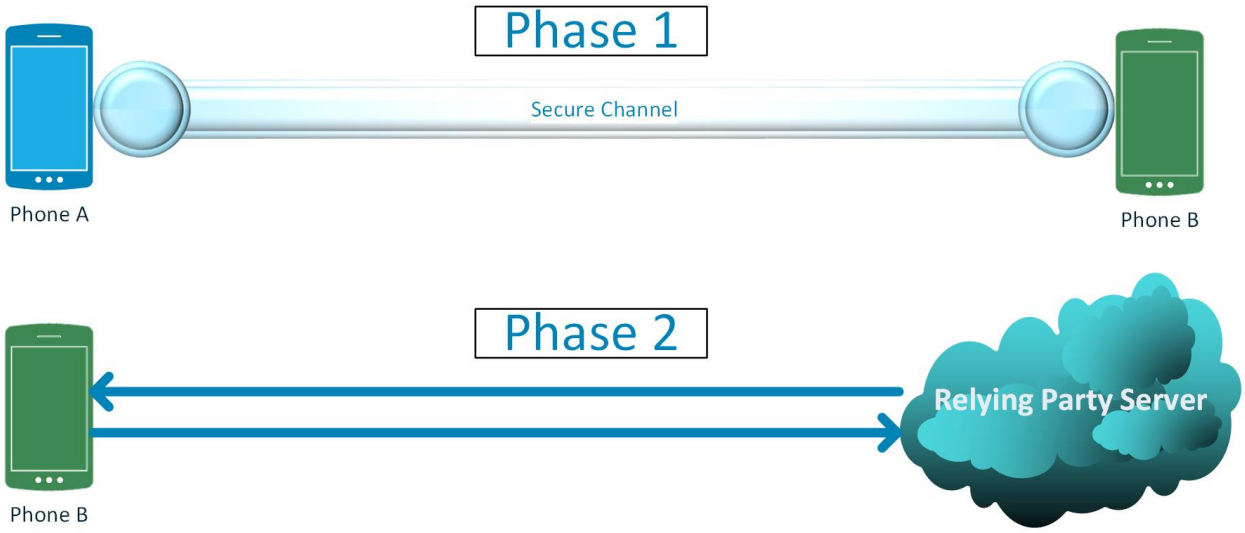


Figure 4.4: The Transfer Access Protocol requires two stages: In Phase 1, The old phone (Phone A) communicates with the new phone (Phone B) over a secure channel. In Phase 2, Phone B takes the results of that communication and delivers them to the Relying Party server.

4.3.3 Components of a Transfer Access Message

In the FIDO scheme, relying party servers communicate with authenticators through the browser. The browser can instruct the FIDO authenticator to respond with one of two messages: 1) A registration response or 2) an authentication response. We would like to take the necessary properties from each of these messages to craft a third response, enabling Transfer Access.

Because we do not want to change the user experience, the user's next log-in should serve as both a registration/enrollment for Phone B's new credentials and an authentication. We briefly discuss the registration and authentication messages crafted by the FIDO authenticator, with a focus on the security properties provided by each component of each message.

Registration/Enrollment During the registration, the relying party (through the browser) asks the authenticator to create a new asymmetric key pair and associate that pair with the relying party. That request contains a challenge parameter which the authenticator can use during the creation of its response. The authenticator creates a certificate in response and sends it to the server so that the relying party can store the necessary credentials for future authentications. The current components of a FIDO registration sent by an authenticator are:

- *Message Header* — Allows for setting flags that can indicate message type, for example Enrollment, Authentication, or Transfer Access.

- *Metadata* — Allows the client and server to efficiently look up keys and binds each key to a specific account.

- *User Public Key* — This is the new public key to be enrolled.

- *Attestation Certificate* — Allows the server to decide whether it trusts the hardware. Attestations are batched by device, each device containing a certificate, public key, and matching private key.

- *Challenge* — Contains a nonce to make each registration unique so that it can not be reused. For example, this prevents an attacker from re-registering a previously registered and removed key — for example, after a user realizes a key is compromised and removes it from an account.

- *Signature* — Proves ownership of the attestation private key so that the server knows the device matches the above Attestation Certificate. This prevents an untrusted device from falsely providing the Attestation Certificate of a trusted device in order to enroll a new private key.

Authentication When the relying party would like to authenticate an already-registered authenticator, it crafts a request containing a challenge and some key metadata for a previously registered key. The authenticator uses this information to look up the corresponding credentials and craft an authentication response that can convince the relying party to authorize the user. The current components of a FIDO authentication sent by an authenticator are:

- *Test of User Presence* — Requires the user to authorize the authentication, preventing attacks relying on remote surreptitious activation of the authenticator.

- *Counter* — Allows for clone detection. In the case of a cloned authenticator, the server will see consecutive log-ins that don't increment the counter correctly.

- *Metadata* — Binds the credential to the relying party, allowing the authenticator to efficiently look up the key and preventing attacks which seek to determine if some other key is present on the authenticator.

- *Challenge* — Contains a nonce to make each log-in unique, preventing replay and phishing attacks.

- *Signature* — Proves ownership of the private key, the basis for authentication.

Transfer Access We would like to preserve each of the security protections afforded by the components of the existing registration and authentication messages. To this end, the Transfer Access Protocol should include the following in response to an authentication request from the relying party:

- *Message Header* — We suggest using one of the available bits to inform the server that the message is a Transfer Access Message.

- *Metadata* — Allows the client and server to efficiently look up keys and binds each key to a specific account.

- *New Public Key* — The new public key to be enrolled by Phone B.

- *Challenge* — This makes each registration unique, preventing replay and phishing attacks.

- *New Attestation Certificate* — Allows the relying party to determine whether it trusts the new hardware.

- *Counter* — Notifies the relying party in the case of a cloned authenticator. Given that this is the first log-in on the new device, we don't think it necessary to continue to increment the old authenticator. As such, we set this counter to zero, which will alert the relying party if there is a clone in future log-in attempts.

- *Signature (Authentication)* — Proves ownership of an authorized private key so that the user can automatically log-in after completing the Transfer Access Protocol. Recall that this Transfer Access Response gets sent in response to an authentication request, so the user expects to log-in.

- *Signature (Attestation)* — Proves ownership of the new attestation private key, so that the server knows the credentials have been created by a device with a matching Attestation Certificate. This prevents an untrusted device from falsely providing the Attestation Certificate of a trusted device in order to enroll a new private key.

Notably absent is the Test of User Presence. Recall that this field prevents attacks relying on remote surreptitious activation of the authenticator. Because we assume that setting up

a secure channel requires user authorization and that the user intends to move from Phone A to Phone B permanently, we deem the Test of User Presence unnecessary for the Transfer Access Protocol. However, one could easily add it to the protocol when the authenticator delivers the Transfer Access response containing the above fields to the server, verifying user presence for that session.

4.3.4 Creating a Chain Through Multiple Devices

In Section 4.3.2, we discuss the two-stage nature of the proposed protocol. Although the user experience won't change for sites which the user visits regularly, the user needs to visit and log-in to each relying party with transferred credentials in order to complete the Transfer Access Protocol for each of those credentials. Though this may be reasonable for most sites, it is feasible that users will transfer to yet another new phone (say Phone C) before logging in to less oft-used sites on Phone B. In this case, we would have a situation where Phone B tries to transfer access to Phone C without first registering its credentials with the server. When Phone C finally does visit the relying party and delivers the Transfer Access credential generated by Phone B, the server will not recognize those credentials and will reject the transfer. To solve this problem we propose a protocol that allows for chaining of Transfer Access credentials. Like the original Transfer Access Protocol, a chain delivered to the relying party would require:

- Storing an Identifier for the Original Key
- Final new Public Key
- Metadata for New Public Key
- Final new Attestation Certificate
- Challenge

- Counter
- Signature proving possession of the new Authentication Private Key
- Signature proving possession of the new Attestation Private Key

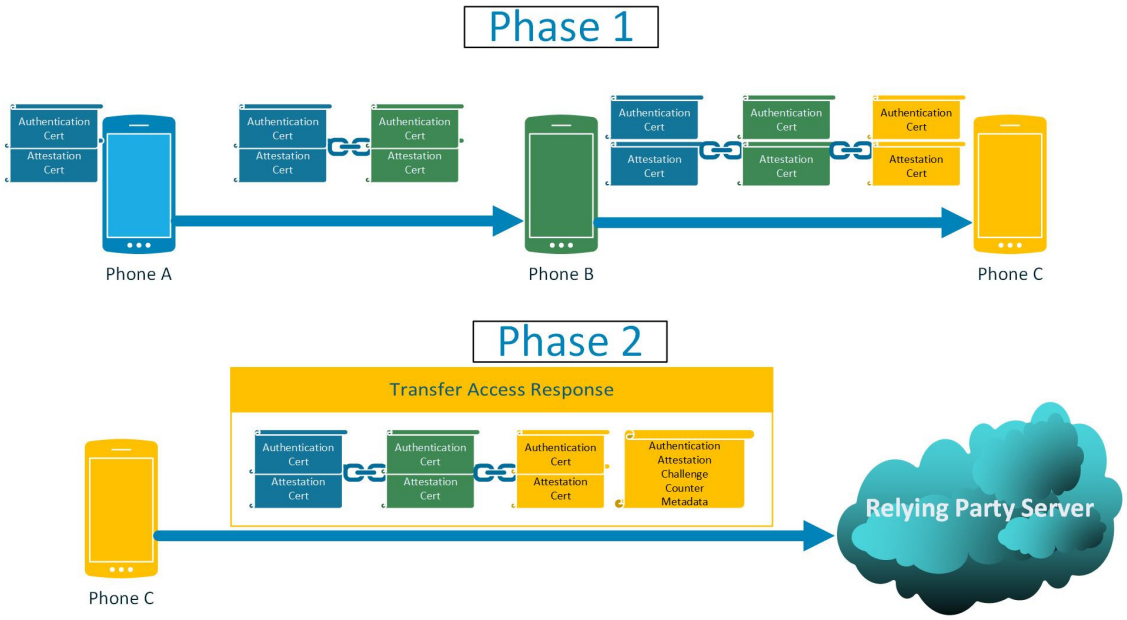


Figure 4.5: When chaining Transfers of Access, Phase 1 may need to include transfers through many devices (in this figure, Phone A transfers to B, which transfers to C before visiting the relying party. In Phase 2, the final device in the chain (Phone C) delivers the entire Transfer Access Chain to the relying party, along with signatures with its Attestation and Authentication Private Keys. It also includes the Challenge, Counter, and Key Metadata for this and future authentications.

Thus, when Phone B tries to transfer access to Phone C, it would simply add its relevant information to the Transfer Access credential given to it by Phone A, creating a chain. We can improve efficiency within this chain by storing and signing over only those items which the server needs. For example, the server does not need metadata, identifiers, or a counter for

Phone B's keys so the chain should neither keep that information nor sign over it. Phone C, when it does eventually deliver the chain to the relying party, can add its counter, metadata for its key, the challenge from the authentication request, and signatures with its Attestation and Authentication Private Keys. With this information the relying party can check to make sure that it trusts the new hardware, the authenticator is not cloned, the response is unique to the authentication request provided, and the device owns the corresponding private key. Fig. 4.5 shows how the chaining works, conceptually.

However, the relying party would also like to check the links in the chain. In the example above (Phone A $\rightarrow$ Phone B $\rightarrow$ Phone C), it needs to check that Phone B has a valid attestation certificate and has the matching attestation private key. It also needs to verify that Phone B has the corresponding authentication private key and agrees to transfer access to Phone C. Therefore, we need to store Phone B's Attestation Certificate and Authentication Public Key in order for the server to check those signatures. When Phone B crafts a transfer to Phone C, it will perform signatures with the corresponding private keys over the included Attestation Certificate and Public Key for Phone C. Such an approach generalizes to a chain of many devices, as the intermediary authenticators can simply use their Authentication Private Keys to sign the next Public Key in the chain and their Attestation Private Keys to sign the next Attestation Certificate in the chain. By chaining the signatures in this manner, the relying party can trust the chain of authentication private key trust, and can trust that none of the devices have been impersonated because each phone signs the next phone's attestation certificate.

4.3.5 Other Challenges

Looking up the Transfer Access Credential in the Authenticator When Phone B navigates to a relying party for the first time after receiving a Transfer Access credential, the server will look up the key metadata it knows for the user account and ask for authentication. Because the server does not know about any of the new credentials created by Phone B, Phone B needs to store, along with the Transfer Access credentials, an identifier for the original

key from Phone A. In the case where the credential has been transferred through a chain of devices, the last phone in the chain needs to be able to look up the Transfer Access credential using metadata for the original key from Phone A, which started the chain.

Parsing the message We mentioned previously that adding a bit in the Message Header to indicate a Transfer Access Response to an authentication request allows the server to easily differentiate between the two possible responses. We further aim to help the server parse the Transfer Access response by providing sequence numbers in the chain of Transfer Access credentials. By appending the existing chain to the back of the new Transfer Access credential (inserting it at the front of the chain), the server can immediately know how much space to allocate for storing the chain.

Simplicity In Section 4.3.2, we mention that the relying party needs signatures with both the attestation and authentication private keys in order to verify the transfer of trust through the chain. However, we note that because the current FIDO implementation requires a signature using the attestation private key during registration of a new key pair, the relying party already trusts Phone A's attestation. Therefore, signing with Phone A's attestation private key during the first transfer of access from Phone A to Phone B does not add any security properties. However, in a longer chain, we need to ensure that if Phone B transfers to Phone C, and C to D, that both Phone B and in turn Phone C are forced to produce signatures using both the attestation and authentication private keys. For simplicity, we have chosen in our implementation to require the extra signature from Phone A using the already-trusted attestation private key so that the messages throughout the chain are formed using the same algorithm.

Ordering We note that the chain of credentials that passes trust from old phones to new ones does not necessarily have to be in order. However, in situations where lots of messages are chained in the wrong order, the complexity of figuring out the correct order lies with the

relying party. Instead of forcing the relying party to try all combinations when the chain arrives out of order, we suggest that it simply discard the chain and fail to transfer the credentials.

Deleting Keys We claim that after the completion of the Transfer Access Protocol the old Phone A should delete all transferred keys, but this raises some interesting trade-offs. Deleting the keys as soon as possible helps protect a user who forgets to factory reset a phone before selling it to a potential attacker (requiring a second factor can help mitigate attacks in this case as well). But in the case of a failure (say one phone runs out of battery during the transfer or the wireless environment becomes disturbed), transfers cannot be rerun. As such, to account for failures during transfer, we suggest waiting until the completion of Stage 1 (where Phone A receives acknowledgement from Phone B for all successfully transferred credential) to delete keys.

As a consequence of deleting keys upon the completion of Stage 1, however, we note that there are potentially times where a user can lose authenticator access. For example, if the user transfers access from Phone A to Phone B, but then loses Phone B before logging in to the relying party (before being able to transfer to Phone C), the server will only know about Phone A, but those credentials will have been deleted. Recovery from this situation is a very interesting problem for which we plan to propose solutions in future work.

Furthermore, if an attacker can prevent the delivery of the final ACK (acknowledgement from Phone B), for example by DOSing the protocol at that phase, Phone A will keep the keys. We propose mitigating the harms of this by alerting the user that the protocol has been interrupted, and allowing the user to then delete the keys manually or rerun the protocol.

In the Threat Model in Section 4.2.2, we discuss some extra attackers beyond the standard Web, Related-Site, Network, and Malware Attackers. Consider, for example, an attacker who obtains temporary access to an unlocked Phone A. This attacker could potentially perform a transfer of access from Phone A to an attacker controlled Phone B. If we did not delete the old keys from Phone A upon the completion of the protocol, the victim may not notice that

credentials have been transferred. The attacker can then phish for the second factor and once obtained, can execute a transfer of access to gain access to an account. If the server does not delete access, the original owner may not ever be aware that an attacker has gained access. We suggest mitigating this by deleting keys on the authenticator and at the relying party so the next log-in will fail and the user will be aware of the problem. Further, we suggest designing the authenticator application in a way that allows users to see and manage stored keys. An authenticator app that requires local authentication to make changes would also help mitigate threats from this type of attack.

Log-In Cross Site Request Forgery We note another interesting attack where Phone A is the attacker's phone. Similar to the attack mentioned above, where an attacker gains temporary access, we can have a situation where an attacker gains temporary access to the user's Phone B and attempts to transfer credentials to it. The next time the victim goes to a site, it is possible they won't realize they are logging in as the attacker, allowing an attacker to collect sensitive data and track activity. As above, requiring some kind of local authentication before using the authenticator application and allowing users to easily manage stored keys can help mitigate this threat.

System Level Malware Though system level malware on either the old phone or new phone is a serious problem for a user even in the case where the user has a FIDO authenticator, we would like to minimize the effects of a compromise on future log-ins on other uncompromised devices. Assuming that the FIDO application is not compromised (a compromise of the FIDO application is out of scope for this work as it could break every aspect of the existing scheme and the proposed Transfer Access Protocol), the authenticator application cannot necessarily trust the OS to create a secure channel between phones. As a result, we suggest putting the crypto library, keys, and potentially some functionality of the authenticator application into a secure element or trusted execution environment.

4.4 Summary

In summary, we propose a two-stage Transfer Access Protocol.

4.4.1 Stage 1

Phone A and Phone B communicate over a shared secure channel. The specifics of such a channel are out of scope for this paper, but we assume that it does not add an attack surface for any in-scope attackers. Ideally, this phase would not impose extra work for the user, for example, it could be done during the initial phone setup when transferring apps and data from the old device. During this phase:

1. Phone A tells Phone B which credentials it would like to transfer. In practice this would be indicated by a unique identifier for each key that Phone B can understand. It should also attach a version number to ensure compatibility. In our implementations, we only accept one valid version number for simplicity.

2. Phone B sends its Attestation Certificate to Phone A. Phone B also generates new credentials for all the valid transferred key identifiers and sends the corresponding public keys back to Phone A. Phone B needs to mark each new public key with the original identifier so that Phone A knows which of its keys to use for signing.

3. Phone A generates a Transfer Access credential, and sends that back to Phone B. That credential may contain a chain, so Phone A must also send the original key metadata (the only one the server knows about) so that Phone B can look up the Transfer Access credential upon the next log-in. The Transfer Access Credential is a function of:

   (a) New Public Key

   (b) The Relying Party Site

   (c) New Attestation Certificate

(d) Old authentication private key

(e) Old attestation private key

4. Phone B acknowledges receipt of the Transfer Access credentials for each transferred key identifier so that Phone A can delete the corresponding credentials.

4.4.2 Stage 2

Phone B navigates to a relying party as normal over TLS. During this phase:

5. The relying party asks for the user account, which the user supplies. This can be done in the browser (for example by typing in a username, etc. and then having the user or authenticator select a key) or it could be done in the authenticator app, which would present credentials by account. The user could, for example, select "log-in with authenticator" and simply select from the accounts with matching domains within the authenticator. We leave the implementation of the authenticator app out of the scope of this paper.

6. Once the relying party knows which key it would like to ask for an authentication, it sends a standard authentication request containing:

   - Challenge
   - Metadata for the selected key

7. Instead of responding with a standard authentication response, Phone B responds with its stored Transfer Access credential chain.

8. The server parses all credentials in the chain and can decide whether to allow or deny access. If it decides to deny access it is up to the relying party whether it wishes to delete old keys or keep them. If it succeeds, it deletes access for Phone A, authorizes

the authentication attempt, and adds the credentials for Phone B so that the user may log-in with a normal FIDO authentication in the future.

These changes would require subtle changes on both the relying party servers and authenticators to process the messages associated with the Transfer Access Protocol.

4.4.3 Summary of Proposed Changes

Here we summarize the concrete changes we propose in the Transfer Access Protocol.

- **Relying Party Servers**

  The server should be updated to handle both authentication and Transfer Access Responses to authentication requests during log-in. We suggest the following changes:

  - Activate a bit in the Message Header to differentiate between authentication and Transfer Access responses.
  - When processing the Transfer Access response, verify the chain of authentication key trust as well as hardware trust.

- **Authenticator Clients**

  - Allow authenticator to create, store, and send Transfer Access credential chains in addition to traditional FIDO authentication credentials and authentication responses.
  - Store metadata for the original key so that the authenticator can look up Transfer Access credential chains when prompted by the key identifier known to the relying party.
  - Expand the API to allow authenticator applications to talk directly to each other and perform the steps from Stage 1(Section 4.4.1) in the Transfer Access Protocol.

4.4.4 Implementation

We have implemented the concepts described in this paper on top of the public FIDO-U2F protocol from Google (`https://github.com/google/u2f-ref-code`). Our changes are available for download from our fork (`https://github.com/alextaka/u2f-ref-code`). The server was implemented in Java; the client was implemented in software in JavaScript.

4.5 FIDO Plenary, Vancouver

I presented the above solution to the FIDO technical working group during the 2017 Plenary in Vancouver, BC. During the discussion that followed, it became clear that, though industry players recognized the importance of such a solution, there were additional problems that needed solving. In particular, many attendees mentioned the hope that an ideal solution would solve not only the problem of Device Upgrade, but that would also help users recover in cases of Device Loss. Inspired by the feedback from that session, we present two additional proposals in Chapters 5 and 6 with the aim of allowing users to recover from lost devices in WebAuthn.

Chapter 5

RECOVERING FROM DEVICE LOSS: PREEMPTIVELY SYNC KEYS

The solution presented in this chapter builds on the Transfer Access Protocol from Chapter 4 to allow users to recover from device loss without having to run a recovery process at each account [103]. In particular, it requires a single local initial setup to preemptively sync keys after which users can authenticate normally with replacement authenticators for all accounts even when the original authenticator is no longer available. Imagine the following scenario: a user is using Phone A as an authenticator but loses that device and must replace it with a new one, Phone B. Phone B should be easy to set up as an authenticator without losing access to any of the accounts stored on the previous (now lost) Phone A. To enable such a simple recovery, this chapter proposes the Preemptively Synced Keys (PSK) Protocol.

Similar to FIDO's recommendation to register multiple authenticators, the PSK protocol described in this chapter uses a second device called a "backup device". However, unlike the FIDO recommendations, users need not manually register the backup device at each account [42]. Instead they have the following additional requirements above the existing basic WebAuthn flow:

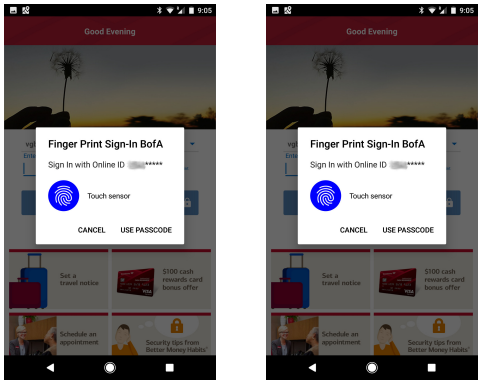
1. The user will need to acquire and store a backup device and retain access to it.

2. The user will need to be able to authenticate locally to the backup device in some way via an authorization gesture.

3. The user will need to retrieve the backup device and use it in a setup protocol each time he sets up a new authenticator.

Thus, in the PSK protocol, when a user gets a replacement authenticator and initializes it as a recoverable WebAuthn authenticator, the device will ask the user to sync with a backup. The user will present the backup device to the replacement authenticator (over a secure connection via network, Bluetooth, NFC, WiFi, etc), and provide a local authorization gesture (biometric, PIN, etc) on the backup device. The user can then store the backup device in any way he sees fit and continue to use the replacement authenticator as a normal WebAuthn authenticator.

This means the registration and authentication experience should be the same on the replacement as it was on the old, now lost, authenticator. For example, when the user navigates to a site where he already has an account, he expects a log-in screen as in 5.1. This experience is simple and recognizable enough that it should not change, keeping the cryptographic communication transparent to the user as it does during normal WebAuthn authentications. The first time they authenticate on the replacement device, the experience should be identical, not requiring an alternative recovery process to restore the log-in screen they expect.



Before (Old Device) After (New Device)

Figure 5.1: An example user experience for an app log-in. This screenshot is from the Bank of America app on a Google Pixel. This figure also appears in Chapter 4 as Figure 4.3

.

The underlying mechanism for the PSK protocol builds upon the Transfer Access Protocol from Chapter 4, which uses credential binding to tie the trust in new devices to the trust in the credentials on old devices. However, whereas in Chapter 4 the credential chain starts with the old device (Phone A) and ends with the new device (Phone B), in PSK the credential chain starts with the backup device and ends with the replacement device (Phone B). This allows users to recover from device loss and remove access from lost authenticators with a single, local recovery procedure rather than forcing them to execute a unique recovery for each account. The subsequent sections describe how a user leverages such a backup device to pre-generate recovery key pairs, store them, and pre-load public keys onto a primary authenticator to enable recovery from device loss without compromising the security and privacy benefits of WebAuthn. This work was done in collaboration with Alexei Czeskis and Arnar Birgisson (Google), Hideo Nishimura (NTT Labs), and Tadayoshi Kohno (UW).

5.1 Introduction

The current WebAuthn ecosystem provides secure standards that promise to improve online account security and simplify the experience for internet connected users. This ecosystem allows users to sign into web services through authenticators (for example, a smartphone or dedicated token) that perform user authentication using an asymmetric cryptographic signature that is resistant to phishing attacks and provides two-factor authentication. Similar to the iPhone's TouchID, users on many platforms will have devices, such as phones, that can serve as WebAuthn authenticators. For example, imagine that a user is using a phone as an authenticator. This phone has an app that allows the user to view and manage keys. It also allows the user to log-in to websites using WebAuthn. When the user goes to example.com and selects "log-in with authenticator", the phone alerts the user to scan a fingerprint. The user complies and the server and authenticator app negotiate in a cryptographic protocol to ensure that the user is safely authenticated and consents to the log-in.

There are, however, some unsolved problems in this ecosystem. Chapter 4 provides a solution to situations where the user retains access to an old device and uses it to set up a

new device. In the current WebAuthn ecosystem, when a user gets a new authenticator he must log into all web services independently and register that device. But using the Transfer Access Protocol from Chapter 4, he can set up the new authenticator without having to log in to any sites or perform any additional actions. However, in cases where the user loses his old device and does not have a second registered authenticator there may be no way for him to authenticate and/or register a replacement device. To solve this problem, this chapter proposes the Preemptively Synced Keys (PSK) Protocol.

The PSK Protocol utilizes a "backup device" to recover from lost primary authenticators. From the user's perspective, this requires a **single additional action per device**. During the set up of each authenticator the user must sync it with a backup device. Then in subsequent registrations, the primary authenticator registers as normal, but also automatically registers a recovery public key given to it by the backup device during the initial syncing process. If the user loses the primary authenticator, he can restore to a new replacement authenticator by syncing the replacement with the same backup device. From that point forward he can use it seamlessly with all existing accounts. But such a solution is not without trade-offs. The following lists the benefits and caveats of using Preemptively Synced Keys for recovering from device loss:

Benefits

- Scalable recovery from device loss
- Preserves security and privacy benefits of WebAuthn
- Retains the Usability benefits of WebAuthn
- Retains almost all of the Deployability benefits of WebAuthn

Caveats

- Requires users to acquire and manage a "backup device"

- Requires an initial set up to sync the backup device with each primary authenticator

- Necessitates a storage and computation overhead on backup devices and primary authenticators.

- Relying parties must implement some kind of credential chaining as in the Transfer Access Protocol in Chapter 4 and allow for registering a recovery public key during a standard WebAuthn registration.

5.2 Goals

The goal of this work is to restore account access on a replacement authenticator without the user having access to the old primary authenticator. Ideally, this can be done while preserving the desirable properties of the existing WebAuthn scheme. The choices made in the design of this protocol serve to primarily protect the existing security and privacy properties of the WebAuthn protocol while minimizing changes to the user experience and deployability. This section analyzes the goals using the Usability, Deployability, Security framework from Section 3.2 and describes some of the challenges that arise.

5.2.1 Usability

As in Section 3.2, solutions enabling recovery from device loss should minimize additional usability burden on users by preserving or improving upon as many of the existing properties as possible. Further, changes to the protocol should require either **no additional action** from the user or, in the worst case, a **single addition action**. Although this solution requires users to manage and remember where they have stored the backup device, it should still aim to preserve the rest of the advantageous properties of WebAuthn:

- **Memorywise-Effortless**: PSK should preserve the user's choice of second factors and retain the possibility of a diverse ecosystem of devices. Should users choose a biomet-

ric second factor, transfer and recovery would ideally not require them to remember anything.

- **Scalable**: PSK should scale well to many different accounts on many relying parties.
- **Nothing to Carry**: PSK should not require users to carry more devices than they already have to carry in the WebAuthn ecosystem. In other words, users should not have to carry the backup device for registrations or authentications.
- **Physically Effortless**: To the extent possible, users should retain the ability to use their selected level of physical effort.
- **Easy to Learn and Use**: PSK should not be more difficult to learn or use than a standard WebAuthn authentication or registration.
- **Infrequent Errors**: PSK should not cause more errors than a standard WebAuthn authentication or registration and should not affect future authentications or registrations.

5.2.2 Deployability

PSK should not excessively hinder the *deployability* of WebAuthn. It should remain:

- **Accessible**: Users should still be able to use devices with many form factors and second factors, instead of being tied to a particular type of device.
- **Negligible Cost per User**: Users will be required to acquire a backup device, which should be low cost (not a high-powered high-capability device).
- **Server and Browser Compatible**: PSK would ideally be compatible with existing devices and code. Any specification changes should require minimal additional code and hardware from end users and relying parties.

- **Mature**: PSK should prefer mature technologies instead of introducing unvetted or untested systems/mathematics.

- **Non-Proprietary**: Anyone should be able to implement any aspect of the recovery procedure.

5.2.3 Security

As noted in Section 2.5.5, the WebAuthn protocol is more secure than passwords in all the criteria posed by Bonneau et al. and also provides additional privacy protections that the password ecosystem doesn't provide [21]. The PSK solution should preserve these properties in all steps of the solution's protocol without weakening future or past authentications or registrations.

- **Resilient to Theft/Require consent**: PSK should still allow users to protect protocols with a local authorization gesture (PIN, Biometric, etc).

- **Not Reliant on Trusted Third Parties**: PSK should only have to trust the prover(s) and verifier for each account.

- **Unlinkable**: PSK should not render future or past credentials linkable by any party, even colluding parties.

- **Cannot be Copied by External Observation**: PSK should not weaken the protections against copying by external observation for any attacker, nor should it present new opportunities for external observers.

- **Cannot be Copied by Internal Observation**: PSK should be implementable within secure elements.

- **Phishing Resistant**: No part of the PSK protocol should be phishable, and future authentications or registrations should remain un-phishable as well.

- **Resilient to Throttled/Unthrottled Guessing**: PSK should not rely on guessable parameters that travel between devices.

- **Unaffected by Leaks from Other Verifiers**: PSK should not allow leaks from any entity to affect the recovery or upgrade of any other entity.

- **Resilient to Targeted Impersonation**: PSK should not rely on the user's personal information outside of using that information for a local authorization gesture, as in current WebAuthn registrations and authentications.

Threat Model : To determine whether the additional steps in the Recovery Protocol uphold these goals, we analyze each step using a threat model based on previous works done in this space. For example, we will use the attackers mentioned in Lang et al.:

- Web Attackers who can phish for credentials by setting up forged web pages, including correct TLS certificates for victim sites.

- Related-Site attackers where users may have reused the same credentials as the victim site.

- Network Attackers who can MITM connections with correct certificates or decrypt traffic.

- Malware Attackers who can install malicious applications or take over benign applications.

These attackers lead to an additional set of security properties that PSK should uphold:

- Defend Against MITM: Attackers who Man-In-The-Middle the connection, for example between the browser and relying party server, should not gain an advantage by attacking during any step in the Recovery Protocol.

- Session-Duplication: The protocol should not aid in the ability for stealing credentials to result in session-duplication, for example, by exposing long-term cookies or passwords.

- Prevent Session Riding: The protocol should not allow an adversary to gain access to an existing session or to a future existing session.

- Trusted Hardware: The protocol should allow the relying party to verify that it trusts the new hardware before allowing access.

- Detecting Clones: The protocol should allow for the continued detection of potential authenticator clones by keeping a counter.

We will also consider other potential attackers through whom we can demonstrate some of the strong security and privacy properties of the existing WebAuthn authentication scheme. For example, Site Attackers who can dump logs and credentials from the victim site may be able to reveal user passwords, and adversaries who are able to gain physical control of devices at later or earlier times (OEM vs. repurchasing an old phone) can raise some interesting concerns.

Out of Scope : However, certain attacks are out-of-scope for this work. For example, we do not consider protecting against a malicious authenticator as existing authentications would not be secure regardless of the choice of recovery protocol. Attackers who can compromise the local wireless environment and attack the network are also out of scope, as are attackers who can compromise the operating system or browser. Because the PSK Protocol grants access to authenticator functionality, rather than performing an independent security and privacy analysis of each piece of this protocol, this chapter aims to present a protocol that introduces no *additional* vulnerabilities.

We believe that the addition of this work to the WebAuthn specifications would help secure potential vulnerabilities that result when users try to recover access to accounts after losing devices.

5.2.4 Goal Conditions

Before diving into potential workable solutions for recovering from a lost authenticator, we discuss the assumptions and the properties constituting goal conditions for the Recovery Protocol. To start, we have the following assumptions:

5.2.4.1 Initial Assumptions

- The user has a backup device (details to be described later) and replacement Phone B.
- The user no longer has access to the original primary authenticator (Phone A), which may or may not have keys and associated metadata, each associated with an account.
- Phone B does not have any existing keys.
- The backup device and Phone B can create a "secure channel".
  - This secure channel is out of scope for the Recovery Protocol. We assume that this channel can only be set up by a legitimate user who explicitly allows the restoration of Phone B from the backup device. For the purposes of this paper, we assume this channel allows communication between the two phones that is resilient to all possible attacks, including eavesdropping and Man-In-The-Middle attacks.
  - Requiring a user to explicitly authenticate with each device (backup device and Phone B) may be in scope.

5.2.4.2 Target Goal Conditions (for each account on Phone A)

At the conclusion of this protocol, we expect the following properties to hold for each account:

- Phone B has a "restored key".
- Phone B is logged in to the relying party.
- The relying party removes Phone A's access.
- The relying party adds access for Phone B so that it will be able to authenticate in the future using standard WebAuthn with the restored key.
- **Security Goals**: Throughout each step of the procedure, we expect the Recovery protocol to give attackers no advantage in attacking WebAuthn.

5.2.4.3 Final Recovered State

At the conclusion of the setup of the replacement device (Phone B), Phone B should have credentials that can authenticate to *all* of the accounts originally set up on Phone A. As was the case with Phone A before it was lost, Phone B should be able to register with new accounts with no extra user actions. Should Phone B be lost, the backup device should be able to restore all Phone B's accounts to its replacement (Phone C).

5.2.5 Valid Backup Devices

Any device that can perform the above required functions can serve as a backup device. The requirements are that it be able to:

- Generate key pairs
- Store key pairs

- Establish a secure channel to authenticators
- Locally authenticate a user
- Provide an attestation certificate
- Sign with attestation private key
- Sign with private keys

Note that though we have presented this in subsequent images as a USB key (see Figure 5.2), it could also be a dedicated device, a phone, or even a cloud service. We could also use a key-wrapping system to offload storage to the cloud while still maintaining a physical security key, as mentioned in Section 5.5. We think that affording users a choice of backup authentication hardware and user experience is best for the WebAuthn ecosystem going forward.

5.3 Solutions

This section starts with the status quo and builds progressively better solutions by solving existing usability problems. Starting with the simple solution presented in Section 5.3.2, the remaining subsections refine this solution to improve usability and present details about how the protocol solves remaining problems.

5.3.1 No Solution

In the current ecosystem, the FIDO recommended way to recover from lost devices is to register multiple authenticators [42]. In this solution, a user has two authenticators, their primary authenticator Phone A and their secondary authenticator Phone A'. For every site where the user registers Phone A, he also registers Phone A'. The user can then use Phone A for authentication and store Phone A' for future registrations. As mentioned in Section 5.1,

such a restoration method negatively affects usability by forcing users to carry multiple devices for registrations.

When the user loses Phone A and replaces it with Phone B, he must retrieve Phone A' to recover access to lost accounts. Using phone A', he must authenticate with each existing account to register Phone B. Because the user must do so with all existing accounts, this solution scales very poorly, potentially causing frequent errors if the user doesn't have a list of all the sites for which he needs to register the replacement Phone B. Better solutions would improve usability so that they require either **no additional actions** or a **single additional action per device** for all accounts, rather than an additional action for *each* account.

5.3.2 Simple Transfer Solution

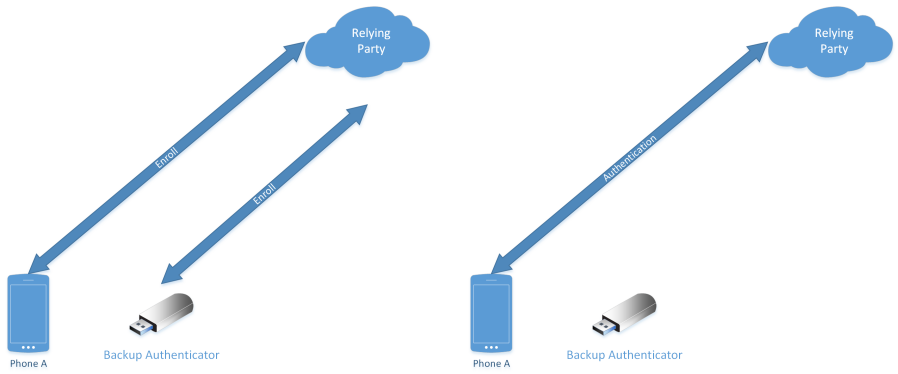


Figure 5.2: Simple Transfer Solution: Each registration requires registering both Phone A **and** the backup device. However, authentication only requires Phone A.

A simple improved solution based on the Transfer Access Protocol [103] can automate some of the recovery process to reduce the user burden during recovery to a **single additional action per device** instead of a **single additional action per account per device**. It could work as follows: As in the previous example, the user has Phone A as a primary authenticator, but this time instead of a second authenticator, he has a backup

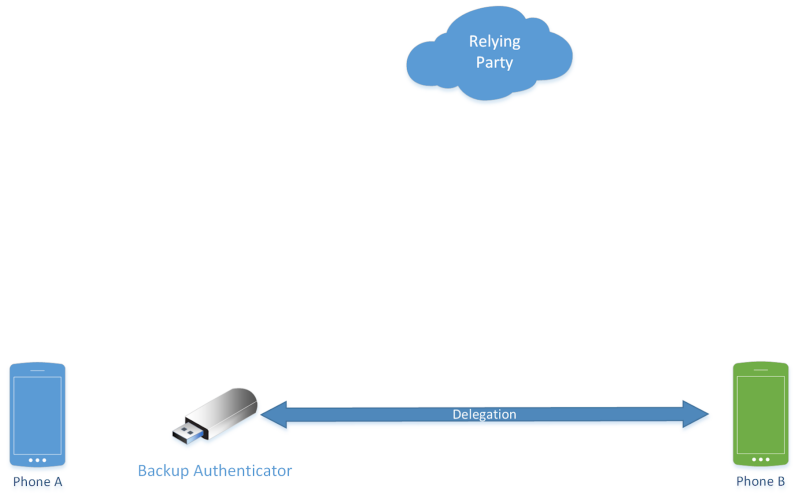


Figure 5.3: Phone B can receive delegations from the backup device without Phone A

device. Like the previous example, for every site where he registers Phone A, the user also registers a second device — in this case the backup device. The user can use Phone A to authenticate as normal and can store the backup device until the next registration.

However, unlike the previous *No Proposal*, when the user loses Phone A and replaces it with Phone B, he can retrieve the backup device and sync it with Phone B. During this syncing process, the backup device will run a delegation protocol similar to the Transfer Access Protocol [103] to delegate access from its existing credentials to credentials generated by Phone B. This has the benefit of binding the trust to a chain of credentials from the backup key to Phone B and can include attestations that relying parties can use to decide whether they trust each device in the chain.

The next time the user tries to authenticate with Phone B, instead of delivering a standard authentication response, it will include the certificate chain containing the delegation from the backup device to Phone B. As in the Transfer Access Protocol [103] from Chapter 4, when the relying party receives this delegation, it should:

1. Check to make sure the delegation is valid, and if so:

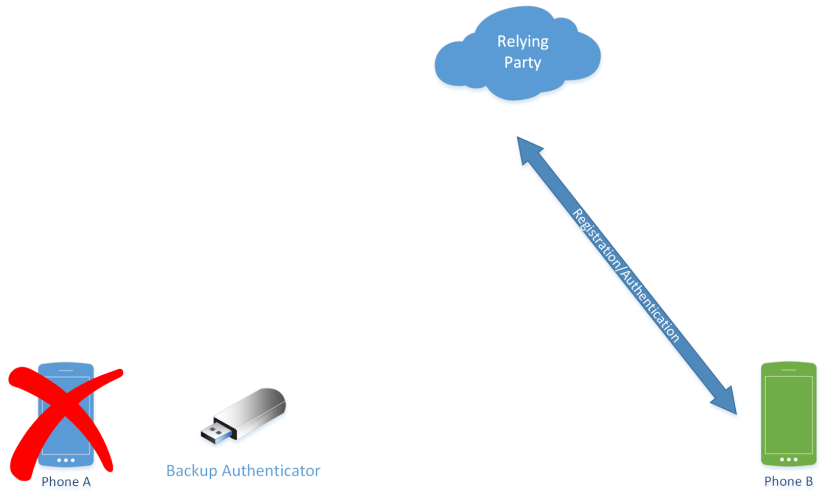


Figure 5.4: On the next login, Phone B registers with the Relying Party and authenticates the session. The Relying Party removes access for Phone A. Phone B can log-in without the backup device.

2. Register Phone B's key

3. Remove Phone A's key

Now the user can use Phone B to authenticate as usual with no change in the standard WebAuthn flow. Also note that the backup key remains registered with the relying party. So should the user lose access to Phone B before delivering the delegation to the server, Phone C should be recoverable by the exact same process.

With this *Simple Transfer Solution*, the user can now execute a single action to restore Phone A's access to all accounts on Phone B, significantly improving the scalability of the recovery procedure. This also has the possibility to reduce errors, make the protocol easier to use, and reduce the physical effort required of the user, drastically improving usability across the board.

However, when the user wants to register an authenticator with a new relying party or create a new account, he still needs to fetch the backup device and enroll two devices instead

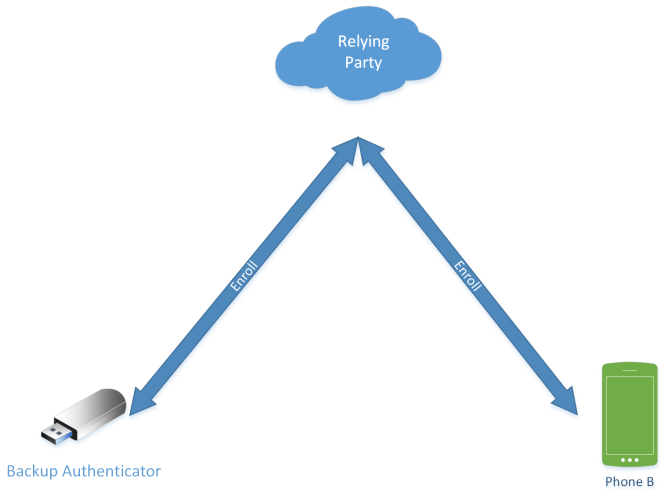


Figure 5.5: Future registrations require enrolling both Phone B and the backup device.

of one. As such, the user must still carry an extra device for registrations and must execute an extra action for every registration.

5.3.3 Preemptively Syncing Keys: Only Require the Backup Key on First Use

We can improve the Simple Transfer Solution from Section 5.3.2 so that it does not require the user to manually register the backup key during every new registration. Instead, we propose a scheme that requires the user to retrieve the backup key only during the setup of a new device. The subsequent sections describe exactly how setup, registration, authentication, and recovery happen at each step for the original authenticator, Phone A and the replacement device, Phone B.

5.3.3.1 Initial Setup of the Original Authenticator (Phone A)

When the user first sets up a recoverable authenticator (Phone A), it will ask the user to present a backup device. We assume that Phone A and the backup device can set up a secure channel and that only a valid user can initiate such a channel. To retain user options, users

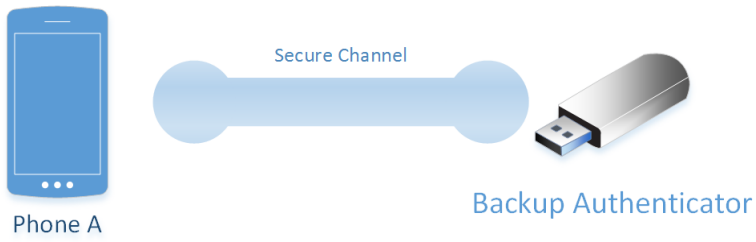


Figure 5.6: During setup of Phone A, the user will provide a backup device. Only a valid user can set up a secure channel between Phone A and the Backup.

can potentially use cloud options or physical backup devices (see Section 5.2.5).

During this initial setup, the backup device will generate backup key pairs and unique credential descriptors for each key and will send each $< public\_key, credential\_descriptor >$ pair to Phone A, along with its attestation object. Note that because the backup device does not know for which RP the user will associate these credentials, it will have to leave that blank. Phone A will store the backup keys and mark them as unused. For the purposes of this document, let us assume that the backup key generates more keys than the user will ever need (more than the total number of accounts the user will create in a lifetime).

At the conclusion of this step, Phone A has the following stored:

- A single block of recovery keys and associated metadata given to it by the backup device

The Backup device has the following stored:

- The same block of recovery keys generated for Phone A

5.3.3.2 Registration/Enrollment of Phone A

When users enroll, they will only need Phone A. The user interface will remain unchanged from standard WebAuthn. As a reminder, during registration, the relying party asks Phone

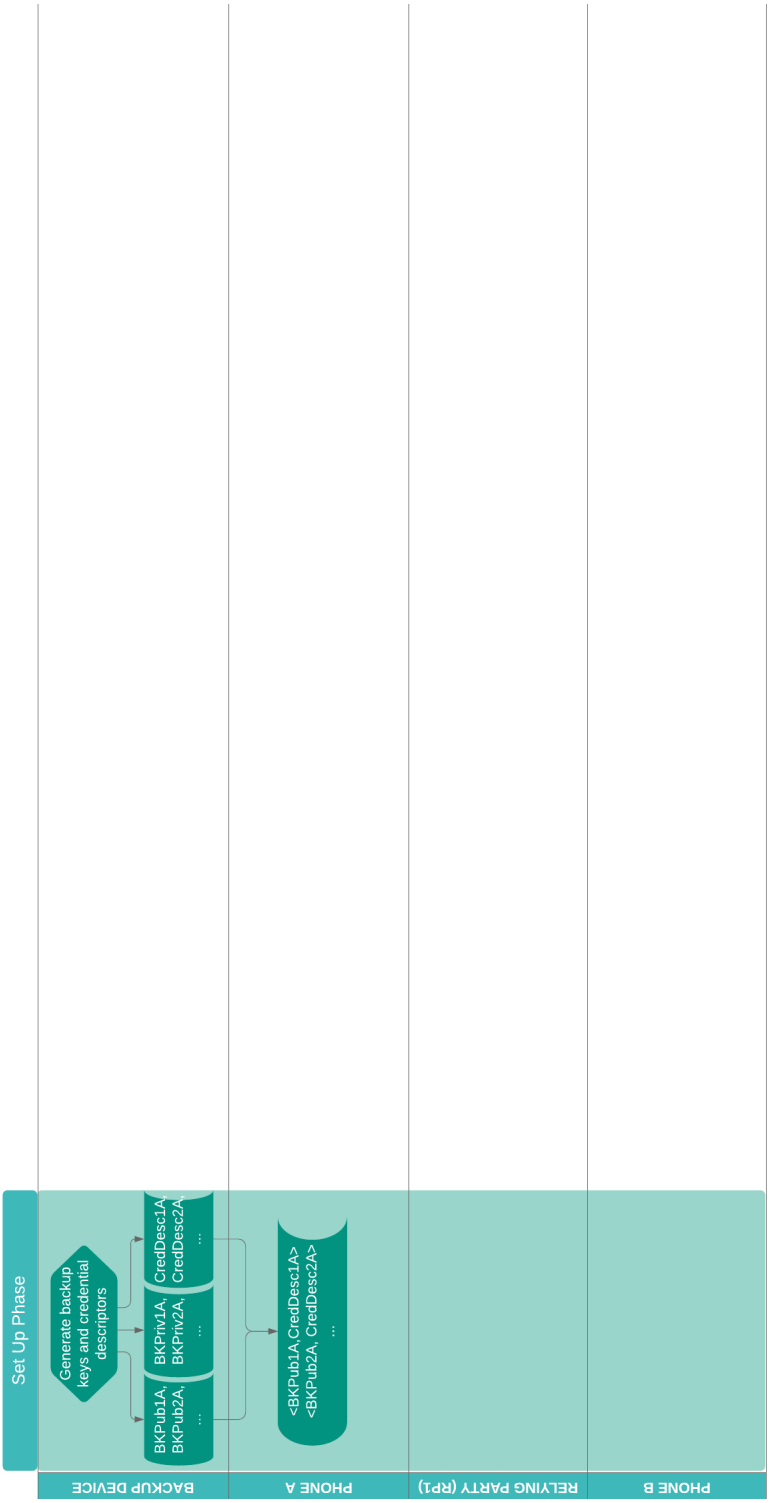


Figure 5.7: Setup phase of the PSK protocol

A to create a new asymmetric key pair and associate that pair with the relying party. That request contains a challenge parameter to be used in Phone A's response. Phone A creates a response and sends it to the server so that the relying party can store the necessary information for future authentications. Key components of that response are:

- User Public Key — This is the new public key to be registered.
- Metadata, including the credential descriptor — Allows the client and server to efficiently look up keys and binds each key to a specific account.
- Attestation Object — Allows the server to decide whether it trusts the hardware. Attestations are batched by device, each device containing a certificate, public key, and matching private key.
- Challenge — Contains a nonce to make each registration unique so that it can not be reused. For example, this prevents an attacker from re-registering a previously registered and removed key.
- Signature — Proves ownership of the attestation private key so that the server knows the device matches the above Attestation Certificate. This prevents an untrusted device from falsely providing the Attestation Certificate of a trusted device in order to enroll a new private key.

However, for recovery to work, Phone A will also select an unused recovery key and its associated credential descriptor given to it by the backup device. Before crafting the response to the relying party, Phone A will replace the credential id for its own key with the credential id of the selected recovery key. From this point on, Phone A will associate the selected backup credential id with the credential it created for the registration. It will also mark the selected recovery key as "used" so that it cannot be selected to recover another key used in a future registration. As a result, Phone A will add the following components to the registration message:

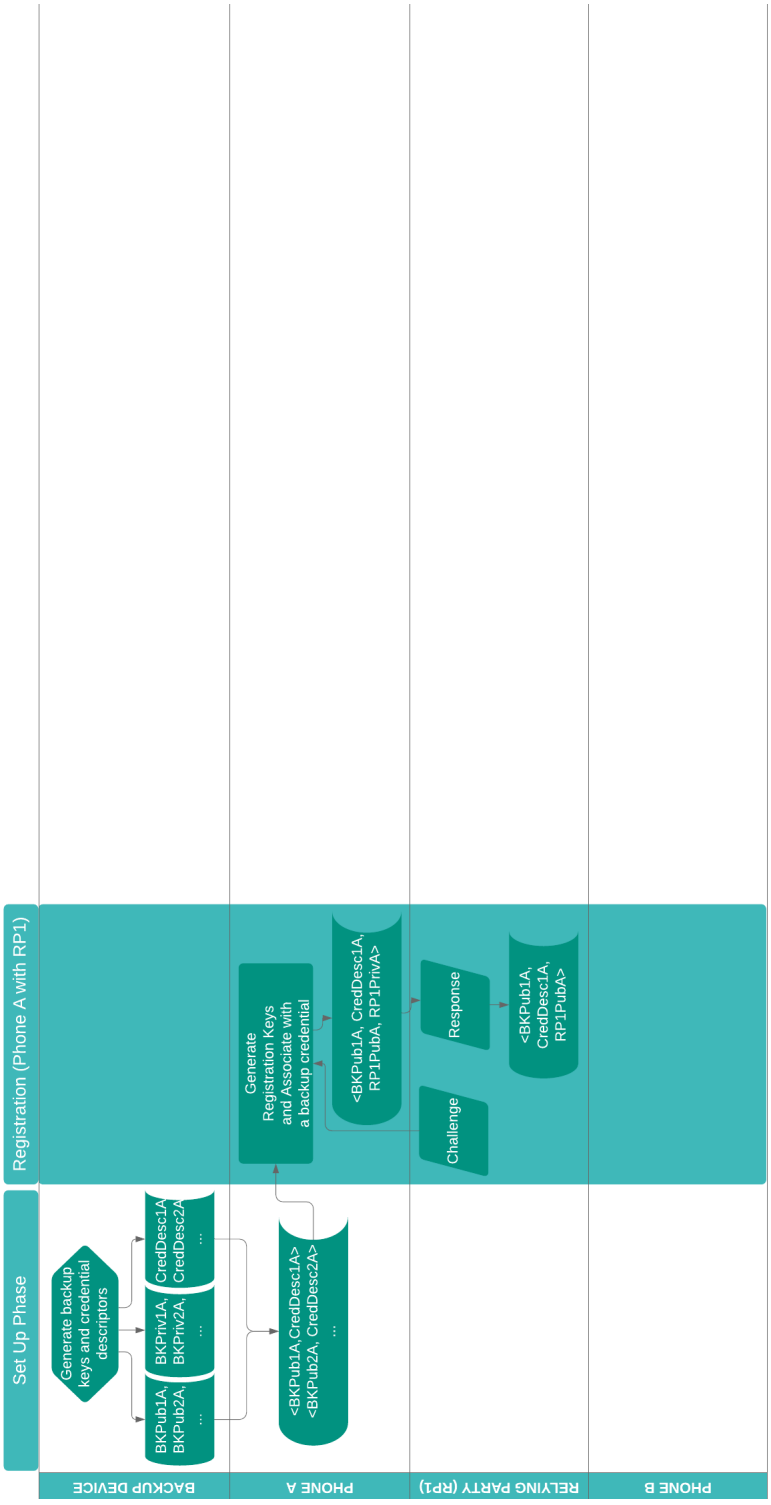


Figure 5.8: Registration of Phone A with RP1 in the PSK protocol

- Backup Public Key (backup device) — This is the backup key for the account.
- Attestation Certificate (backup device) — Attestation for the hardware of the backup device.

Note that the Metadata included in the response will use the credential id generated by the backup device instead of the credential id generated on Phone A.

Upon receipt of the registration response, the Relying Party will check the attestation certificates for both Phone A and the backup device. If both are satisfactory, it will store the backup device's public key as a backup key. It will also store Phone A's public key and associated metadata as it would during a normal WebAuthn registration.

5.3.3.3 Authentication with Phone A

Save for a key point addressed in Section 5.4.1, authentication should be unchanged from the existing WebAuthn protocol. When the relying party would like to authenticate an already-registered authenticator, it crafts a request containing a challenge and the necessary metadata for the authenticator to select keys. The authenticator uses this information to select the appropriate credentials and craft an authentication response that can convince the relying party to authorize the user. Key components of the authentication response sent by the authenticator are:

- User Presence/Authorization Gesture — Requires the user to authorize the authentication, preventing attacks relying on remote surreptitious activation of the authenticator.
- Counter — Allows for clone detection. In the case of a cloned authenticator, the server will see consecutive log-ins that don't increment the counter correctly.
- Metadata — Binds the credential to the relying party, allowing the authenticator to efficiently look up the key and preventing attacks which seek to determine if some other key is present on the authenticator.

- Challenge — Contains a nonce to make each log-in unique, preventing replay and phishing attacks.

- Signature — Proves ownership of the private key, the basis for authentication.

Notice that the authentication procedure does not require the backup device.

5.3.3.4 Recovering from the Loss of an Authenticator

Upon losing Phone A, the user will want to replace it with Phone B and will want the following properties to hold with minimal extra effort:

- Phone B can log in at any Relying Party to which Phone A used to be able to log in.

- Phone B can register keys for new accounts and register backup keys given to it by the backup device.

- Phone A's access should be revoked at each Relying Party

  - Note that this is not provided immediately by the proposed scheme, but will be provided as soon as a replacement device delivers the delegation chain. An online third party that stores all sites and key metadata could do this update immediately (ex: LastPass). However, the online third party may be able to link the user's accounts.

  - A local authorization gesture should prevent log-ins should an attacker get access to the now lost Phone A.

5.3.3.5 Setup New Replacement Device (Phone B)

To enable such a recovery, Phone B requires the user to go through the same setup procedure that occurred with Phone A. However, because Phone B is replacing Phone A, there are different steps occurring behind the scenes, transparent to the user.

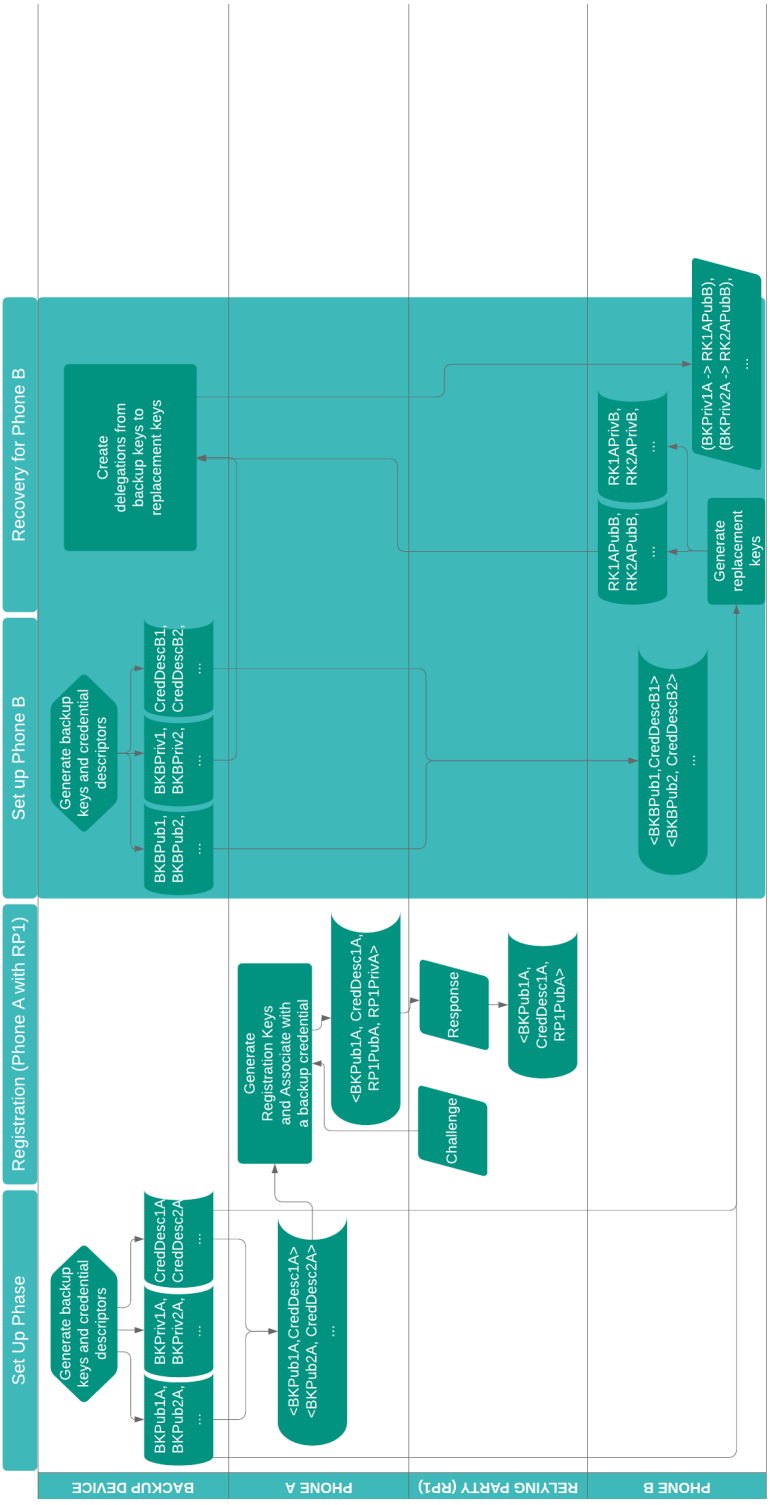


Figure 5.9: Set up and recovery procedure for replacement phone (Phone B).

1. As before, when the user sets up Phone B as a recoverable authenticator, it will ask the user to sync a backup device, but this time the user will tell Phone B it is meant to restore access from a lost device.

2. The user will retrieve the backup device used with Phone A and create a secure channel between Phone B and the backup device.

3. Over the secure channel, the backup device will tell Phone B how many backup keys it has.

4. Phone B will generate that number of key pairs

5. Phone B will send all public keys to the backup device

6. Phone B will send its hardware attestation to the backup device

7. The backup device will perform a "delegation", similar to a transfer access message [103], delegating access from each of its stored backup keys to one of Phone B's newly generated public keys

8. The backup device will send each of these delegations and respective public keys and credential descriptors back to Phone B

9. Phone B will mark these as "pending" and store them for future authentications.

10. The backup device will generate new recovery key pairs and send them, along with its Attestation Object to Phone B for future registrations (as it did with Phone A in the Initial Setup)

At the conclusion of this step, Phone B has synced with a backup device in much the same way that Phone A originally did. But Phone B's state looks a bit different. Instead of storing a single block of keys, it stores three main blocks of credentials:

1. Like Phone A did, it stores a selection of pre-synced backup keys for future registrations given to it by the backup device.

2. Unlike Phone A, Phone B has a block of self-generated keys equal to the number of pre-synced keys the backup device generated for and gave to Phone A. These keys are not associated with any accounts as far as Phone B knows.

3. Phone B has delegations from all the backup keys generated for Phone A to fresh credentials generated on Phone B. These keys are marked "pending" and are tied to the credentials Phone B generated.

The backup device now stores two blocks of keys:

1. The block of recovery keys generated for Phone A

2. The block of recovery keys generated for Phone B

Note that the backup device has no knowledge about which of the keys it generated as backups for Phone A were actually used in registrations. This leads to the overhead discussed in Section 5.4.

5.3.3.6 Registration with the New Replacement Device (Phone B)

Registration will proceed as in Section 5.3.3.2 for Phone A. When the user registers Phone B with an account, the relying party will ask Phone B to create a new asymmetric key pair and associate that pair with the relying party. That request contains a challenge parameter which Phone B can use during the creation of its response. Phone B will generate a new key pair but will also select an unused backup key and associated credential descriptor generated for it by the backup device to associate with its created key. Like registrations for Phone A, before crafting the response to the relying party, Phone B will replace the credential id for its own key with that of the selected recovery key. From this point on, Phone B associates

the selected backup credential id with the credential it created for registration. It will also mark the selected recovery key as "used" so that it cannot be selected to recover another key generated in a future registration.

As before, the registration response will contain the following components (components added by the recovery protocol are **bolded** for emphasis):

- User Public Key — This is the new public key to be registered.
- Metadata, including the credential descriptor — Allows the client and server to efficiently look up keys and binds each key to a specific account.
- Attestation Object — Allows the server to decide whether it trusts the hardware.
- Challenge — Contains a nonce to make each registration unique so that it can not be reused.
- Signature — Proves ownership of the attestation private key so that the server knows the device matches the above Attestation Certificate. This prevents an untrusted device from falsely providing the Attestation Certificate of a trusted device in order to enroll a new private key.
- **Metadata (backup device)** — Metadata specifically for the backup key.
- **Backup Public Key (backup device)** — This is the backup key for the account.
- **Attestation Object (backup device)** — Attestation for the hardware of the backup device.

5.3.3.7 Authentication with the Replacement Device (Phone B)

For credentials originally registered on Phone B, authentication occurs in exactly the same manner as it did on Phone A. However, for accounts originally registered on Phone A, Phone

B must recover access.

For those recovering accounts, Phone B will proceed in a manner similar to a Transfer Access Message [103]. When the user tries to log in with the new device (Phone B), the relying party will include a list available credential descriptors in its authentication request (getAssertion). Phone B will not find any matching credential descriptors for its own keys, but it will find a credential descriptor with a matching credential id in its stored delegations. It can then serve this delegation to the server in order to delegate access from the registered backup key to Phone B. This Recovery Message should have the following components:

- Message Header — We suggest using one of the available bits to inform the server that the message is a Recovery Message.

- Metadata (Phone B's new key) — Allows the client and server to efficiently look up keys and binds each key to a specific account.

- New Public Key — The new public key to be enrolled by Phone B.

- Challenge — This makes each registration unique, preventing replay and phishing attacks.

- New Attestation Object (Phone B) — Allows the relying party to determine whether it trusts the new hardware.

- Counter (Phone B) — Notifies the relying party in the case of a cloned authenticator. Given that this is the first log-in on the new device, we don't think it necessary to continue incrementing the old authenticator. As such, we set this counter to zero, which will alert the relying party if there is a clone in future log-in attempts.

- Signature (Chain of Trust) — Proves a chain of trust exists from the backup device to the new credentials

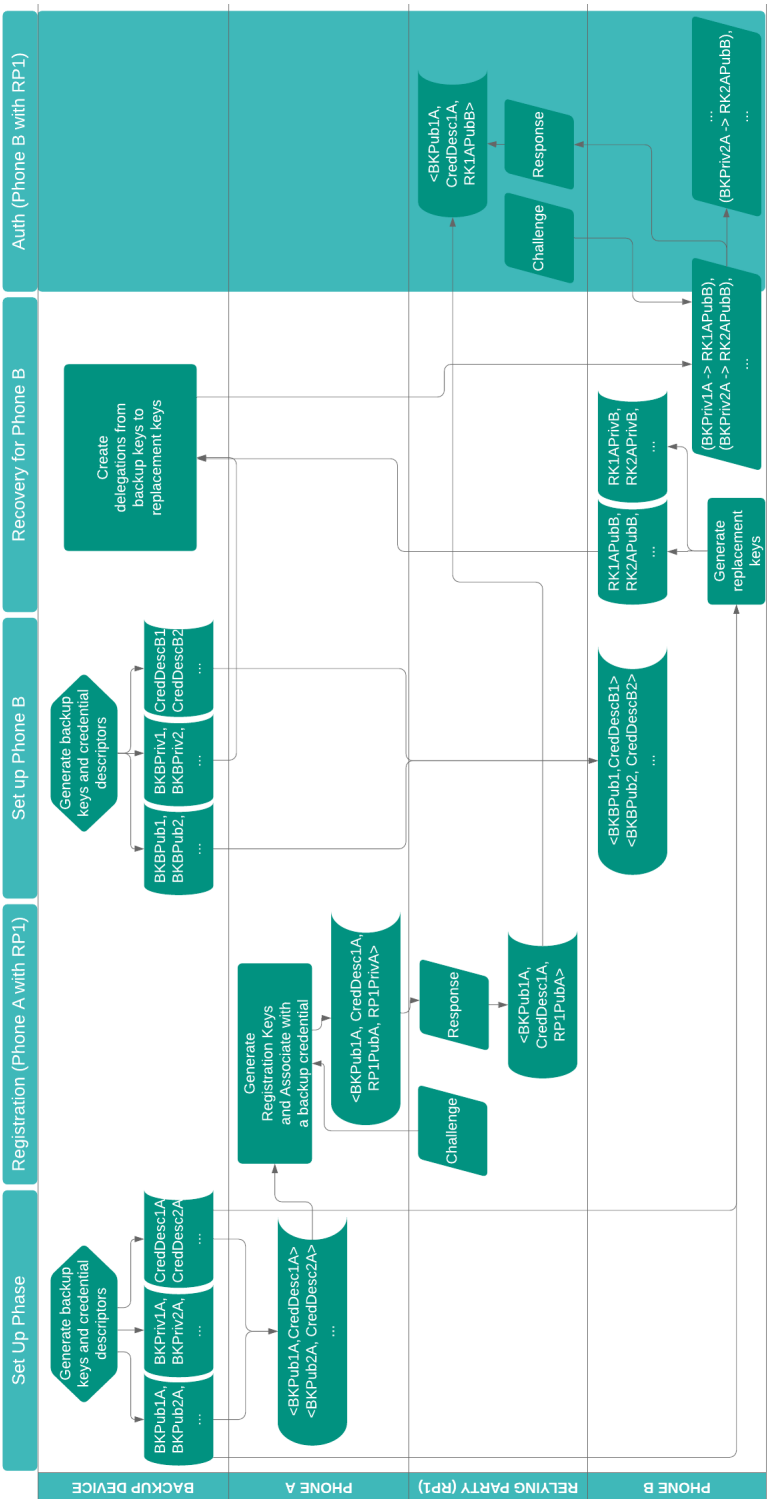


Figure 5.10: Phone B's first authentication with the relying party

- Signature (Authentication) — Proves ownership of an authorized private key so that the user can automatically log-in after completing the Transfer Access Protocol. We expect the user to log-in at the conclusion of this protocol.

- Signature (Attestation) — Proves ownership of the new attestation private key, so that the server knows the credentials have been created by a device with a matching Attestation Certificate. This prevents an untrusted device from falsely providing the Attestation Certificate of a trusted device in order to enroll a new private key.

Notice that this response does not include any information about Phone A or the backup device as they are already registered and trusted by the Relying Party. After receiving this Recovery Message, the Relying Party can remove Phone A's access, register Phone B's keys, and send an acknowledgement to the authenticator. Phone B can receive this acknowledgement and convert its stored certificate into a normal key for future authentications.

If the user loses Phone B before sending the recovery message to the server, the Relying Party will not be aware of a recovery attempt. Thus when the user sets up a new phone, Phone C, the backup device can still perform the recovery protocol as before, providing Phone C with a Recovery Message to give the server. The server will still trust the backup device, so it should be able to process the recovery message for Phone C in the same exact way it would have for Phone B, adding access for the new phone and removing access from the old one (Phone A).

5.3.3.8 Device Upgrade (Transfer Access)

Ideally, a solution allowing recovery from device loss should work seamlessly with device upgrade. In this section, we show how to make the PSK recovery protocol work with the Transfer Access Protocol from Chapter 4. In other words, should a user set up Phone A as a recoverable device, and then upgrade from Phone A to Phone B, that user should not have to re-sync Phone B with the backup device. In the discussion of the Transfer Access Protocol in Chapter 4, we discuss how to build a Transfer Access Message in response to an

authentication request such that we do not sacrifice any of the security or usability properties of WebAuthn [103]. That response contains:

- Message Header — We suggest using one of the available bits to inform the server that the message is a Transfer Access Message.
- Metadata — Allows the client and server to efficiently look up keys and binds each key to a specific account.
- New Public Key — The new public key to be enrolled by Phone B.
- Challenge — This makes each registration unique, preventing replay and phishing attacks.
- New Attestation Certificate — Allows the relying party to determine whether it trusts the new hardware.
- Counter — Notifies the relying party in the case of a cloned authenticator. Given that this is the first log-in on the new device, we don't think it necessary to continue incrementing the old authenticator. As such, we set this counter to zero, which will alert the relying party if there is a clone in future log-in attempts.
- Signature (Authentication) — Proves ownership of an authorized private key so that the user can automatically log-in after completing the Transfer Access Protocol. Recall that this Transfer Access Response gets sent in response to an authentication request, so the user expects to log-in.
- Signature (Attestation) — Proves ownership of the new attestation private key, so that the server knows the credentials have been created by a device with a matching Attestation Certificate. This prevents an untrusted device from falsely providing the Attestation Certificate of a trusted device in order to enroll a new private key.

We also show how to chain these messages in the event that the user performs multiple device upgrades without actually visiting a given relying party to deliver the Transfer Access Message.

We propose that for device recovery to be effective, we can simply add the following to the Transfer Access Message, or the Transfer Access Message Chain, should it be necessary:

- Metadata (backup device) — Metadata specifically for the backup device.
- Backup Public Key (backup device) — This is the backup key for the account.
- Attestation Object (backup device) — Attestation for the hardware of the backup device.

Now when the a replacement device (Phone C) visits the relying party, it can transfer access from Phone B, while keeping the same backup device for future recoveries.

5.3.3.9 Moving to a New Backup Device

As mentioned in Section 3.2, users should be able to select the hardware they want to use. This means that occasionally users will want to migrate to new backup devices. For those users that want to replace a backup device with a new one, there are two options for the user experience.

1. The user can Transfer Access [103] from the old backup device to the new one by creating a secure channel directly between the two. This does not require access to Phone A (the user's active daily authenticator).
2. The user can switch the backup device that backs up Phone A (the user's active primary authenticator) by creating a secure channel between the new backup device and Phone A. The backup device will create backup keys to match all existing keys on Phone A, which will store them and notify the relying party on the authentication that the

backup credentials have changed. The keys on Phone A will update their associated backup keys upon receiving an acknowledgement from the server. This leaves the possibility of some sites not receiving the delegations from Phone A before the user either replaces that device or loses it.

Allowing for both options may be possible, but would require more added messages.

5.4 Caveats

This section lists problems with the Preemptively Synced Keys Protocol.

5.4.1 Username-less flows

Sections 5.3.3.3 and 5.3.3.7 mention that the relying party must put a valid credential descriptor with a matching credential id in the *AllowCredentialDescriptorList*. For normal WebAuthn authentications, this parameter is optional as the authenticator will have stored information about the RP and the user handle that can help the authenticator and user select keys. However, after a user loses Phone A and replaces it with Phone B, Phone B will have none of that stored information. As such, in order for Phone B to know which credentials it should use to recover, the relying party must provide a valid credential id. Unfortunately, relying parties implementing username-less flows will not be able to provide the user with a valid credential id.

5.4.2 Storage

When syncing with a primary authenticator, a backup device needs to generate enough keys in advance for the lifetime of that primary authenticator. Ideally, if backing up Phone A, the backup device will produce exactly the number of backup keys that Phone A will need for future registrations and no more. However, because the backup device cannot possibly know this number beforehand, it may end up generating a large surplus of keys. Then when restoring from Phone A to Phone B, the backup device must create delegations with all of

the existing keys, as it doesn't know which keys Phone A has or has not used as backups in registrations. Phone B also must store all of these delegations whether or not they will ever be used to recover accounts from Phone A.

A cloud option could potentially mitigate this excess creation of keys by recording which keys have been used as backups in registrations. However, doing so raises the risk of compromising linkability with timing attacks, even if the third party can be kept oblivious to the underlying keys.

5.4.3 Computation

Additionally, every time the backup device syncs with a new authenticator, it will have to perform cryptographic signatures for every existing stored key. If it has generated too many excess keys, this could become a very time-consuming and computationally heavy restoration process. For example, imagine the key-wrapping solution mentioned in the discussion of backup devices in Section 5.2.5, where keys are stored in the cloud but signed on a local device. For each of potentially thousands of keys, this backup key would have to fetch, unwrap, and create signatures, which could be a very laborious task.

5.5 Future Work: Reducing Overhead and Enabling Username-less Flows

As mentioned in Section 5.4, the PSK recovery protocol requires the backup device to create a potentially large number of surplus backup keys to ensure that Phone A will not need more. This causes both a storage and computation overhead on backup devices and primary authenticators. To help reduce this overhead, the backup device could generate only a modest number of keys and simply re-generate when Phone A runs low. In this case, Phone A would warn the user that it is running low on backup keys and ask him to re-sync the backup device. At this point, the backup device can generate a modest number of additional keys and send them to Phone A. Phone A can also inform the backup device which of its keys have been registered with relying parties and send the metadata from those registrations to the backup key. When it comes time to recover from a lost Phone A, the backup device can

pass the metadata to the replacement device (Phone B) so that it can select the appropriate credentials upon the next authentication (see the issue raised in Section 5.4 that prevented Phone B from selecting the appropriate credentials in username-less flows). However, note that this type of solution will not notify the backup device of registrations done after the last synchronization, potentially locking the user out of those accounts.

5.6 Conclusion

This Chapter presents the PSK Recovery Protocol, allowing users to recover from lost devices by syncing a backup device with primary authenticators. To reach this solution, the chapter identifies concrete goals and progressively improves straw man approaches until the solution satisfies those listed goals. PSK solves a significant problem in the WebAuthn ecosystem, only requiring a single action to recover all accounts, drastically improving the usability of the recovery procedure without sacrificing the deployability or security properties of WebAuthn. The PSK recovery protocol has the following advantageous properties:

Usability

- *Memorywise-Effortless*: Users only need to remember where they have stored their backup device.
- *Scalable*: PSK scales gracefully to many different accounts with many relying parties.
- *Nothing to Carry*: PSK does not require users to carry additional devices for registrations or authentications.
- *Physically Effortless*: PSK drastically reduces the effort required to recover from all accounts.
- *Easy to Learn and Use*: PSK relies on the standard WebAuthn use cases and does additional operations in ways that are transparent to the user.

- *Infrequent Errors*: By scaling nicely, PSK has the potential to drastically reduce errors.

Deployability

- *Accessible*: Users retain the ability to use many different types of devices and authorization gestures.

- *Server and Browser Compatible*: Implementing PSK is very similar to implementing the Transfer Access Protocol [103]

- *Mature and Non-Proprietary*: PSK relies on the same cryptographic signatures and parameters used in existing WebAuthn specifications.

Security The PSK Recovery Protocol retains all the security benefits of WebAuthn.

However, the PSK Recovery Protocol does suffer from some significant drawbacks. It requires a storage and computation overhead on both backup devices and primary authenticators, potentially increasing the cost of those devices. It also is not usable with username-less flows. We propose solutions to solve these problems in Chapter 6.

Chapter 6

RECOVERING FROM DEVICE LOSS: ONLINE RECOVERY STORAGE

This chapter presents the *Online Recovery Storage* (ORS) protocol to allow users to recover from lost devices in WebAuthn. The underlying mechanisms in this protocol build upon previous work presented in Chapter 4 (Transfer Access) and Chapter 5 (Preemptively Synced Keys). Like those proposals, ORS uses credential binding to tie the trust in new devices to the trust in the credentials on old devices, allowing users to recover from device loss with a single setup procedure rather than forcing them to execute a unique recovery for each account. Further, ORS solves the remaining issues from the PSK recovery protocol, namely, it minimizes the storage and offloads the remaining storage overhead to an oblivious online third party storage entity, removes the computation overhead, and works with username-less flows without sacrificing the security and privacy properties of WebAuthn. The subsequent sections describe the impact on the user and the changes to WebAuthn required to enable such a solution. This work was done in collaboration with Alexei Czeskis and Arnar Birgisson (Google), Hideo Nishimura (NTT Labs), and Tadayoshi Kohno (UW).

6.1 Introduction

The current WebAuthn ecosystem provides secure standards that promise to improve online account security and simplify the experience for internet connected users. This ecosystem allows users to sign into web services through authenticators (for example, a smartphone or dedicated token) that perform user authentication using an asymmetric cryptographic signature that is resistant to phishing attacks and provides two-factor authentication. Similar to the iPhone's TouchID, users on many platforms will have devices, such as phones, that can

serve as WebAuthn authenticators. For example, imagine that a user is using a phone as an authenticator. This phone has an app that allows the user to view and manage keys. It also allows the user to log-in to websites using WebAuthn. When the user goes to example.com and selects "log-in with authenticator", the phone alerts the user to scan a fingerprint. The user complies and the server and authenticator app negotiate in a cryptographic protocol to ensure that the user is safely authenticated and consents to the log-in.

There are, however, some unsolved problems in this ecosystem. Chapter 4 provides a solution to situations where the user retains access to an old device and uses it to set up a new device. In the current WebAuthn ecosystem, when a user gets a new authenticator she must log into all web services independently and register that device. But using the Transfer Access Protocol from Chapter 4, she can set up the new authenticator without having to log in to any sites or perform any additional actions. However, in cases where the user loses her old device and does not have a second registered authenticator there may be no way for her to authenticate and/or register a replacement device. Chapter 5 provides one potential solution to this problem, requiring the user to sync a backup device with each primary authenticator during setup at which time the backup device provides recovery keys for future registrations. When the primary authenticator registers with relying parties, it also registers a recovery key. However, the PSK solution had some potentially prohibitive issues:

1. PSK requires a storage overhead on backup devices and authenticators. Small dedicated devices may struggle to implement the PSK protocol.

2. PSK requires a computation overhead to perform cryptographic operations on all the stored keys during setup. This could prove prohibitive for resource constrained devices, especially if they attempt to perform all computation within a secure element.

3. PSK does not work with username-less flows. Without a username, relying parties do not know which key should provide credentials during recovery, preventing the backup

and replacement devices from delivering the correct credentials.

To solve these issues, this chapter presents the Online Recovery Storage (ORS) protocol. Like the PSK recovery protocol, ORS utilizes a "backup device", but also utilizes an untrusted online recovery storage provider (ORSP) to recover from lost primary authenticators. From the user's perspective, set up of each authenticator requires the same extra step — sync with a backup device, but in ORS that setup requires a connection to the ORSP service. During subsequent registrations, the primary authenticator will both register a backup key and download encrypted data from the ORSP to update metadata about which keys have been registered with each relying party. Doing so solves the three issues with PSK without having to trust the ORSP; the ORSP does not see any of the data because it can be stored as an encrypted blob. If the user loses her primary authenticator, she can restore to a new replacement by syncing with the backup device and ORSP and from that point can use the primary authenticator with all existing accounts.

ORS has the following benefits and caveats:

Benefits

- Scalable recovery from device loss
- Preserves security and privacy benefits of WebAuthn
- Retains the Usability benefits of WebAuthn
- Retains almost all of the Deployability benefits of WebAuthn
- Minimizes storage overhead and offloads storage to the cloud
- Minimizes computation overhead on the backup device and authenticators
- Works with username-less flows

Caveats

- Requires users to acquire and manage a "backup device"
- Requires an initial set up to sync the backup device with each primary authenticator
- Authenticators and backup devices must be able to connect to the internet
- Must rely on an untrusted ORSP for availability during registration and recovery
- Relying parties must implement some kind of credential chaining as in the Transfer Access Protocol in Chapter 4 and allow for registering a recovery public key during a standard WebAuthn registration

6.2 User Experience

From the user's perspective, this solution is very similar to PSK. Like PSK it requires a single local initial setup to exchange data between a backup device and a new primary authenticator after which users can authenticate normally with replacement authenticators for all accounts even when the original authenticator is no longer available. However, during the setup phase, users must ensure that authenticators have access to the ORSP - either directly through their device's internet connection or by connecting to an internet connected client (ex: plugging the authenticator into an internet connected computer). During this step, users may also need to authenticate with the ORSP using their backup device to update the data stored on the ORSP.

Similar to FIDO's recommendation to register multiple authenticators, the ORS protocol described in this chapter uses a second device called a "backup device". However, unlike the FIDO recommendations, users need not manually register the backup device at each account [42]. Instead they have the following additional requirements above the existing basic WebAuthn flow:

1. The user will need to acquire and store a backup device and retain access to it.

2. The user will need to be able to authenticate locally to the backup device in some way via an authorization gesture.

3. The user will need to setup the backup device with an Online Recovery Storage Provider (ORSP).

4. The user will need to retrieve the backup device and use it in a setup protocol each time she sets up a new authenticator.

The subsections below outline the user experience for each user action: setup, registration and authentication on the initial device, recovery, registration and authentication on the new device, transfer access, and moving to new backup devices and ORSPs.

6.2.1 Setup

1. User syncs backup device with primary authenticator (Phone A).

2. User sets up the backup device with OSRP (requires internet connection).

3. User can use the primary authenticator for future authentications and registrations.

   (a) Note that authentications do not require the backup device.

   (b) The relying party is unaware of the ORSP.

6.2.2 Initial registration (Phone A)

1. User does a standard WebAuthn registration (completely unchanged) with the relying party.

2. Primary authenticator (Phone A) updates the ORSP (requires internet connection).

(a) Note that this does not require the backup device.

(b) The relying party is unaware of the ORSP.

(c) This step can potentially happen automatically, transparently to the user.

6.2.3 Authentications (Phone A)

The authentication flow is identical to standard WebAuthn authentications.

6.2.4 Recovery Procedure

1. User syncs backup device with the new replacement primary authenticator (Phone B).

   (a) The backup device updates the ORS (requires internet connectivity).

   (b) Note that this can happen transparently to the user.

2. User tells the backup device which device the new primary authenticator (Phone B) is replacing (Phone A)

3. User uses new primary authenticator (Phone B) for future authentications and registrations

6.2.5 Registrations on replacement (Phone B)

Phone B registers in the same manner that Phone A registered: the user registers Phone B with the relying party in a standard WebAuthn registration and then Phone B must update the ORSP. Again, Phone B can update the ORSP automatically in a manner transparent to the user.

6.2.6 Authentications on replacement (Phone B)

As was the case with Phone A, the authentication flow for Phone B is identical to that of standard WebAuthn authentications.

6.2.7 Transfer Access in ORS

As in Section 5.3.3.8 from the PSK protocol, ideally a recovery procedure would work well with device upgrade, allowing users to execute a standard upgrade protocol like the Transfer Access Protocol [103] from Chapter 4 without having to execute extra steps like syncing with a backup device.

The ORS protocol can be made to do so, allowing users to upgrade devices and carry over syncing information from an old authenticator to a replacement authenticator without having to sync with a backup device. As a result, the user experience ends up being the same as that for the Transfer Access Protocol [103] from Chapter 4.

6.2.8 Moving to new backup devices and ORSPs

Because the ORSP only stores an encrypted blob, moving to a new ORSP is as simple as moving the blob to a new storage provide and letting devices know where the blob is stored. To move to a new backup device, users should be able to run a Transfer Access-like protocol from the old backup device to the new backup device and update the ORSP with a single user action.

6.2.9 Valid ORSPs

ORSPs are simply untrusted storage for encrypted blobs. They can be a dedicated service for recovery or tie in to existing online storage providers. Users can manage this manually if they would like, for example by storing the blob locally on one of their machines or by attaching it to an email. Section 6.4 contains more details about the encrypted blob, but at a high level it contains keys and associated metadata for each account. The blob is encrypted with a symmetric key, so storing all a user's credentials should be less than a few kilobytes (32 byte keys plus metadata for each account).

6.3 Goals

The goal of this work is to restore account access on a replacement authenticator without the user having access to the old primary authenticator. Ideally, this can be done while preserving the desirable properties of the existing WebAuthn scheme. The choices made in the design of this protocol serve to primarily protect the existing security and privacy properties of the WebAuthn protocol while minimizing changes to the user experience and deployability. In addition, the ORS solution aims to solve the remaining drawbacks from the PSK protocol introduced in Chapter 5 by mitigating the storage and computation overhead and enabling use with username-less flows.

This section breaks down the goals for the ORS protocol into categories defined in Section 3.2. Because the goals of this work are very similar to those of the PSK protocol, this section only briefly mentions the categories without further explanations. For more in-depth explanations, see the details in Section 5.2.

6.3.1 Usability

As in the PSK protocol, ORS should require either **no additional action** from the user or, in the worst case, a **single addition action per device**. Although this solution requires users to manage and remember where they have stored the backup device, it should still aim to preserve the rest of the advantageous properties of WebAuthn:

- **Memorywise-Effortless**
- **Scalable**
- **Nothing to Carry**
- **Physically Effortless**
- **Easy to Learn and Use**

- **Infrequent Errors**

6.3.2 Deployability

ORS should also should preserve the following deployability properties:

- **Accessible**
- **Negligible Cost per User**
- **Server and Browser Compatible**
- **Mature**
- **Non-Proprietary**

6.3.3 Security

Most importantly, ORS should preserve all security and privacy protections provided by WebAuthn, both for each step of the recover protocol and for future or past authentications and registrations. From Bonneau et al., those properties are:

- **Resilient to Theft/Require consent**
- **Not Reliant on Trusted Third Parties**
- **Unlinkable**
- **Cannot be Copied by External Observation**
- **Cannot be Copied by Internal Observation**
- **Phishing Resistant**

- **Resilient to Throttled/Unthrottled Guessing**
- **Unaffected by Leaks from Other Verifiers**
- **Resilient to Targeted Impersonation**

As in Chapter 5, ORS should also provide the following additional security properties:

- Defend Against MITM: Attackers who Man-In-The-Middle the connection, for example between the browser and relying party server, should not gain an advantage by attacking during any step in the Recovery Protocol.
- Session-Duplication: The protocol should not aid in the ability for stealing credentials to result in session-duplication, for example, by exposing long-term cookies or passwords.
- Prevent Session Riding: The protocol should not allow an adversary to gain access to an existing session or to a future existing session.
- Trusted Hardware: The protocol should allow the relying party to verify that it trusts the new hardware before allowing access.
- Detecting Clones: The protocol should allow for the continued detection of potential authenticator clones by keeping a counter.

Out of Scope: As in previous chapters, certain attacks are out-of-scope for this work. We do not consider protecting against a malicious authenticator as existing authentications would not be secure independent of the recovery protocol. Attackers who can compromise the local wireless environment and attack the network are also out of scope, as are attackers who can compromise the operating system or browser. Because the ORS protocol grants access to authenticator functionality, rather than performing an independent security and

privacy analysis of each piece of this protocol, this chapter aims to present a protocol that introduces no *additional* vulnerabilities.

6.3.4 Goal Conditions

Before diving into potential workable solutions for recovering from a lost authenticator, we discuss the assumptions and the properties constituting goal conditions for the Recovery Protocol. To start, we have the following assumptions:

6.3.4.1 Initial Assumptions

- The user has a backup device and replacement Phone B.
- The user no longer has access to the original primary authenticator (Phone A), which may or may not have keys and associated metadata, each associated with an account
- Phone B does not have any existing keys
- The backup device and Phone B can create a "secure channel"
  - This secure channel is out of scope for the Recovery Protocol. We assume that this channel can only be set up by a legitimate user who explicitly allows the restoration of Phone B from the backup device. For the purposes of this paper, we assume this channel allows communication between the two phones that is resilient to all possible attacks, including eavesdropping and Man-In-The-Middle attacks.
  - Requiring a user to explicitly authenticate with each device (backup device and Phone B) may be in scope.
- The backup device and Phone B can access the ORSP

6.3.4.2 Target Goal Conditions (for each account on Phone A)

At the conclusion of this protocol, we expect the following properties to hold for each account:

- Phone B has a "restored key"
- The relying party removes Phone A's access
- The relying party adds access for Phone B so that it will be able to authenticate in the future using standard WebAuthn with the restored key.
- **Security Goals**: Throughout each step of the procedure, we expect the Recovery protocol to give attackers no advantage in attacking WebAuthn.

6.3.4.3 Final Recovered State

At the conclusion of the setup of the replacement device (Phone B), Phone B should have credentials that can authenticate to *all* of the accounts originally set up on Phone A. As was the case with Phone A before it was lost, Phone B should be able to register with new accounts with no extra user actions. Should Phone B be lost, the backup device should be able to restore all Phone B's accounts to its replacement (Phone C).

6.4 Technical Solution

This section outlines the details at each step that enable the user experience set up in Section 6.2 to satisfy the goals outlined in Section 6.3 while supporting username-less flows without incurring a storage and computation overhead on user devices. The steps are:

1. Setup
2. Registration with Phone A
3. Authentication with Phone A

4. Recovery to Phone B

5. Registration with Phone B

6. Authentication with Phone B

After the overview in Section 6.4.1, each subsection details the key components involved in each step and the state of each device involved following the completion of the step.

6.4.1 Overview

There are a number of key insights that motivate the design of ORS. Namely, ORS aims to reduce on-device storage, preserve privacy from the ORSP, allow delegation to devices without copying keys, and allow devices in the user's ecosystem to identify each other.

6.4.1.1 Reduce on-device storage

First, to alleviate the storage overhead on devices, ORS uses a similar strategy to the resource constrained key-wrapping security keys, which physically store key pairs and metadata off-device encrypted with a wrapping key that never leaves the device [66]. When the device needs to perform cryptographic operations, it retrieves the encrypted data and decrypts it with its wrapping key to perform those operations. Therefore each device in ORS will have its own unique *"Security Wrapping Key"* to decrypt keys and metadata necessary for operations in this ecosystem. A device's *Security Wrapping Key* never leaves that device, preserving a key security aspect of WebAuthn — credentials unique to each device that cannot be copied across devices.

6.4.1.2 Privacy from the ORSP

In Lang et. al. [66], relying parties store the encrypted key data and deliver it to devices to unwrap before authentication. However, in ORS all those keys are stored together at the ORSP. In order to prevent the ORSP from linking or tampering with that data, devices

encrypt it before giving it to the ORSP. This means that all the devices must share a decryption key called the *"Privacy wrapping key"*. Because this key only serves to keep data private from the ORSP, copying it across devices does not equate to copying key data nor does it violate the security properties of WebAuthn. Each device's credentials are still encrypted with its own unique *Security Wrapping Key* so that no other device can use those credentials.

6.4.1.3 Delegating

Credentials are unique to each device and cannot be copied to other devices. This means that whenever a device creates a key pair, only it can ever use those keys. If it stores those keys on the ORSP, other devices may be able to see that they exist, but they won't be able to use them to sign data or transfer them to other devices who will be able to use them. In order to transfer or delegate access from those keys to other devices, ORS uses a similar idea to the credential chaining from the Transfer Access Protocol [103] in Chapters 4 and 5.

6.4.1.4 Trusting other devices

Lastly, devices within a user's ecosystem (their backup device and various authenticators) will each need to update and change the data stored on the ORSP. Some of those operations require identifying data from another device and using it in operations without the other device being present. In order to enable that, each device in a user's ecosystem has a key pair (*IDPubKey, IDPrivKey*) unique to that device which it uses to sign data and identify itself to other devices. This way, other devices can trust that data it leaves with the ORSP actually belongs to that device. Devices can also use the keys associated with those identifiers to authenticate with the ORSP if the service enables that type of authentication.

In addition, the user should give each device a user-identifiable name that can be tied to the *IDPubKey* so that should the user need to replace devices in the future, she can identify data associated with that device.

6.4.2 Setup

Setup is the first step in the recovery protocol and occurs when a user first gets a new device and sets it up as a recoverable authenticator. This requires that the user obtain a backup device and have an online recovery storage provider (*ORSP*) where they can store a data blob. The setup proceeds as follows

1. Primary Authenticator (Phone A / PA) and Backup Device (BD) create long-lived public/private key pairs (*IDPubKeyPA / IDPrivKeyPA*) and (*IDPubKeyBD / IDPrivKeyBD*). The public keys are long-term identifiers used to identify each device to other devices in the user's ecosystem. For example, *IDPubKeyBD* helps the user's primary authenticator identify data associated with the backup device.

2. Backup Device (BD) creates a *Privacy Wrapping Key* (PWK). This key will be shared with all devices for which this device serves as a backup and will encrypt all data before storage to keep it private from the ORSP.

3. User creates secure channel between Phone A and the Backup Device (BD), who can exchange their identifiers.

4. Backup Device (BD) sends Privacy Wrapping Key (PWK) to Primary Authenticator (PA).

5. Backup Device (BD) generates († some number) of key pairs:

   $$(PubKeyBD1/PrivKeyBD1),\ (PubKeyBD2/PrivKeyBD2),\ etc.$$

   It encrypts each private key with its *Security Wrapping Key* (SWKBD), and creates a signature over each public key with the private key corresponding to its identifier (*IDPrivKeyBD*) so that the other devices can verify those keys are owned by a trusted Backup Device.

6. Backup Device (BD) creates a data blob including the results from step 5, *IDPubKeyBD*, and *IDPubKeyPA*. If BD also backs up other devices (OD*), it should also include their identifiers in the blob (*IDPubKeyOD**).

7. Backup Device (BD) encrypts the data blob with the Privacy Wrapping Key (PWK) and stores this at the ORSP.

† Note that the backup device should generate (at first setup) enough key pairs to last for all registrations from all authenticators backed up by this device. ($number\ of\ devices * lifetime\ number\ of\ accounts$). It does not store these locally.

Figure 6.1 shows the state of each device at the conclusion of the setup phase.

6.4.2.1 State of the Backup Device After Setup

At the conclusion of the setup phase, the backup device has the following stored:

- (*IDPubKeyBD / IDPrivKeyBD*) — Identity key pair that it generated to identify itself to other devices

- (*IDPubKeyPA)* — Public key identifying the primary authenticator with which it was set up originally

- (*SWKBD*) — Symmetric security wrapping key for decrypting credentials

- (*PWK*) — Symmetric privacy wrapping key created by the backup device to encrypt and decrypt the blob stored at the ORSP

- Attestation key pair used to attest to the hardware.

6.4.2.2 State of the Primary Authenticator (Phone A) After Setup

At the conclusion of the setup phase, Phone A has the following stored:

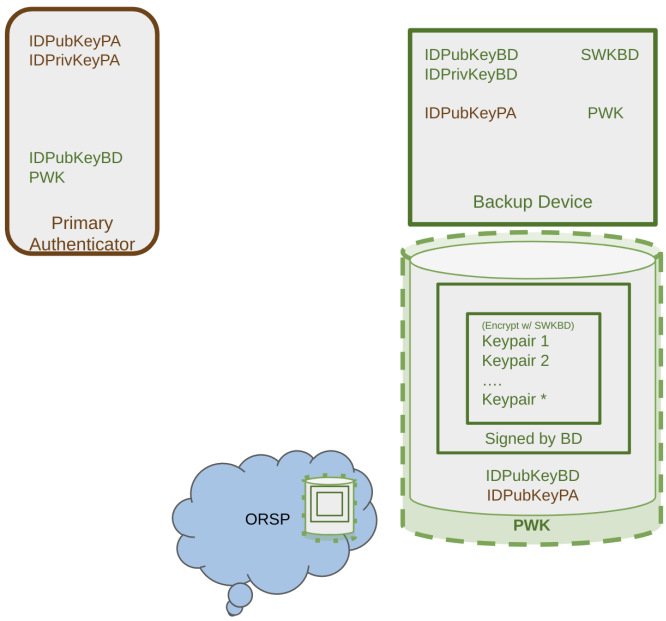


Figure 6.1: State of each entity at the conclusion of the setup phase. The three entities pictured are the primary authenticator Phone A (brown rounded rectangle), the backup device (green rectangle), and the online recovery storage provider (blue cloud). The green cylinder is the data blob encrypted with the privacy wrapping key. The data blob can move between each of the other three entities, but only the primary authenticator and the backup device can read and write it. Green objects are generated by the backup device (BD). Brown objects are generated by Phone A (PA)

- (*IDPubKeyPA / IDPrivKeyPA*) — Identity key pair that it generated to identify itself to other devices.

- (*IDPubKeyBD*) — Public key identifying the backup device which will be used to restore credentials created by Phone A.

- (*PWK*) — Symmetric privacy wrapping key generated by the backup device to encrypt and decrypt the blob stored at the ORSP

- Attestation key pair used to attest to the hardware.

6.4.2.3 State of the ORSP After Setup

The ORSP has the encrypted blob given to it by the backup device. This blob contains the following:

- (*IDPubKeyBD*) — Public key identifying the backup device that gave it the blob.
- (*IDPubKeyPA*) — Public key identifying Phone A
- A set of public keys and their associated private keys encrypted with the backup device's security wrapping key (*SWKBD*). The backup device created these keys to back up Phone A's future registrations. The public keys and associated encrypted private keys are signed by the backup device using *IDPrivKeyBD* so that other devices can verify that the backup device owns those keys.

These components make up the plaintext of the blob, which is encrypted with the PWK. As such, the ORSP cannot read any of the data listed above.

6.4.3 Initial registration (Phone A)

Once setup completes, the user can store the backup device and use Phone A as a normal WebAuthn authenticator for registrations and authentications. Registrations are very similar to standard WebAuthn, with an additional ORSP update step. They proceed as follows:

1. Standard WebAuthn registration
   (a) Primary authenticator generates key pair (*RP1PubKeyPA / RP1PrivKeyPA*) for future authentications with the relying party (RP1)
   (b) Registers public key (*RP1PubKeyPA*) with the relying party (RP1)

2. Primary authenticator retrieves the encrypted data blob from the ORSP and decrypts it with the privacy wrapping key (*PWK*).

3. Primary authenticator adds information about the registration with RP1 and signs that information with its identity private key *IDPrivKeyPA*.

   (a) RP1PubKeyPA

   (b) Metadata including user handle, credential descriptor, domain, and relying party id (*RPID*)

4. Primary Authenticator selects an unused backup public key (*PubKeyBD1*), checks the signature to make sure it matches its stored *IDPubKeyBD*, and creates a delegation from its own *RP1PrivKeyPA* to the selected backup public key (*PubKeyBD1*).

5. Primary authenticator updates the decrypted blob to mark all relevant keys as "used" and associates them with information about the registration with RP1.

6. Primary authenticator encrypts the blob with the privacy wrapping key (*PWK)* and updates the ORSP.

Figure 6.2 shows the state of each device after Phone A registers with RP1.

6.4.3.1 State of the Relying Party after Phone A registers with RP1

At the conclusion of the registration with RP1, the relying party has the registered key created by Phone A (*RP1PubKeyPA*) as it would after a normal WebAuthn registration. It has no knowledge of any of the other components of the system, including the ORSP.

6.4.3.2 State of the Backup Device after Phone A registers with RP1

The backup device is not included in the registration, so its state does not change at all from Section 6.4.2.1.

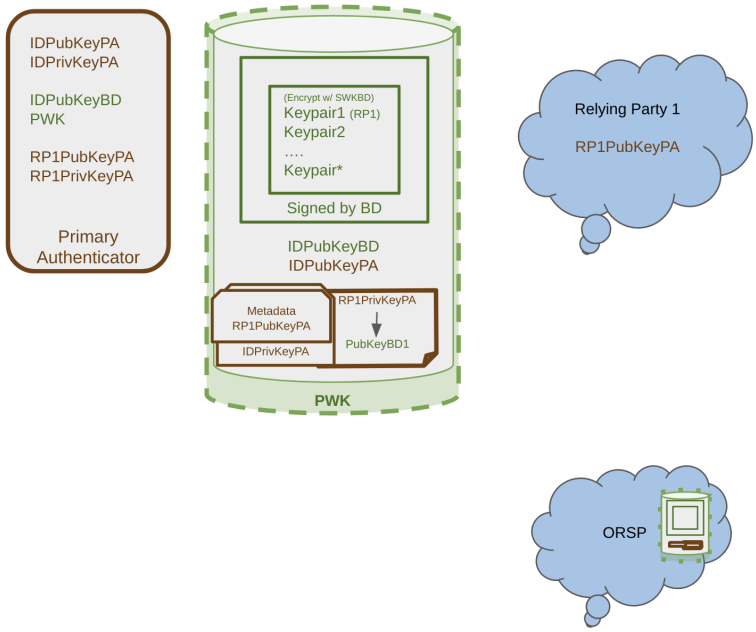


Figure 6.2: State of each entity at the conclusion of a registration with RP1. The three entities pictured are the primary authenticator Phone A (brown rounded rectangle), the relying party (top blue cloud), and the online recovery storage provider (bottom blue cloud). The green cylinder is the data blob encrypted with the privacy wrapping key. Of the pictured entities, only the primary authenticator can read and manipulate the data blob, which it stores on the ORSP encrypted with the privacy wrapping key. Green objects are generated by the backup device (BD). Brown objects are generated by Phone A (PA)

6.4.3.3 State of the Primary Authenticator (Phone A) after it registers with RP1

At the conclusion of the registration with RP1, Phone A has the same data from Section 6.4.2.2 in addition to standard WebAuthn data that results from a standard registration. Additional components **bolded**:

- (*IDPubKeyPA / IDPrivKeyPA*) — Identity key pair that it generated to identify itself to other devices.

- (*IDPubKeyBD*) — Public key identifying the backup device which will be used to restore credentials created by Phone A.

- (*PWK*) — Symmetric privacy wrapping key generated by the backup device to encrypt and decrypt the blob stored at the ORSP

- Attestation key pair used to attest to the hardware.

- ***RP1PubKeyPA/RP1PrivKeyPA*** — the key pair and associated metadata resulting from a standard WebAuthn registration with RP1.

6.4.3.4 State of the ORSP after Phone A registers with RP1

The ORSP has the encrypted blob which has just been updated by Phone A. This blob contains the following (additional items **bolded**):

- (*IDPubKeyBD*) — Public key identifying the backup device that gave it the blob.

- (*IDPubKeyPA*) — Public key identifying Phone A.

- A set of public keys and their associated private keys encrypted with the backup device's security wrapping key (*SWKBD*). The backup device created these keys to back up Phone A's future registrations. The public keys and associated encrypted private keys are signed by the backup device using *IDPrivKeyBD* so that other devices can verify that the backup device owns those keys.

- **Metadata and *RP1PubKeyPA*** — data about the registration with RP1, signed with *IDPrivKeyPA* so that other devices know it is valid and created by Phone A.

- **Delegation from *RP1PubKeyPA* to *PubKeyBD1*** — This delegation should be similar to those from the Transfer Access Protocol [103] from Chapter 4 and Chapter 5,

using the corresponding private key (*RP1PrivKeyPA*) and attestation private keys to create a credential chain from Phone A to the backup device.

These components make up the plaintext of the blob, which is encrypted with the PWK. As such, the ORSP cannot read any of the data listed above.

6.4.4 Authentications (Phone A)

The authentication procedure is exactly the same as standard WebAuthn.

6.4.5 Recovery Procedure

When the user loses Phone A and replaces it with Phone B, the user must run a setup procedure that is similar to that which she ran to setup Phone A in Section 6.4.2. However, instead of an initial setup procedure, the user instructs the backup device that Phone B will replace Phone A. Remember that only Phone A can use the credentials it generated. Since the user no longer has Phone A, those credentials must be replaced by trusted keys. Since the relying parties don't know about Phone B, the backup device must delegate trust to any new credentials generated by Phone B. The procedure then is as follows:

1. Create a secure channel between the new primary authenticator, Phone B, and the backup device so they can exchange IDs and the privacy wrapping key (PWK).

2. User indicates which device Phone B should replace (replaces Phone A).

3. Backup device gets up-to-date blob from ORSP and decrypts it with the privacy wrapping key (PWK).

4. The backup device looks through all public keys actively registered with relying parties. For each of Phone A's actively registered public keys (*RP*PubKeyPA*), the backup device asks Phone B to generate a key pair (*RP*PubKeyPB / RP*PrivKeyPB*) to

replace Phone A's credentials. Phone B will use these new credentials to authenticate with the relying party in the future.

5. Phone B gives each generated public key (*RP*PubKeyPA*) to the backup device.

6. For each of the public keys given to it by Phone B (*RP*PubKeyPB*), the backup device uses its security wrapping key (*SWKBD*) to decrypt the corresponding backup key (*PrivKeyBD**) that it generated for Phone A's recoveries. It uses this key to create a Transfer Access [103] style delegation from that backup private key to the newly generated public key (*RP*PubKeyPB*) generated by Phone B.

7. The backup device gives the delegation chain it created in Step 6 to Phone B along with associated metadata.

8. Repeat from step 6 until all keys have been delegated.

9. Backup device encrypts the blob with the privacy wrapping key (*PWK*) and sends it to the ORSP.

Figure 6.3 shows the state of each device after setting up Phone B as a replacement for a lost Phone A.

6.4.5.1 State of the Backup Device after the Recovery Procedure

After the user sets up Phone B as a replacement for Phone A, the backup device has the following components (items added at this step are **bolded**):

- (*IDPubKeyBD / IDPrivKeyBD*) — Identity key pair that it generated to identify itself to other devices

- (*IDPubKeyPA)* — Public key identifying the primary authenticator with which it was set up originally

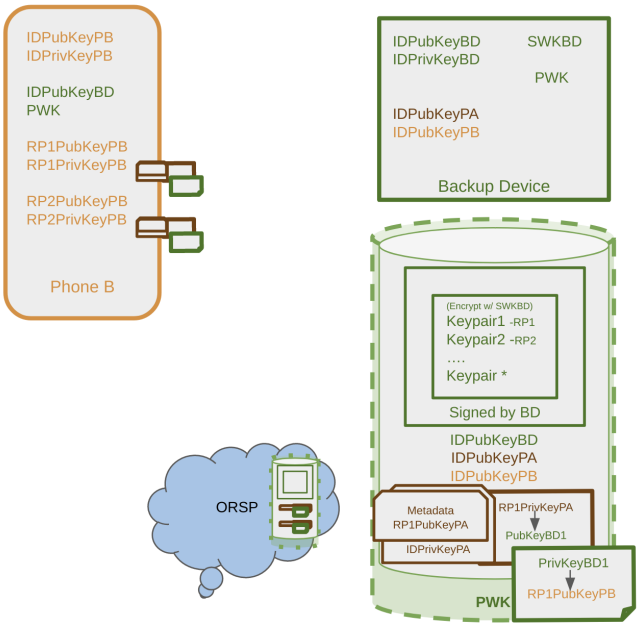


Figure 6.3: State of each entity at the conclusion of the recovery phase. The three entities pictured are the primary authenticator Phone B (orange rounded rectangle), the backup device (green rectangle), and the online recovery storage provider (blue cloud). The green cylinder is the data blob encrypted with the privacy wrapping key. The data blob can move between each of the other three entities, but only the primary authenticator and the backup device can read and write it. Green objects are generated by the backup device (BD). Brown objects are generated by Phone A (PA). Orange objects are generated by Phone B (PB).

- (***IDPubKeyPB)*** — Public key identifying the replacement authenticator, Phone B, with which is was just set up.
- (*SWKBD*) — Symmetric security wrapping key for decrypting credentials
- (*PWK*) — Symmetric privacy wrapping key created by the backup device to encrypt and decrypt the blob stored at the ORSP
- Attestation key pair used to attest to the hardware.

6.4.5.2 State of the Replacement Authenticator (Phone B) after the Recovery Procedure

After the user sets up Phone B as the replacement for Phone A, Phone B has the following components stored (items used in recovery beyond the standard setup of Phone B are in **bold**):

- (*IDPubKeyPB / IDPrivKeyPB*) — Identity key pair that it generated to identify itself to other devices.
- (*IDPubKeyBD*) — Public key identifying the backup device which will be used to restore credentials created by Phone B.
- (*PWK*) — Symmetric privacy wrapping key generated by the backup device to encrypt and decrypt the blob stored at the ORSP
- Attestation key pair used to attest to the hardware.
- **Delegations from Phone A** — For each set of credentials Phone A created, Phone B stores a delegation chain transferring access from Phone A through the backup device. See Figure 6.4.

$$PhoneA \rightarrow BackupDevice \rightarrow PhoneB$$

- **Corresponding Credentials** — Each of the above delegations delegates to a credential created by Phone B. Phone B must store that key pair (*RP*PubKeyPB / RP*PrivKeyPB*) and associated metadata for future authentications.

6.4.5.3 State of the ORSP after the Recovery Procedure

The ORSP has the encrypted blob given to it by the backup device. This blob contains the following (information added during the recovery step in **bold**):

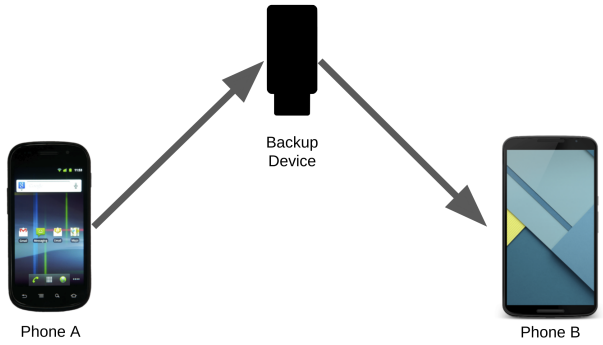


Figure 6.4: The recovery chain as stored on phone B after the recovery phase. The delegation goes from Phone A to the backup device to Phone B. Note that the delegation contains signatures from each device. Phone A signs a delegation from Phone A to the backup device. The backup device signs a delegation from the backup device to Phone B. Phone B will provide a signature during the next authentication with the relying party. (Images for Phones A and B from [112]. Image for backup device from [98])

- (*IDPubKeyBD*) — Public key identifying the backup device that gave it the blob.
- (*IDPubKeyPA*) — Public key identifying Phone A.
- (***IDPubKeyPB***) — Public key identifying Phone B.
- A set of public keys and their associated private keys encrypted with the backup device's security wrapping key (*SWKBD*). The backup device created these keys to back up Phone A's future registrations. The public keys and associated encrypted private keys are signed by the backup device using *IDPrivKeyBD* so that other devices can verify that the backup device owns those keys.
- Metadata and *RP1PubKeyPA* — data about the registration with RP1, signed with *IDPrivKeyPA* so that other devices know it is valid and created by Phone A.

- Delegation from *RP1PubKeyPA* to *PubKeyBD1* — This delegation should be similar to those from the Transfer Access Protocol [103] from Chapter 4 and Chapter 5, using the corresponding private key (*RP1PrivKeyPA*) and attestation private keys to create a credential chain from Phone A to the backup device.

- **Delegation from *PubKeyBD1* to *RP1PubKeyPB*** — This delegation is again similar to those from the Transfer Access Protocol [103] and the one above. The delegation uses the private key (*PrivKeyBD1*) and attestation private keys on the backup device to create a credential chain from the backup device to Phone B.

These components make up the plaintext of the blob, which is encrypted with the PWK. As such, the ORSP cannot read any of the data listed above.

6.4.6 Registrations on replacement (Phone B)

Registrations on the replacement device proceed exactly as they did before on Phone A. See Section 6.4.3.

6.4.7 Authentications on replacement (Phone B)

The next time the user logs in to the account at RP1, Phone B will have to recognize that the credential descriptor points to credentials that need recovery. From the user's perspective, this will act like a normal authentication, but behind the scenes, Phone B must deliver a recovery delegation chain from Phone A to the backup device to Phone B in order to recover Phone A's access going forward. Figure 6.5 shows an example that chain being delivered to an RP.

1. At next login, Phone B looks up credentials for the relying party. If the user selects a credential descriptor linked to a delegation (for example for RP1), Phone B will deliver the delegation chain

$$(RP1PrivKeyPA \rightarrow PubKeyBD1) + (PrivKeyBD1 \rightarrow RP1PubKeyPB)$$

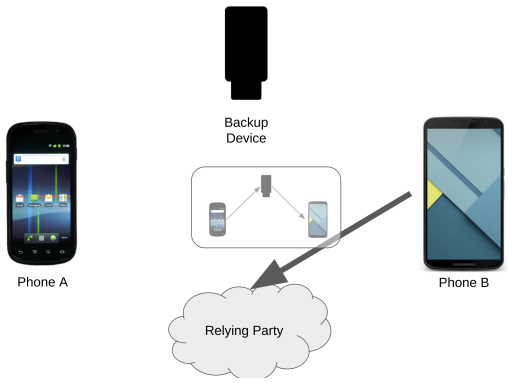


Figure 6.5: Phone B delivers the recovery chain to the relying party upon the next authentication for that account. Like all authentications, this requires Phone B to provide a signature over a challenge. (Images for Phones A and B from [112]. Image for backup device from [98])

.

2. The Relying Party checks the delegation chain and if valid, removes access from Phone A and adds access for Phone B.

3. Phone B updates the ORSP, removing information about Phone A for RP1 and replacing it with its own, including a credential delegating from its new active primary key (*RP1PrivKeyPB*) to the backup key (*PubKeyBD1*) for future recoveries.

$$RP1PrivKeyPB \rightarrow PubKeyBD1$$

Figure 6.6 shows the state of each device after Phone B first authenticates with RP1 after receiving the delegation chain from the backup device.

6.4.7.1 State of the Backup Device after Phone B Authenticates with RP1

The backup device is not involved in the authentication so its data does not change.

6.4.7.2 State of the Replacement Authenticator (Phone B) after it Authenticates with RP1

After Phone B authenticates with the relying party and delivers the recovery chain, it removes the recovery delegation chain for RP1 as it has already delivered it to RP1. Phone B still has recovery delegation chains for other RPs where it has not yet authenticated and thus has not yet delivered the respective recovery delegation chains. Beyond removal of the recover delegation chain for RP1, the data on Phone B is unchanged from Section 6.4.5.2.

6.4.7.3 State of the ORSP after Phone B Authenticates with RP1

The ORSP has the encrypted blob updated by Phone B after its first authentication. This blob removes the extra delegation from *PrivKeyBD1* to *RP1PubKeyPB* since it has already been delivered. It also re-associates the metadata and RP1 account with *IDPubKeyPB* instead of *IDPubKeyPA* and updates the stored key for that account to *RP1PubKeyPB*. Beyond that, all data is the same as in Section 6.4.5.3.

6.4.8 Transfer Access in ORS

ORS should work well with Device Upgrade as does PSK. Should a user upgrade Phone A to Phone B, they can create Transfer Access messages from Phone A's credentials to Phone B's credentials so that during the next authentication, Phone B can update the RPs. To enable recovery from device loss using ORS, after receiving the Transfer Access messages, Phone B simply needs to create delegations from each of its keys to the corresponding recovery keys stored at the ORSP. After the next authentication, Phone B can update the ORSP to remove the binding between Phone A's credentials and the updated accounts.

6.4.9 Moving to new backup devices

Moving to a new backup device can utilize the Transfer Access Protocol [103] as well. The new backup device simply needs delegations from each of the old backup device's active private keys to one of its own generated backup keys. Then the new backup device needs to

remove the excess pre-loaded backup keys and generate its own. In some cases, users will want to "split" backup keys. For example, say Backup Device 1 (BD1) backs up Phone A, B, and C. The user gets Backup Device 2 and wants BD2 to back up Phone A, but not phones B and C. For simplicity's sake, this document does not explicitly outline each step of this procedure, but enabling this type of user flow should be possible.

6.5 Caveats

6.5.1 Timing Attacks

Because devices contact the ORSP after each registration, there is the possibility of timing attacks between a colluding RPs and an ORSP to link between a device and accounts at each RP. For example, say a user registers their primary authenticator, Phone A, at `www.obscureexample.com` at 11:59:59 and then updates the ORSP at 12:00:00. If `www.obscureexample.com` and the ORSP gather data about when each device updates the data blob and when users register, they may be able to link a device to an account.

6.5.2 ORSP Availability

One major caveat of the ORS approach is that it relies on the ORSP for availability during recoveries and registrations. If the ORSP is not available immediately following a registration, it is possible that a user will lose their recently registered device before updating the ORSP causing account lockout even though the user has a valid backup device. Less pressingly, if users replace devices and don't have access to the ORSP, their account access cannot be restored to their new device. These issues should be easily mitigated by using highly available ORSPs.

6.5.3 Complexity

The ORS system introduces a third party (ORSP) and additional complexity with multiple wrapping keys and updates that must occur. Additionally, it adds requirements on

devices (authenticators and backup devices) that may be prohibitive, hurting the usability and deployability of such a scheme.

6.6 Summit Feedback

I presented this solution to a group of FIDO members interested in recovering from device loss at a summit at the University of Washington in 2018. Those in attendance included Aiki Matsushita (DDS), Alexei Czeskis (Google), Arnar Birgisson (Google), Christiaan Brand (Google), Anthony Nadalin (Microsoft), Samuel Weiler (MIT/W3C), Jan Suhr (Nitrokey), Matt Lourie (Nok Nok Labs), Hideo Nishimura (NTT Labs), Salah Machani (RSA), Tadayoshi Kohno (University of Washington), Alex Takakuwa (University of Washington), David Treece (Yubico), Derek Hanson (Yubico). The following is a summary written in collaboration with those in attendance.

The goals of the summit were as follows:

1. Create a list of requirements for solutions to *Recovering from Device Loss*

2. Determine next steps

6.6.1 Requirements

Those in attendance came to agreement on the following requirements:

6.6.1.1 Usability Requirements

- We are trying to solve this problem (Recovering from Device Loss) in a scalable way. In other words, we require solutions to allow WebAuthn/FIDO2 users to recover all accounts with one recovery action per device instead of requiring a recovery action for each RP for each device.

- There are cases where users may want to recover only a subset of accounts using a given mechanism. See item #3 under User Choice.

- Users should not have to carry multiple devices simultaneously for authentication or registration events.

- We discussed whether recovery processes should have a revoke / temporary-revoke/ resume state comparable to putting a temporary block of a lost (but not known to be stolen) credit card, but did not explicitly place this in scope for recovery work. We should discuss further whether this is necessary for recovery protocols or whether recovery and revocation should be separate.

6.6.1.2 User Choice

- Should users decide to use a recovery method with a lower "Security Assurance Level" (ex: copying keys, using a third party for federation, etc), they should be able to do so as long as that type of recovery has a "Security Assurance Level" that meets or exceeds the minimum requirements of the Relying Party.

- Should users decide to recover with each RP instead of recovering with a single action, they should be able to do so (fallback to status quo).

- It may be useful for users to be able, at the time of each registration, for the user to be able to opt-out of recovery for that account - i.e. if the user wishes to NOT empower a recovery device or service to recover that particular account.

- Similarly, users should be empowered, at the time of each registration, to prevent enumeration of a particular account by the normal recovery device or service.

6.6.1.3 Relying Party Choice

- We should allow a way for RPs to specify any registration or recovery security requirements should they choose to enforce them.

- RPs should also be allowed to deny registrations or recoveries coming from or going through insufficient devices/services, or deny recoveries altogether.

6.6.1.4 Security Goal

We discussed whether users should be able to see when recoveries have occurred (non-concealable recoveries). Some solutions provide this, while others do not. Though we don't know whether this is in scope for any standards push, it merits future discussion.

There was considerable discussion about different use cases for different security levels. For most consumer accounts, an account recovery procedure that relies on lower level security may be fine for most users, but proposals should allow for recoveries that satisfy the above requirements in both "high" and "low" security cases.

However, we recognize that users may have difficulty differentiating between high and low security for many apps (ex: social media, email). Further discussion is required to determine what the user flow should be in these cases. For example, should a user be prompted to select the level they want compared to having it default to, e.g., low security? Should this be left up to users or relying parties?

6.6.2 Next Steps

Attendees also agreed on the following next steps to help reach the requirements listed above:

- Expand the "Security Assurance Level" to account for recoveries
- This may require a framework to allow Relying Parties to describe the "Security Assurance Level" to authenticators ahead of time
- We would like to push this to a standard.
- Find a place to push standards changes. Proposals are ISO and IETF, but further discussion is necessary.

- Evaluate Concrete Proposals

- Proposals should be discussed and evaluated against the above requirements as well as the usability/security goals for different user groups. Ideally, we will continue to refine proposals until we are comfortable enough to push a solution to spec.

6.6.3 Adapting Proposed Protocols

Based on the feedback from the summit, protocols proposed in Chapters 4, 5, 6 could benefit from changes to allow for different assurance levels.

6.7 Conclusion

This chapter presents the Online Recovery Storage (ORS) protocol, allowing users to recover from lost devices in WebAuthn without sacrificing the security and privacy properties of the existing WebAuthn Scheme. ORS builds upon the Transfer Access and PSK proposals presented in Chapters 4 and 5 by using simple cryptographic constructs to solve the remaining issues in the PSK protocol. As a result, ORS works with username-less flows and removes the computation and storage overhead from end user devices. Further, it maintains the usability benefits of PSK by reducing user burden to an **additional action per device**. In summary, the ORS protocol has the following advantageous properties:

Deployability

- *Accessible*: Users retain the ability to use many different types of devices and authorization gestures.

- *Server and Browser Compatible*: Implementing ORS is very similar to implementing the Transfer Access Protocol [103], but significant code complexity may limit the availability of devices supporting this type of recovery.

- *Mature and Non-Proprietary*: ORS relies on the same cryptographic signatures and parameters used in existing WebAuthn specifications.

Usability

- *Memorywise-Effortless*: Users only need to remember where they have stored their backup device.

- *Scalable*: ORS scales gracefully to many different accounts with many relying parties. The lack of a storage overhead means devices won't be constrained by large numbers of credentials either.

- *Nothing to Carry*: ORS does not require users to carry additional devices for registrations or authentications.

- *Physically Effortless*: ORS drastically reduces the effort required to recover from all accounts.

- *Easy to Learn and Use*: ORS relies on the standard WebAuthn use cases and does additional operations in ways that are transparent to the user.

- *Infrequent Errors*: By scaling nicely, ORS has the potential to drastically reduce errors, but higher code and system complexity may contribute to more errors than standard WebAuthn.

Security The ORS protocol retains all the security and privacy benefits of WebAuthn.

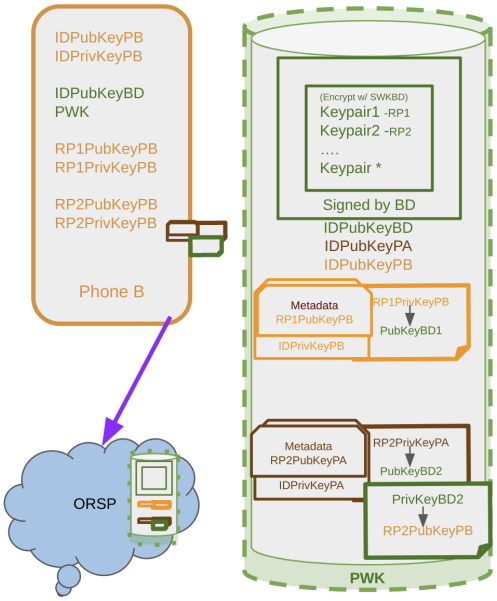


Figure 6.6: State of each entity at the conclusion of the Phone B's first authentication with RP1 after receiving the recovery delegation chain from the backup device. The two entities pictured are the primary authenticator Phone B (orange rounded rectangle), and the online recovery storage provider (blue cloud). The green cylinder is the data blob encrypted with the privacy wrapping key. The data blob can move between each each entity, but only the primary authenticator can read and write it. Green objects are generated by the backup device (BD). Brown objects are generated by Phone A (PA). Orange objects are generated by Phone B (PB).

Chapter 7

FUTURE WORK

In addition to identifying issues regarding *device upgrade* and *device loss*, over the course of this work we also identified other aspects of the WebAuthn specifications that would benefit from academic research.

Security of Second Factors - There are a number of proposed local second factors for the WebAuthn authentication scheme, ranging from iris scans, to fingerprints, to pins, to passwords. As users try different second factors, researchers will need to analyze the security and usability trade-offs of these factors to determine how they should be best implemented in the framework of WebAuthn.

Enabling privacy preserving first pairings for WebAuthn registrations and authentications - When a user employs a phone as an authenticator to give access to browser sessions on a personal computer, the PC's browser and the phone need a way to set up a secure communication channel. For devices that have paired in the past, we can simply re-use those connections. For example, the PC can store trusted Bluetooth authenticators and query the authenticators in range when the user tries to authenticate. This will cause a notification to pop up on the user's phone, which he can use to authenticate on the website. However, for devices that have no previous pairing relationship, a broadcast message leaks information to all Bluetooth devices in range that a certain account holder is trying to log in. Applying privacy preserving cryptography, like a Bloom Filter, may allow the PC and authenticator to setup a secure connection using privacy preserving broadcast messages. If we cannot come up with an optimal (no privacy leaks and no user necessary user interaction) solution to this problem, we can study the different methods of Bluetooth pairing and create a framework to discuss the pros and cons of each approach.

Integration with the Internet of Things - Many previous studies have investigated the ability for web authenticators to function as authenticators for physical devices in the Internet of Things (IoT). They found that users stand to benefit from using powerful authenticators and modern authentication schemes in the IoT [74]. In some preliminary tests, users reported being very satisfied with a door-control system that allowed for phone-based control, tiered access, and delegation. The authors find that users adapted to the system and were able to use it sufficiently fast, in some cases authenticating quicker than they could with traditional physical door keys [16], despite a clunky user interface and much slower hardware and software than we have today. Other studies reveal physical layer attacks that IoT authentication schemes must consider, such as defending against *Mafia Fraud Attacks* [113]. Further, Hayashi et al. [52] show that users actually preferred some constraints. For example, users reported enjoying being forced to knock to unlock computers, rather than having it done at the press of a button. In all, there were many unexpected results when researchers observed use of modern authentication schemes in the physical world, indicating that we will need to continue to research this space to fully realize the benefits of modern authentication schemes.

Chapter 8

CONCLUSION

This dissertation presents the landscape of web authentication including an analysis of issues with existing systems and proposals which solve these issues. It highlights the WebAuthn specifications from the W3C which aim to replace passwords as the dominant form of web authentication and identifies key areas in which the WebAuthn ecosystem can improve. In particular, this dissertation focuses on two of the important remaining problems: allowing users to easily upgrade devices and recover from device loss.

Chapter 4 presents the *Transfer Access Protocol* to solve the former problem, allowing a user to upgrade devices in the WebAuthn ecosystem without requiring that she perform any additional task and without sacrificing any of the security benefits of the WebAuthn scheme. We implemented this solution on a public FIDO software reference implementation and presented the work at a FIDO plenary for feedback. The *Transfer Access Protocol* uses a credential signature chaining scheme that helps inform subsequent proposals for device loss.

Chapter 5 presents one such proposal — Pre-emptively Synced Keys *(PSK)* — a protocol allowing users to recover from device loss with an offline *backup device*. The user must sync new authenticators once during setup in order for the authenticator to be recoverable. This solution allows those authenticators to deliver credential chains similar to those from the *Transfer Access Protocol* to recover from lost authenticators without requiring additional user action. However, *PSK* suffers from a few drawbacks. Notably, it cannot work with usernameless flows and carries a computation and storage overhead on backup devices and authenticators.

To mitigate these drawbacks, Chapter 6 presents a final proposal — Online Recovery Storage *(ORS)* — to allow users to recover from device loss with a backup device and an

untrusted third party for available storage of metadata. As in *PSK*, a user must sync the backup device with authenticators once during setup in order for new authenticators to be recoverable, but instead of storing information pre-emptively on each device, *ORS* stores data with an untrusted Online Recovery Storage Provider (ORSP). To keep data private from the ORSP, *ORS* encrypts all stored data with a wrapping key shared with all the user's recoverable devices. Storing the metadata with the ORSP allows *ORS* to both remove the storage and computation overhead on user devices and enables authenticators to work with usernameless flows. Further, as in previous proposals, *ORS* does not sacrifice any of the security or privacy benefits of the existing WebAuthn protocol.

I hope that this work and others like it will provide a foundation for addressing the remaining problems preventing the web from moving away from passwords and toward modern authentication schemes.

BIBLIOGRAPHY

[1] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky thirteen: Breaking the tls and dtls record protocols. *2013 IEEE Symposium on Security and Privacy*, pages 526–540, 2013.

[2] FIDO Alliance. Fido alliance membership agreement. `https://fidoalliance.org/wp-content/uploads/FIDO_Alliance_-_Membership_Agreement.pdf`, 2017.

[3] FIDO Alliance. Functional certification overview. `https://fidoalliance.org/certification/`, 2017.

[4] FIDO Alliance. Client to authenticator protocol (ctap). January 2019.

[5] FIDO Alliance. Fast id online alliance. `https://fidoalliance.org`, 2019.

[6] FIDO Alliance. Members: Bringing together an ecosystem. `https://fidoalliance.org/participate/members/`, 2019.

[7] Fadi A. Aloul, Syed Zahidi, and Wassim El-Hajj. Two factor authentication using mobile phones. *2009 IEEE/ACS International Conference on Computer Systems and Applications*, pages 641–644, 2009.

[8] Mohamed Alsharnouby, Furkan Alaca, and Sonia Chiasson. Why phishing still works: User strategies for combating phishing attacks. *Int. J. Hum.-Comput. Stud.*, 82:69–82, 2015.

[9] Apple. Two-factor authentication for apple id. `https://support.apple.com/en-us/HT204915`, 2016.

[10] Apple. About touch id advanced security technology. https://support.apple.com/en-us/HT204587, 2017.

[11] Apple. Use face id on iphone x. https://support.apple.com/en-us/HT208109, 2017.

[12] AWPG. Anti-phishing working group - phishing activity trends report. https://apwg.org/resources/apwg-reports/, 2017.

[13] Leah Bachman. Never lose access to lastpass with account recovery on mobile. https://blog.lastpass.com/2019/05/never-lose-access-lastpass-account-recovery-mobile.html/, May 2019.

[14] Dirk Balfanz, Alexei Czeskis, Jeff Hodges, J.C. Jones, Michael B. Jones, Akshay Kumar, Angelo Liao, Rolf Lindemann, and Emil Lundberg. Web authentication: An api for accessing public key credentials level 1. In *W3C Recommendation*, March 2019.

[15] Natã M. Barbosa, Jordan Hayes, and Yang Wang. Unipass: design and evaluation of a smart device-based password manager for visually impaired users. In *UbiComp*, 2016.

[16] Lujo Bauer, Lorrie Faith Cranor, Michael K. Reiter, and Kami Vaniea. Lessons learned from the deployment of a smartphone-based access-control system. In *SOUPS*, 2007.

[17] Allan Beaufour and Philippe Bonnet. Personal servers as digital keys. *Second IEEE Annual Conference on Pervasive Computing and Communications, 2004. Proceedings of the*, pages 319–328, 2004.

[18] Vijay Bharadwaj, Hubert Le Van Gong, Alexei Czeskis, Jeff Hodges, Michael Jones, Rolf Lindemann, Akshay Kumar, Christiaan Brand, Johan Verrept, Jakob Ehrensvard, Mirko J. Ploch, and Mattieu Antoine. Fido 2.0: Client to authenticator protocol. In *FIDO Alliance Review Draft*, October 2017.

[19] Joseph Bonneau. The science of guessing: Analyzing an anonymized corpus of 70

million passwords. *2012 IEEE Symposium on Security and Privacy*, pages 538–552, 2012.

[20] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. *2012 IEEE Symposium on Security and Privacy*, pages 553–567, 2012.

[21] Joseph Bonneau and Soren Preibusch. The password thicket: Technical and market failures in human authentication on the web. In *WEIS*, 2010.

[22] Joseph Bonneau and Stuart E. Schechter. Towards reliable storage of 56-bit secrets in human memory. In *USENIX Security Symposium*, 2014.

[23] John G. Brainard, Ari Juels, Ronald L. Rivest, Michael Szydlo, and Moti Yung. Fourth-factor authentication: somebody you know. In *ACM Conference on Computer and Communications Security*, 2006.

[24] Sean Cassidy. Lost pass. January 2016.

[25] Lulu Yilun Chen and Yuji Nakamura. Cryptocurrency cyber crime has cost victims millions this year. `https://www.bloomberg.com/news/articles/2017-08-24/cyber-criminals-extracting-a-heavy-toll-from-ethereum-advocates`, 2017.

[26] Mark D. Corner and Brian D. Noble. Protecting applications with transient authentication. In *MobiSys*, 2003.

[27] Emiliano De Cristofaro, Honglu Du, Julien Freudiger, and Gregory Norcie. Two-factor or not two-factor? a comparative usability study of two-factor authentication. *CoRR*, abs/1309.5344, 2013.

[28] Alexei Czeskis. *Practical, usable, and secure authentication and authorization on the Web*. PhD thesis, University of Washington, 2013.

[29] Alexei Czeskis, Michael Dietz, Tadayoshi Kohno, Dan S. Wallach, and Dirk Balfanz. Strengthening user authentication through opportunistic cryptographic identity assertions. In *ACM Conference on Computer and Communications Security*, 2012.

[30] Alexei Czeskis, Karl Koscher, Joshua R. Smith, and Tadayoshi Kohno. Rfids and secret handshakes: defending against ghost-and-leech attacks and unauthorized reads with context-aware communications. In *ACM Conference on Computer and Communications Security*, 2008.

[31] Alexandra Dmitrienko, Christopher Liebchen, Christian Rossow, and Ahmad-Reza Sadeghi. On the (in)security of mobile two-factor authentication. In *Financial Cryptography*, 2014.

[32] Mohamed Hamdy Eldefrawy, Khaled Alghathbar, and Muhammad Khurram Khan. Otp-based two-factor authentication using mobile phones. *2011 Eighth International Conference on Information Technology: New Generations*, pages 327–331, 2011.

[33] Adam Clark Estes. The sony hack gets even worse as thousands of passwords leak. `https://gizmodo.com/sony-pictures-hack-keeps-getting-worse-thousands-of-pa-1666761704`, 2014.

[34] Pam Fessler and Michael Martin. Russians believed to have used spear-phishing in election hacking. `https://www.npr.org/2017/06/18/533438850/russians-believed-to-have-used-spear-phishing-in-election-hacking`, 2017.

[35] Lorenzo Franceschi-Bicchierai. Hacker tries to sell 427 milllion stolen myspace passwords for $2,800. `https://motherboard.vice.com/en_us/article/pgkk8v/427-million-myspace-passwords-emails-data-breach`, 2017.

[36] Frank. Chaos computer club breaks apple touchid. `http://www.ccc.de/en/updates/2013/ccc-breaks-apple-touchid`, 2017.

[37] Masayuki Fukumitsu, Shingo Hasegawa, Jun ya Iwazaki, Masao Sakai, and Daiki Takahashi. A proposal of a password manager satisfying security and usability by using the secret sharing and a personal server. *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 661–668, 2016.

[38] Shirley Gaw and Edward W. Felten. Password management strategies for online accounts. In *SOUPS*, 2006.

[39] Samuel Gibbs. Dropbox hack leads to leaking of 68m user passwords on the internet. `https://www.theguardian.com/technology/2016/aug/31/dropbox-hack-passwords-68m-data-breach`, 2016.

[40] Samuel Gibbs. Phishing attack could steal lastpass password manager details. `https://www.theguardian.com/technology/2016/jan/18/phishing-attack-steal-lastpass-password-manager-details`, 2016.

[41] Vindu Goel and Nicole Perlroth. Yahoo says 1 billion user accounts were hacked. `https://www.nytimes.com/2016/12/14/technology/yahoo-hack.html`, 2016.

[42] Hidehito Gomi, Billy Leddy, and Dean H. Saxe. Recommended account recovery practices for fido relying parties. February 2019.

[43] Google. Set up your new nexus device - nexus help - google support. `https://support.google.com/nexus/answer/6073630?hl=en`, 2017.

[44] Google. Google advanced protection. `https://landing.google.com/advancedprotection/`, 2019.

[45] Amber Gott. Introducing sms recovery to secure your account. `https://blog.lastpass.com/2015/10/introducing-sms-recovery-to-secure-your-account.html/`, October 2015.

[46] Eric Grosse and Mayank Upadhyay. Authentication at scale. *IEEE Security & Privacy*, 11:15–22, 2013.

[47] Robert Hackett. Linkedin lost 167 million account credentials in data breach. `http://fortune.com/2016/05/18/linkedin-data-breach-email-password/`, 2016.

[48] Luke Harding. Top democrat's emails hacked by russia after aide made typo, investigation finds. `https://www.theguardian.com/us-news/2016/dec/14/dnc-hillary-clinton-emails-hacked-russia-aide-typo-investigation-finds`, 2016.

[49] Eiji Hayashi, Lorrie Faith Cranor, Anind K. Dey, and Stuart E. Schechter. Uniauth: Building a human-centered identity management system. 2015.

[50] Eiji Hayashi, Sauvik Das, Shahriyar Amini, Jason I. Hong, and Ian Oakley. Casa: context-aware scalable authentication. In *SOUPS*, 2013.

[51] Eiji Hayashi and Jason Hong. "it's hidden in my computer": Exploring account management tools and behaviors. 2013.

[52] Eiji Hayashi and Jason I. Hong. Knock x knock: the design and evaluation of a unified authentication management system. In *UbiComp*, 2015.

[53] Ki Mae Heussner. Playstation hack: What you need to know. April 2011.

[54] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Xiaodong Song, and David A. Wagner. Smart locks: Lessons for securing commodity internet of things devices. In *AsiaCCS*, 2016.

[55] Mat Honan. How apple and amazon security flaws led to my epic hacking. `https://www.wired.com/2012/08/apple-amazon-mat-honan-hacking/`, 2012.

[56] Iulia Ion, Rob Reeder, and Sunny Consolvo. "...no one can hack my mind": Comparing expert and non-expert security practices. In *SOUPS*, 2015.

[57] Mike Isaac, Katie Benner, and Sheera Frenkel. Uber hid 2016 breach, paying hackers to delete stolen data. https://www.nytimes.com/2017/11/21/technology/uber-hack.html, 2017.

[58] Graeme Jenkinson, Max Spencer, Chris Warrington, and Frank Stajano. I bought a new security token and all i got was this lousy phish - relay attacks on visual code authentication schemes. In *Security Protocols Workshop*, 2014.

[59] Nikolaos Karapanos, Claudio Marforio, Claudio Soriente, and Srdjan Capkun. Soundproof: Usable two-factor authentication based on ambient sound. In *USENIX Security Symposium*, 2015.

[60] Ambarish Karole, Nitesh Saxena, and Nicolas Christin. A comparative usability evaluation of traditional password managers. In *ICISC*, 2010.

[61] S. Karthikeyan, Sophia Feng, Ashwini Rao, and Norman M. Sadeh. Smartphone fingerprint authentication versus pins: A usability study. 2014.

[62] Kellen. Android 5.0 feature: Tap & go restore and restore from specific devices. https://www.droid-life.com/2014/11/25/android-5-0-feature-tap-go-restore-and-restore-from-specific-devices/, 2014.

[63] Tracy Kitten. New trojan exploits mobile channel. https://www.bankinfosecurity.com/interviews/darrell-burkey-i-1730, 2012.

[64] kpaulh. Add registration/authentication extensions for cloud-assisted ble.

[65] Brian Krebs. Adobe breach impacted at least 38 million users. https://krebsonsecurity.com/2013/10/adobe-breach-impacted-at-least-38-million-users/, 2013.

[66] Juan Lang, Alexei Czeskis, Dirk Balfanz, Marius Schilder, and Sampath Srinivas. Security keys: Practical cryptographic second factors for the modern web. In *Financial Cryptography*, 2016.

[67] Yue Li, Haining Wang, and Kun Sun. Personal information in passwords and its security implications. *IEEE Transactions on Information Forensics and Security*, 12:2320–2333, 2017.

[68] Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Xiaodong Song. The emperor's new password manager: Security analysis of web-based password managers. In *USENIX Security Symposium*, 2014.

[69] Jian Liu, Chen Wang, Yingying Chen, and Nitesh Saxena. Vibwrite: Towards finger-input authentication on ubiquitous surfaces via physical vibration. In *CCS*, 2017.

[70] Alexander De Luca and Janne Lindqvist. Is secure and usable smartphone authentication asking too much? *Computer*, 48:64–68, 2015.

[71] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. Power analysis attacks - revealing the secrets of smart cards. 2007.

[72] Mohammad Mannan and Paul C. van Oorschot. Leveraging personal devices for stronger password authentication from untrusted computers. *Journal of Computer Security*, 19:703–750, 2011.

[73] Ziqing Mao, Dinei A. F. Florêncio, and Cormac Herley. Painless migration from passwords to two factor authentication. *2011 IEEE International Workshop on Information Forensics and Security*, pages 1–6, 2011.

[74] Shrirang Mare, Mary Baker, and Jeremy Gummeson. A study of authentication in daily life. In *SOUPS*, 2016.

[75] Daniel McCarney, David Barrera, Jeremy Clark, Sonia Chiasson, and Paul C. van Oorschot. Tapas: design, implementation, and usability evaluation of a password manager. In *ACSAC*, 2012.

[76] Jeffrey Meisner. Microsoft account gets more secure. `https://blogs.microsoft.com/blog/2013/04/17/microsoft-account-gets-more-secure/`, 2013.

[77] Ellen Mesmer. The sony playstation network breach: An identity-theft bonanza. April 2011.

[78] Mozilla. Security/cryptoengineering. 2017.

[79] Mozilla. Web authentication api. 2018.

[80] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. Why do developers get password storage wrong? a qualitative usability study. In *CCS*, 2017.

[81] Lily Hay Newman. The devious netflix phish that just won't die. `https://www.wired.com/story/netflix-phishing-scam/`, 2017.

[82] Christoforos Ntantogian, Stefanos Malliaros, and Christos Xenakis. Gaithashing: A two-factor authentication scheme based on gait features. *Computers & Security*, 52:17–32, 2015.

[83] Ying-Han Pang, Andrew Beng Jin Teoh, and David Chek Ling Ngo. Two-factor cancelable biometrics authenticator. *Journal of Computer Science and Technology*, 22:54–59, 2007.

[84] Bryan Parno, Cynthia Kuo, and Adrian Perrig. Phoolproof phishing prevention. In *Financial Cryptography*, 2006.

[85] Last Pass. The best way to manage passwords. `https://www.lastpass.com/how-lastpass-works`, 2019.

[86] Sarah Pearman, Jeremy Thomas, Pardis Emami Naeini, Hana Habib, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Serge Egelman, and Alain Forget. Let's go in for a closer look: Observing passwords in their natural habitat. In *CCS*, 2017.

[87] Olivier Pereira, Florentin Rochet, and Cyrille Wiedling. Formal analysis of the fido 1.x protocol. 2017.

[88] Thanasis Petsas, Giorgos Tsirantonakis, Elias Athanasopoulos, and Sotiris Ioannidis. Two-factor authentication: is the world ready?: quantifying 2fa adoption. In *EUROSEC*, 2015.

[89] Martin Potthast, Christian Forler, Eik List, and Stefan Lucks. Passphone: Outsourcing phone-based web authentication while protecting user privacy. *IACR Cryptology ePrint Archive*, 2017:158, 2016.

[90] Qiong Pu. An improved two-factor authentication protocol. *2010 Second International Conference on Multimedia and Information Technology*, 2:223–226, 2010.

[91] Samsung. Security. `http://www.samsung.com/global/galaxy/galaxy-s8/security/`, 2017.

[92] Pratik Sarkar, Shawn Fitzgerald, and Juliano Rizzo. Attacks on ssl a comprehensive study of beast, crime, time, breach, lucky 13 & rc4 biases. 2013.

[93] Florian Schaub, Marcel Walch, Bastian Konings, and Michael Weber. Exploring the design space of graphical passwords on smartphones. In *SOUPS*, 2013.

[94] Adi Shamir. How to share a secret. *Commun. ACM*, 22:612–613, 1979.

[95] Michael Sherman, Gradeigh Clark, Yulong Yang, Shridatt Sugrim, Arttu Modig, Janne Lindqvist, Antti Oulasvirta, and Teemu Roos. User-generated free-form gestures for authentication: Security and memorability. In *MobiSys*, 2014.

[96] Maliheh Shirvanian, Stanislaw Jarecki, Hugo Krawczyk, and Nitesh Saxena. Sphinx: A password store that perfectly hides from itself. 2017.

[97] Maliheh Shirvanian, Stanislaw Jarecki, Nitesh Saxena, and Naveen Nathan. Two-factor authentication resilient to server compromise using mix-bandwidth devices. In *NDSS*, 2014.

[98] Silh. Svg silh. `https://svgsilh.com/ru/image/160514.html`.

[99] Alex Simons. Secure password-less sign-in for your microsoft account using a security key or windows hello. November 2018.

[100] Andrew Song. Introducing login approvals. `https://www.facebook.com/notes/facebook-engineering/introducing-login-approvals/10150172618258920/`, 2011.

[101] Sampath Srinivas and Karthik Lakshminarayanan. Simplifying identity and access management of your employees, partners, and customers. April 2019.

[102] Mark Stockley. A gargantuan all-seeing eye is watching you on popular websites. `https://nakedsecurity.sophos.com/2017/11/24/a-gargantuan-all-seeing-eye-is-watching-you-on-popular-websites/`, 2017.

[103] Alex Takakuwa, Alexei Czeskis, and Tadayoshi Kohno. The transfer access protocol - moving to new authenticators in the fido ecosystem. In *UW CSE Technical Reports*, June 2017.

[104] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, Angelique Moscicki, Dan Margolis, Vern Paxson, and Elie Bursztein. Data breaches, phishing, or malware?: Understanding the risks of stolen credentials. In *CCS*, 2017.

[105] Simon Thorpe. How authy 2fa backups work. https://authy.com/blog/how-the-authy-two-factor-backups-work/, December 2016.

[106] Roland M. van Rijswijk and Joost van Dijk. Tiqr: A novel take on two-factor authentication. In *LISA*, 2011.

[107] Roland van Rijswijk-Deij and Erik Poll. Using trusted execution environments in two-factor authentication: comparing approaches. In *Open Identity Summit*, 2013.

[108] W3C. W3c and fido alliance finalize web standard for secure, passwordless logins. March 2019.

[109] Ding Wang and Ping Wang. The emperor's new password creation policies. *IACR Cryptology ePrint Archive*, 2015:825, 2015.

[110] Rick Wash, Emilee J. Rader, Ruthie Berman, and Zac Wellmer. Understanding password choices: How frequently entered passwords are re-used across websites. In *SOUPS*, 2016.

[111] Catherine S. Weir, Gary Douglas, Tim Richardson, and Mervyn A. Jack. Usable security: User preferences for authentication methods in ebanking and the effects of experience. *Interacting with Computers*, 22:153–164, 2010.

[112] Wikipedia. Comparison of google nexus smartphones. https://en.wikipedia.org/wiki/Comparison_of_Google_Nexus_smartphones.

[113] Feng W. Zhu, Matt W. Mutka, and Lionel M. Ni. The master key: a private authentication approach for pervasive computing environments. *Fourth Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM'06)*, pages 10 pp.–221, 2006.