Let's Authenticate: Automated Certificates for User Authentication

James Conners, Stephen Derbidge, Corey Devenport, Natalie Farnsworth, Kyler Gates,
Stephen Lambert, Christopher McClain, Parker Nichols, Daniel Zappala
Brigham Young University
{ james.conners, derbiste, creedev, njf26, kylerg, stephen7150, christopher.mcclain, nicholsp, zappala }@byu.edu

***Abstract*—Passwords have numerous drawbacks, and as a result many systems have been designed to replace them. Password replacements have generally failed to dislodge passwords due to the complexity of balancing usability, deployability, and security. However, despite this lack of success, recent advances with password managers and FIDO2 afford new opportunities to explore system design for password replacements. In this work, we explore the feasibility of a system for user authentication based on certificates. Rather than developing new cryptography, we develop a new *system*, called Let's Authenticate, which combines elements of password managers, FIDO2, and certificates. Our design incorporates feedback from a survey of 397 participants to understand their preferences for system features. Let's Authenticate issues privacy-preserving certificates to users, automatically manages their credentials, and eliminates trust in third parties. We provide a detailed security and privacy analysis, an overhead analysis, and a systematic comparison of the system to a variety of alternatives using a well-known framework. We discuss how Let's Authenticate compares to other systems, lessons learned from our design, and issues related to centralized management of authentication data.**

I. Introduction

Researchers seeking to improve user authentication face a quandary—passwords have well-known flaws, but no alternative has been shown to be better. It's well known that users choose weak passwords [19], that password composition rules can be counterproductive [21], and that users have difficulty remembering strong passwords [69]. As a result, people often choose weak passwords or reuse passwords across sites. Password managers greatly simplify password use, but adoption is far from universal [58] and users may simply use them to store weak passwords [46]. Moreover, even when users do select strong, unique passwords, service providers may instead become a weak link if they fail to follow best practices with respect to password storage. Despite these issues, a thorough review of alternatives by Bonneau et al. found that no replacement scheme comes close to supplying the benefits of passwords, and "none even retains the full set of benefits that legacy passwords already provide" [5]. When replacements offer significant security benefits in comparison to passwords, they are typically more costly to deploy or more difficult to use. Decades of research suggest there is unlikely to be a solution that kills off passwords, and trade-offs among solutions will always exist.

Despite this bleak outlook, in recent years there has been a resurgence of interest in augmenting or replacing passwords with cryptographic authentication. FIDO2 is being touted as the standard for public-key-based authentication on the web and has begun to be adopted by web browsers for both single and two factor authentication. FIDO2 works by having users adopt authenticators, such as a hardware token (*e.g.,* YubiKey) or a smartphone app (*e.g.,* Duo). The authenticator generates a private-public key pair for a website at registration and the website then stores the public key as an identifier. To authenticate to the website, the user requests sign-in, and the website asks the FIDO2 token to sign some piece of data with the private key that matches the stored public key. The token signs the data and returns it, authenticating the user.

While this momentum is encouraging, substantial work is still needed to ensure that cryptographic authentication provides a usable alternative to passwords for users. Recent work has demonstrated users find them easy to use [33], though more work is needed to help users with revocation and account recovery. Currently, if a user loses a passwordless hardware token, they cannot recover their accounts unless they have registered a backup token, individually, with each website where they use the tokens.

In this paper we explore an alternative design, called Let's Authenticate, that provides automated account registration and login, along with simple account recovery when an authenticator is lost. Let's Authenticate is inspired by Let's Encrypt[1] and seeks to similarly make it easy to issue certificates to users, which they can then use to register and login to websites. Certificates are issued based on proving ownership of an account with a Certificate Authority (*e.g.,* FIDO2 hardware token), which allows for easy re-issuance if an authenticator device that stores certificates is lost. Given the above usability results for FIDO2, we believe grounding ownership of accounts in ownership of a token, while providing more scalable account recovery and credential management, can help users make the transition to stronger cryptography. This also enables us to make the user interface similar to password managers, which have already garnered significant users. Let's Authenticate transitions users to managing cryptographic keys rather than passwords.

The primary complications for any system that issues certificates to users are (a) simplifying all interactions, so that users do not need to know anything about cryptography

Network and Distributed Systems Security (NDSS) Symposium 2022
24-28 April 2022, San Diego, CA, USA
ISBN 1-891562-74-6
https://dx.doi.org/10.14722/ndss.2022.24272
www.ndss-symposium.org

[1] https://letsencrypt.org/

to use the system, (b) designing for privacy, since certificates are usually intended to identify the owner, and (c) avoiding dependence on a trusted third party, which would allow a malicious or hacked certificate authority to gain access to a user's accounts. While these are daunting challenges, our design demonstrates that we are able to overcome all three. Software authenticators manage all aspects of certificates for users, privacy is enabled by issuing certificates for separate, random identifiers for each service, and we split authentication between certificates and keys stored in authenticators to prevent a hacked authority from gaining access to accounts.

The Let's Authenticate system combines a number of unique features relative to prior work in the field:

- **Automated issuance and management of certificates**: Automated issuance of certificates to individual users, along with automated management of keys and certificates for users to communicate with relying parties.
- **Strong privacy protections for users**: Unlike most certificate systems, our system ensures that user accounts among relying parties are not linkable and certificates do not include any identifying information.
- **No trusted third party**: Our system does not require a trusted third party, preventing a certificate authority from impersonating users.
- **Centralized administration**: Our system uses FIDO2 hardware tokens to authenticate with a certificate authority. This centralizes authentication for all relying parties a user communicates with, so they only need to complete FIDO2 registration once for each device they use for authentication (instead of once per relying party) and only need to manage backup tokens for one account.
- **Simple account recovery**: Our system makes it easy for users to recover their account with the Certificate Authority and all of their web accounts, even if losing a hardware token associated with the account.

Our system improves on passwordless FIDO2 authentication in significant ways. (1) Users only need to have a backup token for their account with the Certificate Authority (CA), rather than having to register backups with all relying parties. (2) Users don't need to carry their hardware tokens regularly, since they only use them for authorizing and deauthorizing software authenticators. (3) Users can easily recover accounts with relying parties even if they lose all of their software authenticators. (4) If a user loses a hardware token, they only need to revoke it with the CA, not with every individual relying party.

In this paper, we make the following contributions:

- We report on a survey of 397 participants that measures perceptions toward replacing passwords generally and toward specific features that represent trade-offs among password alternatives. This helps shape the design of our system.
- We provide a detailed design of the Let's Authenticate system, which has the benefits listed above.
- We provide source code for the system that includes a certificate authority; a software authenticator for iOS and Android smartphones; a browser extension that acts as a software authenticator for Chrome, Firefox, Edge, and Opera; a standalone software authenticator for Windows, MacOS, and Linux systems; and a variety of test websites and mobile applications that allow a user to login with Let's Authenticate.
- We perform a detailed security and privacy analysis demonstrating how Let's Authenticate protects against a variety of attacks defined in our threat model.
- We include a systematic comparison of Let's Authenticate, passwordless FIDO2, 1Password, and Mozilla Persona, each evaluated using the framework presented by Bonneau et al. [5].

We emphasize that in this paper we do not develop new cryptography. Rather, we develop a new *system*, which combines features from password managers, FIDO2, and certificates. This system offers a unique combination of features and thus unique trade-offs compared to other password replacement systems. We believe recent advances with password managers and FIDO2 afford new opportunities to explore system design for password replacements, and our effort represents a first attempt at exploring these opportunities.

We demonstrate that issuing certificates to users can be made relatively easy to use, while still preserving privacy and avoiding the vulnerabilities that come with trusting a third party. Our systematic comparison shows that Let's Authenticate offers security improvements to a password manager, while preserving their usability properties. Our system also offers many of the security benefits of passwordless FIDO2, with significant usability improvements. Future work includes both lab and long-term usability studies to better ascertain the willingness of users to adopt the system.

II. Related Work

User authentication is a large, well developed space, with a vast number of systems in use and proposed in research. Here we focus on systems designed to replace passwords when authenticating on the Internet, particularly those for login on websites and those designed to scale for use by the general public at large.

In 2012, Bonneau et al. [5] produced a comprehensive review of the password-replacement systems designed and developed to date. These include encrypted password managers [36], [31], proxy-based schemes [20], [44], federated single sign-on [48], [28], [24], [64], graphical sign-on [10], [63], cognitive [30], [66], [25], [59], paper tokens [29], [22], [67], visual cryptography [45], hardware tokens [55], [26], [70], [15], [61], phone-based authentication [42], [40], [34], [16], biometrics [54], [13], [2], [8], and recovery [56], [27], [6]. The authors developed a framework for evaluating each of these systems, using 25 properties that are divided into categories for usability, deployability, and security. This demonstrated that no solution was a "silver bullet" that had only clear advantages over passwords without any relative disadvantages. Rather, each system offers its own trade-offs among these properties, and passwords are likely to be around for a long time.

A. Current Authentication Systems

In the years since the review by Bonneau et al., several systems have become fairly widely used, while many others have never seen significant deployment. Password managers are one system that has become more popular, with about 20

to 30% of Americans using a password manager, though only 5% say this is the method they use the most [9]. LastPass[2] and 1Password[3] are among the more well-known systems. An early study of password managers showed that users had difficulty using several systems, with incomplete mental models being the primary factor [11]. More recent work shows that users tend to adopt password managers for usability rather than security benefits [17], though subsequent work showed that users of separate password managers are driven more by security [46]. Password managers can increase password strength on average [32], and a variety of factors influence whether people use password managers correctly [46]. There are still a variety of security problems plaguing password managers [38].

Federated login is another password alternative that remains popular today. These systems offer single-sign on, allowing a user to create an identity with a specific service such as Facebook, Apple, or Google, and then use that identity to log into other various websites. This reduces the number of passwords users have to remember, but the user is placing their trust in the identity provider, which can track their logins and, if compromised, could access their accounts. Some work has developed a federated login system that provides stronger privacy guarantees [23], [4]. Apple has recently deployed a federated service that provides users with a unique, anonymous email address to use for each site.

B. Cryptographic Authentication Systems

Our focus is on cryptographic systems that are used for single-factor authentication. FIDO2 is the foremost of these systems because of the significant industry effort spent to standardize and develop this alternative. FIDO2 consists of two standards. First, WebAuthn [65], specifies how a web application can use JavaScript for user registration and login. An authenticator handles cryptographic logins and can take many forms such as a software authenticator or a hardware token. Second, CTAP2 [3] specifies a method for web browsers to communicate with external authenticator via USB, BlueTooth, or NFC. The standards specify methods for using hardware and software tokens as both two-factor and one-factor authentication, with the latter often referred to as *passwordless*.

Three recent studies have demonstrated the viability of passwordless FIDO2 and its trade-offs for users. Lyastani et al. conducted a large-scale lab study of passwordless FIDO2 hardware tokens and found that users were highly satisfied with FIDO2 and willing to consider it as a password replacement [33]. However, they also surfaced significant issues, including loss of access if the token is lost or stolen, lack of suitability for everyday use, and a lack of a mental model to understand a token's functionality and security. Revocation and backup are significant open issues for passwordless FIDO2.

Farke et al. deployed passwordless FIDO2 tokens in a small company setting [18], finding that most users opted to still use a browser-based password manager that autofilled passwords instead of FIDO2. Participants were concerned about losing the token and longer authentication times, and they did not see a benefit to using the tokens.

[2]https://www.lastpass.com

[3]https://1password.com

Similarly, Owens et al. found that using smart phones as a roaming authenticator was less usable than passwords [41]. Participants were similarly concerned about availability of smart phone devices, account recovery and backup, as well as issues with setting up the roaming authenticator to log into a service.

Mozilla Persona [24], a now defunct authentication method, was a set of protocols that allowed users to authenticate to websites using a certificate they obtained from their email provider proving ownership of their email address. Persona automated all interactions with certificates, and account recovery required adding a secondary email address. Persona protected user privacy by ensuring that email providers were not given information they could use to track user logins to websites. However, colluding websites could still track users because email addresses were used as an identifier.

Another major past effort designing cryptographic authentication is the SPKI/SDSI system designed by Rivest, Lampson, and Ellison [53]. This system was intended to be simple and bottom-up, allowing names for keys that are local and meaningful to the user. The system was ultimately not successful because it was designed primarily for access control rather than for authentication [52].

Kerberos [37] has classically been used for client-server authentication. Various proposals have been made to incorporate public key cryptography into Kerberos, among them PKDA [57], which distributes authentication away from key distribution centers, eliminating a trusted intermediary. However the system still issues a single, long-term certificate to users and requires a trusted certificate authority.

Cryptographic authentication has been most prominently used in Belgium [14] and Estonia [35], where citizens are issued an electronic identification card, or eID. Citizens can obtain an eID at a municipal center by proving their identity, for example with a passport. Many countries have now adopted eIDs, and some are adding systems that allow a smartphone to be used in place of the eID using a PKI-capable SIM card and a user-assigned PIN.

A relatively new set of systems, called decentralized identity providers, use a blockchain to provide privacy and security for user authentication. These systems use a wallet, which stores a private key for a user on a secure device (*e.g.*, a mobile phone or hardware token). The public key is then published on a public blockchain, such as Sovrin [60]. Once the public key is published, the chosen blockchain will return a did, or decentralized identifier, which is a unique identifier for the user. The did is derived from the public key. When a user attempts to authenticate, the relying party contacts the ledger with the user's did. The ledger responds with the corresponding public key, which then allows the relying party to encrypt a challenge with the public key, to be decrypted and returned by the user's wallet. Currently, recovery in the case of a lost private key (device), has some hurdles to overcome. Some wallets do not provide recovery, while others require users to print off a paper with some number of specific words, in order to regain access to the private key on a new device.

Client certificates in TLS have likewise been available for many years, though rarely used. Using certificates to authenticate users to servers has suffered from many usability

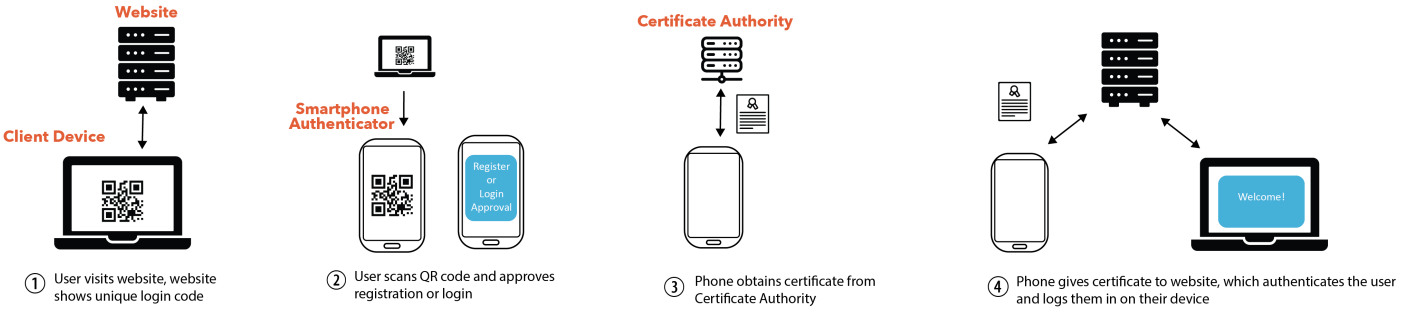


Fig. 1. Initial Conceptual Let's Authenticate design

issues, including difficulties that plague server administration [43], the expense of getting a certificate, and the complexity of managing keys and certificates through complex browser interfaces. O'Neill designed a system to improve all aspects of client authentication in TLS, demonstrating they could be made usable by using the operating system to centralize how certificates are handled system-wide [39]. In comparison, our work uses certificates above TLS and HTTPS and focuses on both privacy and account recovery.

Several academic systems have developed cryptographic authentication systems. Loxin [71] proposed issuing a single certificate to each user, which would be stored in a smartphone app and used to authenticate to web sites. This has obvious drawbacks in terms of privacy. The n-Auth system [47] proposed using public keys and an authentication protocol to enable a smartphone to register and login users on websites. The system does not allow users to have multiple authenticators or to recover accounts after an authenticator is lost. Early work in this space used phone-based systems to help prevent phishing [34], [42].

Finally, cryptographic systems are regularly used to provide second-factor authentication, or 2FA, with hardware tokens such as YubiKey[4] and software systems such as Duo[5] and Google Authenticator[6]. The usability of 2FA systems has been well studied recently, with studies showing that 2FA systems may be difficult to setup [51], but are generally easy to use on a daily basis [51], [49] and do not take significant amounts of time to use in the aggregate [50].

Our previous work [12] introduced an initial design for Let's Authenticate that was similar to the concept we present in §III-A. As discussed in §III-E and §IV, we have substantially modified the design and protocol from this early version based on feedback from a survey.

III. User Input on Design

Due to the long history of failed attempts at replacing passwords for authentication, we wanted to involve users in the design of the Let's Authenticate system. Accordingly, we developed a conceptual design and prototype, then surveyed users about their trade-offs of authentication systems. Our survey was approved by our Institutional Review Board.

[4]https://www.yubico.com

[5]https://duo.com/product/multi-factor-authentication-mfa/two-factor-authentication-2fa

[6]https://support.google.com/accounts/answer/1066447

A. Initial Design

We began with a conceptual design, shown in Figure 1. The user (1) visits a website, which displays a unique login code for that device. The user (2) scans the QR code with their phone and approves the registration or login to the website. The phone (3) obtains a certificate from a certificate authority (CA), attesting to the user's ownership of an opaque identifier that it uses to identify them to the website. The phone (4) provides this certificate to the website and the website can then log the client device into the user's account. Not shown in this picture is that the user authenticates to the CA on their phone using a username and a master password, similar to a password manager. The phone then acts as a certificate manager, storing the certificates it uses for each website.

B. Survey Development

We developed a survey designed to understand user preferences regarding systems that can replace passwords and regarding trade-offs in our conceptual system as compared to several alternatives. For all Likert questions we used a 7-point scale. We conducted a short pilot study of the survey to determine whether the questions were understandable and made minor adjustments prior to collecting results.

For alternative authentication systems we chose to compare Let's Authenticate to passwordless FIDO2 hardware tokens and to password managers. We chose the former because of current industry interest in developing this standard and the latter because of their current popularity as a way to make managing passwords easier (as opposed to replacing them entirely).

The survey first asks participants how important it is to them that websites adopt new systems to replace passwords. We then ask participants about nine different features a system may have, such as needing to carry a smartphone or hardware token with you, or requiring you to memorize a master password such that if you forget it you lose access to all your accounts. For these questions, we examined the properties from Bonneau et al. [5] to find the properties that best illustrate trade-offs among the systems we are comparing, adding some additional trade-offs because neither passwordless FIDO2 nor Let's Authenticate were available when the systematic comparison from Bonneau et al. was created. We then phrased questions in language that was easy to understand and avoided jargon. We randomized the ordering of these features for each participant.

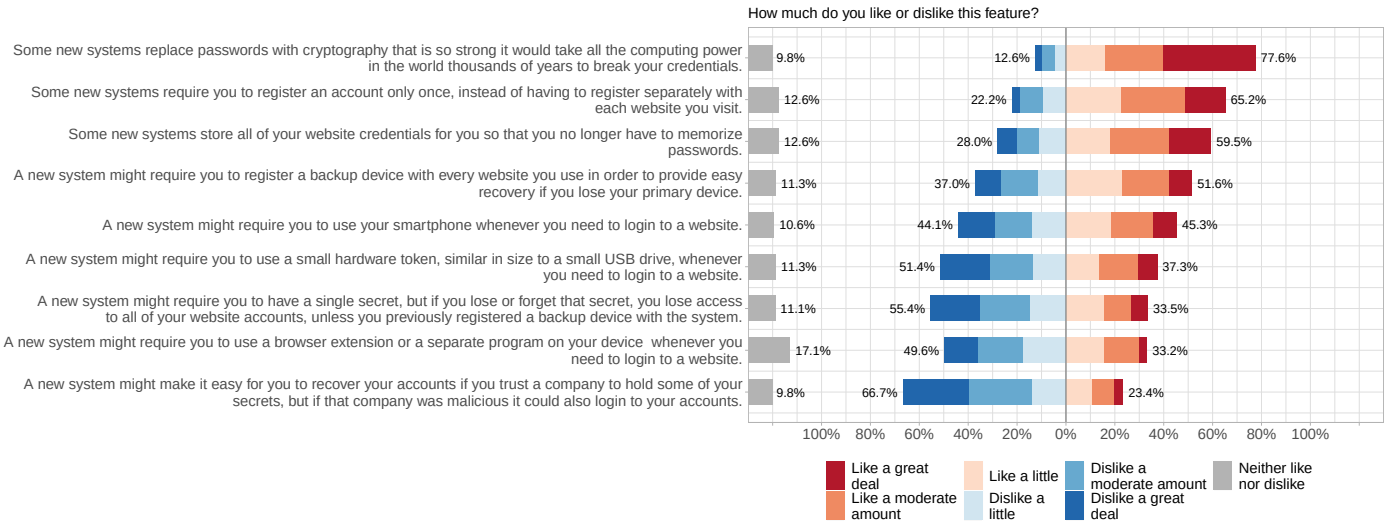


Fig. 2. Positive and negative features of Let's Authenticate, Password Managers, and Hardware-based authentication

C. Recruitment and Demographics

We recruited 417 participants using Amazon's Mechanical Turk. To get reliable (non-spam) responses, we set the qualification requirement to a 96% acceptance rate. We limited our survey to the U.S. population. We removed 20 of the participants for non-completion. We paid each participant $1.50 for completing the survey. The average time to complete the survey was 6 minutes and 9 seconds, which amounts to an average compensation of about $15 per hour, above the United States minimum wage and exceeding the payment standards of the Mechanical Turk community.

The age ranges of participants were as follows: 18-24 (5.5%, N=22), 25-34 (46.6%, N=185), 35-44 (28.7%, N=114), 45-54 (9.1%, N=36), 55-64 (6.8%, N=27), 65-74 (2.5% N=10), and declining to answer (0.8%, N=3). Our sample skewed male (65.5%, N= 259), with fewer female (33.8% N=133), and 0.75% (N=3) self-identified as transgender, gender variant, or nonconforming, while 0.5% (N=2) preferred to not answer.

D. Results

When we asked participants *How important is it to you that websites adopt new systems that replace passwords?*, 52.4% indicated this was at least moderately important, 33.5% indicating it was of slight to no importance, and the rest neutral. Nearly half (49%) indicated they already use a password manager. We asked this question after showing them the animated video of a password manager, to ensure they knew how one worked. This high percentage may bias our results in favor of password managers, since they are already widely used among our sample.

Figure 2 shows participant ratings for authentication system features, ordered from most to least favored. Participants overwhelmingly liked the security benefits of replacing passwords with cryptographic methods and also rated highly usability features such as only having to register for one account and the convenience of not having to remember passwords. Many were accepting of the need to register a backup account in advance to provide account recovery (a primary drawback of hardware tokens), likely viewing the trade-off of backup registration favorably relative to the advantage of easy account recovery. The most disliked features were having to use a hardware token to login, using a high-risk master password, needing to install separate software, and having to trust a third party. Participants reacted more negatively to carrying a hardware token than a smartphone.

We also asked participants about their preference for devices to use when authenticating. Results were split between only a cell phone (34%), only a browser extension (25%), or both (40%).

E. Impact on Design

Based on feedback from our survey, we identified the following changes we should make to the conceptual system design:

- **No trusted third party**: Our conceptual design requires trusting the certificate authority, which could create a certificate that authorizes anyone to login to a user's accounts. While it might be possible to put in place strong management practices for a certificate authority, and to convince users they can place their trust in this party, participants in our survey strongly disliked the idea of trusting the CA. Thus, in our revised design we seek to eliminate this flaw and ensure that users are always protected against third-party access of their accounts.
- **No master password**: Survey participants indicated a strong preference for replacing passwords with cryptography and a strong negative reaction to using a master password. Thus our revised design eliminates the master password in favor of a FIDO2 hardware token, which has received generally positive results from prior user studies [33], [18].
- **Centralized administration**: Survey participants indicated they strongly prefer systems that store all of their credentials and that enable them to register once in a central account, rather than having to complete registration steps with each site they visit. They also prefer not to use

a hardware token every time they login. Thus we seek to create a centralized way for users to manage their account access privileges and avoid using a hardware token for every login.
- **Simple account recovery**: Survey participants indicated a desire to easily recover accounts using a backup device. Thus our revised design provides a way for users to register backup hardware tokens to protect against loss.
- **Device independence**: Numerous participants in our survey indicated a preference for not being required to use a smartphone, hardware token, or browser extension. We thus aim to allow users to use either a smartphone or a browser extension, based on their preference, expanding on our original design that focused exclusively on a smartphone. We also note that a browser could directly implement our system, avoiding the need for an extension.

IV. System Design

Our high level goal for the Let's Authenticate system is to develop a way to automatically issue certificates to users for website login, rather than using bare keys, such as with FIDO2 hardware tokens. We started with our conceptual design, shown in Figure 1, then developed goals for a revised system based on feedback from the survey. We then completed a full system design. Due to the complexity of designing a new authentication system, we focus on authentication to websites, using either a web browser or a mobile application as the front end, leaving exploration of other use cases for certificate-based authentication for future work.

A. Design Goals

Our design of Let's Authenticate system is based on the following goals, many taken from user input from our survey:

1) **Automated issuance and management of certificates**: The system should simplify the entire certificate life cycle for users, with automation employed everywhere. This includes obtaining certificates, configuring their browser to use certificates, and performing other associated tasks, such as protecting private keys and handling revocation.
2) **Strong privacy protections for users**: The system should prevent colluding services from tracking users using identifiers in certificates and prevent a certificate authority from knowing which sites a user authenticates to. This rules out using a single certificate for all services and including any identifying information in certificates such as an email address.
3) **No trusted third party**: The system should not allow the certificate authority to directly access a user's accounts or to issue certificates to unauthorized users.
4) **Centralized administration**: The system should provide a centralized way for users to manage their account access privileges, including authorizing and de-authorizing devices that may access their accounts.
5) **Account protection:** Centralized credential storage must have strong protection from attackers who are able to breach a cloud service. We use best practices taken from password managers, such as encrypting credentials and storing the encryption key only locally on devices.
6) **No requirement to carry tokens:** Users do not need to carry a token and use it to login to relying parties. They only use a FIDO2 hardware token to authorize and deauthorize software authenticators.
7) **Simple account recovery**: The system should make it easy for users to recover their account with the CA and all of their web accounts, even if losing a hardware token associated with the account. A user who owns only a single device used for authentication, such a smartphone or laptop, should still be able to access their account if they lose that device and then get a new one.
8) **Device independence**: The system should allow users to choose either a smartphone or a browser extension, based on their preference.

B. Revised Design

Our system encompasses the following entities, shown in Figure 3. A *Client* is a device that can run a web browser, such as a laptop or smartphone. An *Authenticator* obtains certificates and authenticates a user to a relying party. An authenticator may be a mobile app on a smartphone, a browser extension, or a desktop app. A *Certificate Authority (CA)* issues certificates to a user's authenticator. A *Relying Party* is a server that wants to authenticate users, either for a website or a mobile application.

We focus first on how the system uses certificates to authorize logins. We use the notation (K, S) to refer to a key pair, where K is the public key and S is the private key. The CA owns a key pair (K_{root}, S_{root}).

The system uses the certificates shown in Figure 3:

Authenticator Certificate: An authenticator certificate certifies that an authenticator is authorized to access a user's account with the CA. To obtain an authenticator certificate, the authenticator creates a key pair, (K_{auth}, S_{auth}). The user then accesses their account with the CA using a passwordless FIDO2 hardware token, and authorizes the creation of an authenticator certificate for K_{auth}. The authenticator certificate lists the username and authenticator public key, K_{auth}, and is signed by S_{root}. Renewal of this certificate requires proving possession of the FIDO2 hardware token.

Account Certificate: An account certificate enables an authenticator to prove ownership of an account with a relying party. The authenticator creates a key pair $(K_{account}, S_{account})$ and generates a random $accountID$. It then uses the authenticator certificate to obtain an account certificate, which lists the account public key, $K_{account}$, and the $accountID$. The $accountID$ is unique to a relying party (*e.g.,* a website). This certificate is signed by S_{root}. This certificate authorizes the authenticator to register or login to a relying party (RP).

Session Certificate: A session certificate authorizes the login of a given session with an RP. Each time a client logs into an RP, the RP generates a unique session identifier for the client. The authenticator obtains this $sessionID$ and creates a key pair $(K_{session}, S_{session})$. The authenticator then creates a session certificate that lists the $sessionID$, the domain name of the relying party, and $K_{session}$. This certificate is signed by $S_{account}$. The authenticator needs to present both an account certificate and a session certificate to login.

Every time the authenticator uses a certificate, it proves ownership of the keys that it holds, to prevent a replay attack.

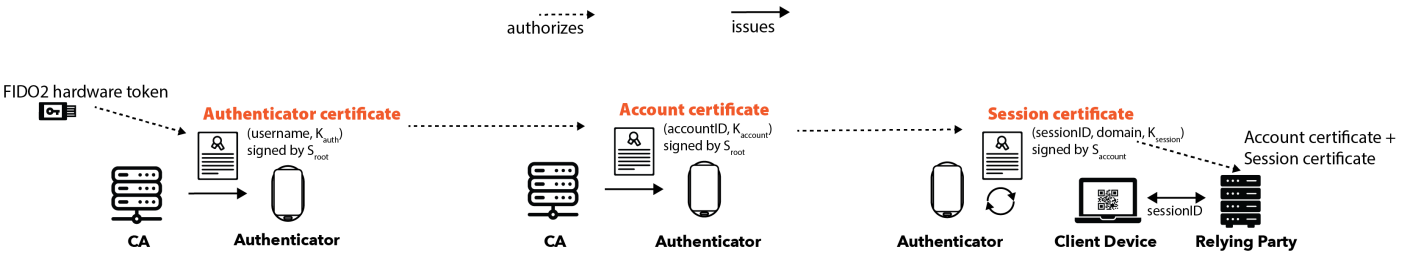


Fig. 3. Certificates used in Let's Authenticate system

An important part of preventing unauthorized access is the system's use of $K_{session}$. This is only accessible to the authenticator and is encrypted when stored (see §IV-F). This prevents a hacked CA from authorizing a login with fake certificates because it does not have access to $K_{session}$ for the account. See §V for a complete analysis.

To protect a user against lost or stolen authenticators, a user can deauthorize an authenticator at any time by proving possession of the FIDO2 hardware token. See §V for details. Limiting tokens to authorizing and deauthorizing authenticators provides major advantages as compared to passwordless FIDO2 authentication because users no longer need to carry tokens regularly or register a backup token at each RP. See §VIII for more details.

Finally, we note that where possible private keys used for authentication should be stored in secure hardware. This provides protection against private keys being inadvertently leaked due to software errors. In our prototype, the only keys stored in software are $K_{session}, S_{session}$.

C. Account Creation and Login

A user creates an account with the certificate authority (CA) using the CA's website and FIDO2 passwordless authentication. The user chooses a username and registers an account with the CA using a FIDO2 hardware token. The CA gives the user the option of registering a backup token to protect against token loss. Note that the use of passwordless FIDO2 here is orthogonal to the Let's Authenticate system design and could be replaced by another form of authentication. We use passwordless FIDO2 because the user's account with the CA is highly sensitive, since it will store an encrypted form of their login credentials for all RPs. The hardware token essentially acts like the user's master password for a password manager.

D. Authenticator Authorization

A user communicates with the CA, through its website, to authorize and deauthorize authenticators to access accounts at relying parties (RPs). Proving ownership of a hardware token associated with their CA account is required to authorize and deauthorize authenticators.

Figure 4 shows the steps to authorize an authenticator. The authenticator first generates an authenticator key pair, (K_{auth}, S_{auth}). It then opens a browser window to communicate with the CA, supplying K_{auth} via a URL parameter. The user should already be logged into their account with the CA, but could complete login if needed using their hardware token. Once logged in, the user can agree to add K_{auth} as an

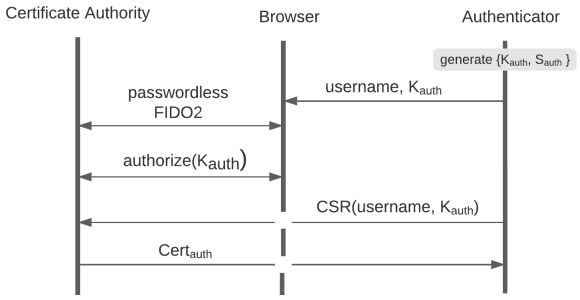


Fig. 4. Authorizing an authenticator

authorized key for their account. Finally, the user can go back to the authenticator and request an *authenticator certificate*, which authorizes the authenticator to access the user's account with the CA. The authenticator sends a certificate signing request (CSR) to the CA; this request includes the username for the account and K_{auth} and it is signed by S_{auth}. This CA only returns an authenticator certificate, $Cert_{auth}$, if K_{auth} has been authorized for the user's account. Note that we have chosen this flow, using a browser, because FIDO2 is currently best supported in browsers; support for browser extensions to use FIDO2 keys is disallowed and support for mobile apps is not universal. If FIDO2 was widely supported, an alternative flow could have the authenticator directly use the FIDO2 hardware key to communicate with the CA.

Authenticator certificates have a default life of 10 days, which we choose to be long enough so that the user does not need to repeatedly re-authenticate with their hardware token, but short enough that the user will likely not lose track of it. Once the certificate has expired, the user will be prompted to re-authenticate with their hardware token, and the authenticator will automatically request a new certificate. A user can configure the lifetime of an account certificate. Some users may prefer to be more careful, by setting the lifetime of the authenticator certificate to 1 day so that they must re-use the hardware token regularly. Other users may prefer greater convenience, by setting the lifetime to infinity so that they only use the hardware token to authorize new devices or deauthorize lost or stolen devices.

To cope with a lost or stolen authenticator, a user can deauthorize an authenticator by likewise proving ownership of the hardware token associated with their account. Figure 5 shows these steps, which are similar to the steps for authorizing an authenticator.

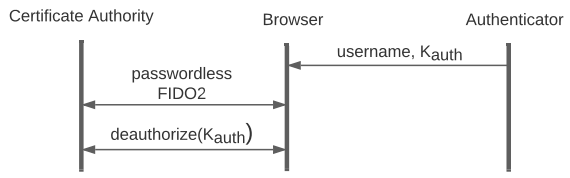


Fig. 5. Deauthorizing an authenticator

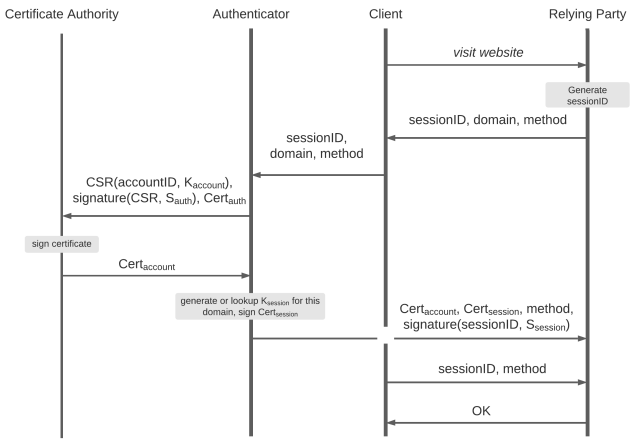


Fig. 6. Relying Party Authentication

We also note that, for convenience, the user assigns an *authenticator name* to each authenticator, to help distinguish among multiple authenticators.

E. Relying Party Authentication

Figure 6 illustrates the process for a user to authenticate to a relying party. Registering and logging in are nearly identical, with the $method$ parameter indicating which operation to complete. The entire process is automated except for the first step.

1) Communicating session information to the authenticator: A user wanting to authenticate to a relying party may either visit a website (using either a traditional computer or a mobile device) or use a mobile app, either of which contacts the relying party's back end server for authentication. To identify clients trying to authenticate, the relying party generates a unique $sessionID$ for the client.

To authenticate with the relying party, the user needs their authenticator to obtain the $sessionID$ for their client. If using a traditional computer, this can be done by having the web browser display a QR code representation of the $sessionID$ (for an authenticator that is a mobile device) or use hidden form fields (for an authenticator that is a browser extension). If using a mobile device, the $sessionID$ can be sent directly to the authenticator software running on the same mobile device using app linking.

If the user scans a QR code for this step, this is counted as the user approving the login. Otherwise, the user is prompted to confirm the login on their authenticator.

2) Obtaining an account certificate: The next step is for the authenticator to obtain an *account certificate* that is signed by a CA, indicating it owns a particular $accountID$. To register an account with an RP, the authenticator generates an $accountID$, which is a random, unique string, and a unique account key pair, (K_{auth}, S_{auth}). It then creates a certificate signing request (CSR), containing the $accountID$ and $K_{account}$ and signed by $S_{account}$. It sends the CSR to the CA along with an additional signature of the CSR using its authenticator private key, S_{auth}. It also includes its authenticator certificate. This enables the CA to verify that the authenticator is authorized to obtain certificates for the username listed in the authenticator certificate and owns both S_{auth} and $S_{account}$. The CA then associates $accountID$ with this username and returns an account certificate, listing the $accountID$ and $K_{account}$.

Account certificates have a very short lifetime, *e.g.,* 1 minute. This provides enough time for the certificate to be used to authenticate with a relying party. To login to an RP after registration, the authenticator obtains a new account certificate for the same $accountID$. Account certificates are not renewed, but are re-issued each time they are needed. This ensures that the authenticator must contact the CA each time it wants to authorize a login. When it does this, the CA can verify whether the authenticator is still authorized for the account, which protects against lost or stolen authenticators being used to access accounts. Contacting the CA for each login has acceptable communication overhead, similar to single sign-on, but with improved privacy since the CA sees only the opaque $accountID$ and does not know which RP the authenticator is logging into. Moreover, most websites, aside from high-security sites like banks, use cookies to keep a user logged in indefinitely, drastically reducing the number of actual logins.

3) Obtaining a service certificate: The next step is to obtain a service certificate to authorize login for a given session identified by a $sessionID$. The authenticator generates and signs its own session certificate. The session certificate contains the $sessionID$, the domain name of the relying party, and $K_{session}$ for that relying party. It is signed by $S_{account}$. The authenticator sends the account certificate, session certificate, and a signature of the $sessionID$ (signed by $S_{session}$) to the relying party, authorizing the registration. The authenticator does not have to be connected to the same network as the client device, or to the client device itself. The relying party stores $K_{session}$ with the account for that user and subsequently will only allow logins from an authenticator that proves it owns $S_{session}$.[7]

For login, the process is nearly identical, but authenticator must look up the $K_{session}$ that it uses for the relying party. This mapping is only accessible to the authenticator and is encrypted when stored (see §IV-F). It then proves that it owns $S_{session}$ by signing the $sessionID$ in order to login.

The expiration time in the session certificate indicates how long the session is authorized for, which allows a relying party to set the expiration time on a session cookie or in its session database. The user can choose a session lifetime by

[7]One might imagine that $K_{session}$ could be replaced with a symmetric key. However, this could lead to the RP accidentally storing it in plaintext (as some sites have done with passwords) and anyone would obtains this key in a breach has valuable information for compromising an account.

selecting high security (10 minutes), medium security (1 day), low security (forever), or temporary (expires after 5 minutes of activity). The temporary login is useful for logins on a public computer. These levels can be customized by the user.

4) Logging out: To logout a session, the client can directly logout as usual with the relying party (*e.g.,* clicking a logout button), or the authenticator can send a logout request directly to the RP. In the latter case, the logout request includes a signature of the $sessionID$ with $S_{session}$.

5) Using multiple accounts: Finally, in the case where a user has multiple accounts with a relying party, they can distinguish these by associating an $accountName$ with each $accountID$. This can be helpful since the $accountID$ is a long, random identifier.

F. Multiple Authenticators

For a user to operate multiple authenticators, the authenticators need to share a mapping of domain names to the $accountID$ used for each domain, along with $K_{session}$ and $S_{session}$ for that domain. This enables an authenticator to obtain an account certificate for the appropriate $accountID$ and then generate a session certificate when needing to authenticate to a relying party.

To avoid impersonation by an attacker (see §V), these are stored in encrypted form with the CA, similar to how some password managers store a password vault in the cloud. We take a similar approach to the design of LastPass [31] and 1Password [1].

Specifically, authenticators manage an *authenticator map*:

$$map = [(domainName, accountName, accountID, K_{session}, S_{session})]$$

and a list of all the authenticators belonging to an account:

$$authenticatorList = [(authenticatorName, K_{auth})]$$

The authenticator generates a secret key and a symmetric key. It then encrypts the recovery data with the symmetric key and encrypts the symmetric key with the secret key:

$$authenticatorData = E(secretKey, symmetricKey)| E(symmetricKey, authenticatorList + map)$$

The authenticator prompts the user to print the secret key, known as a *recovery kit*, so that it can be re-entered on any additional authenticators the user initializes. We note that an alternative to a secret key is to use the HMAC secret key extension in FIDO2, which enables an authenticator to retrieve a secret key from a hardware token.[8]

Each authenticator updates the $authenticatorData$ whenever it registers for a new account with a service and periodically synchronizes the data through its account with the CA. Possession of an authenticator certificate and its associated S_{auth} proves ownership of the account and ability to synchronize data. A simple locking protocol can be used to ensure that no updates are lost in case multiple authenticators try to update the data simultaneously.

Note that users may want to have multiple accounts at a given relying party. The system accommodates this by allowing an authenticator to register for a second account with a relying party and then listing the associated account information in the authenticator map, each with a unique $accountName$.

G. Public Computers and Remote Logout

One advantage that comes from separating an authenticator from a client device is that it is possible to safely login to a public computer without sharing any sensitive authentication data with the computer. A user can have the public computer display a QR code that contains the domain name and session ID, scan it with their smartphone, and authorize the login directly with the relying party.

This same architecture also enables a user to remotely logout of an account with a given relying party, for example if they leave a public computer or a work computer logged into a site and need to logout from a different location. To make this work, each authenticator needs to synchronize session certificates within the authenticator data whenever logging into a relying party. These can be maintained as a list of sessions active for the account ID in the authenticator map:

$$map = [(domainName, accountName, accountID, K_{session}, S_{session}, sessionList)]$$

where

$$sessionList = [(authenticatorName, [(Cert_{session}, geoLocation)])]$$

The session certificates can be removed from this list as the certificates expire. Geolocation information can help a user identify a location where the certificate was used.

By using the authenticator list in the encrypted data along with the map, any authenticator can display a list of all the owner's authenticators, a list of all their accounts being managed with the system, and a list of all their active sessions, including which authenticator authorized each session and when that session expires.

One drawback of providing remote logout is that keeping the $sessionList$ current requires synchronizing the authenticator data with the CA on each login instead of each registration. Thus a CA may not want to offer this service if the overhead is too large. We discuss overhead of the entire system in §VI.

H. Shared Accounts

Users often like to share accounts with their friends and family. Some service providers prefer that their users don't do this. Authentication with usernames and passwords

[8] https://fidoalliance.org/specs/fido-v2.0-rd-20180702/fido-client-to-authenticator-protocol-v2.0-rd-20180702.html#sctn-hmac-secret-extension

makes it relatively easy to share accounts. However, the Let's Authenticate system makes it more difficult to share accounts. To allow a friend to share their account, a user could do one of the following:

- Authorize each login to the system by a friend, by obtaining the session ID and domain (*e.g.,* from a QR code), then approving the login with their authenticator. Their friend can't use their own authenticator since it can't access the authentication data.
- Associate an additional FIDO2 hardware token with their account, and give this to their friend, which provides access to *all* of their accounts. It is unlikely a user would want to do this. However, they could create a second account with the CA and then only add shared services to this account and associate a friend's hardware token with this second account. This could allow them to segment sensitive accounts (*e.g.,* email, banks) from their shared accounts.

I. Account Recovery

A key feature of the Let's Authenticate system is that it allows a user to maintain access to all of their accounts with relying parties even if they lose all of their authenticators. To recover their accounts, a user simply obtains a new authenticator, logs into their account with the CA using their FIDO2 hardware token, and then authorizes the new authenticator for their account, as described above. Once this is complete, they can initialize the authenticator with the secret key from their recovery kit. The authenticator uses its authenticator certificate to authorize downloading that user's authenticator data, and uses the secret key to decrypt the authenticator data. The user now has access to all of their accounts.

Account recovery for a user's account with the CA is a complex problem, similar to account recovery for an account for an online password manager service. When the user first creates an account with the CA, the system prompts the user to associate a backup FIDO2 hardware token with their account, and asks them to store it in a secure place.

In the case that a user loses all hardware tokens associated with their account, and thus access to their accounts managed with various relying parties, recovery of their accounts is still possible with cooperation from relying parties. A relying party can choose to use email reset or some other method that requires the user to establish ownership of their account. In this case, the user may create a new account with the CA, initialize an authenticator, and then use an account recovery procedure at individual relying party websites to associate a new $accountID$ with their account at the relying party.

In the event that the CA is unavailable, a user may be unable to access their accounts with relying parties. This only occurs if an authenticator needs to obtain a new account certificate for a relying party. This in turn only occurs if the user is (a) creating a new account with an RP or (b) logging into an account with an RP after a session has expired. Section §IV-E discusses varying lifetimes for sessions depending on the account's sensitivity or user preference. In practice, many low security accounts today use indefinite session lifetimes, so short-term CA unavailability will generally only affect access to high security accounts. Mitigating this issue requires a CA to provide a high-availability service, following customary practices.

J. Implementation

We developed a prototype of the Let's Authenticate system consisting of (1) a certificate authority, (2) a mobile app authenticator, (2) a browser extension authenticator, and (3) a standalone application authenticator. Our work supports Android and iOS phones, plus the Chrome, Firefox, and Edge browsers. The functionality of an authenticator includes (1) registration with the CA, (2) authentication with sites, (3) listing all currently active authenticators and the sites they are authorized to login to (either by service or by authenticator), (4) logging out of sessions authorized by that authenticator, (5) and deauthorizing an authenticator (and logging out all its sessions). We also developed prototype websites and mobile apps that operate with the Let's Authenticate system, including mockups of Google, Facebook, and Netflix.

V. Security and Privacy Analysis

An attacker has the capability to (1) compromise a relying party, (2) compromise a certificate authority, or (3) obtain physical access to an authenticator or client. We assume the web PKI and TLS are safe from attacks. We also assume that authenticators are safe from malware and any attacks against a user require physical access to their devices.

The Let's Authenticate system prevents passive attacks by using TLS for communication between all pairs of entities in the system, along with the web PKI to ensure authentication among participants.

Our system preserves the following properties:

P1: If an attacker compromises a certificate authority, it cannot identify which relying parties a user authenticates to.

The attacker in this case can observe all of the $accountID$s owned by each user. However, the $accountID$s by themselves do not identify which relying parties a user authenticates to. The authentication map, which does link relying party domains to $accountID$s is encrypted as described in § IV-F, and can only be decrypted with knowledge of the user's secret key. The secret key is not shared with the CA. The attacker is unable to brute force the secret key because it is long and random.

P2: If an attacker compromises a certificate authority, it cannot access any user accounts at relying parties.

The attacker can identify the list of $accountID$s that a user owns. The attacker does not know which domain is associated with each $accountID$ because this information is encrypted in the authentication data. A clever attacker might try to create an account certificate and a session certificate for each $accountID$ and then try these certificates at popular websites. However, the attacker does not know the $K_{session}$ and $S_{session}$ used for this account, so they will be unable to authenticate with the replying party. Recall that $S_{session}$ is encrypted with the authentication data, so it is unavailable to the attacker. The relying party can notify the user, the next time they login, of the failed login attempt, which is indication of a breach at the CA.

P3: If an attacker compromises a certificate authority and a relying party, it can only identify which users authenticate to the compromised relying party but cannot tell which other relying parties a user authenticates to.

The attacker in this case also can observe the $accountID$ used to login to the compromised relying party. However, obtaining a list of other $accountID$s for that same user does not provide the attacker with information it can use to identify other non-compromised relying parties the user authenticates to. As above, information linking $accountID$s to relying parties is encrypted and requires knowledge of the secret key to decrypt.

P4: If an attacker compromises multiple relying parties, the Let's Authenticate system does not provide information that allows the attacker to link a user account at one relying party with their account at another relying party.

A relying party sees only the account certificates and session certificates given to it by an authenticator. The former contain a random $accountID$ and $K_{account}$ and the latter contain a $sessionID$, the domain name of the relying party, and $K_{session}$. Because the $accountID$ and public keys are unique to the relying party, then if these values are generated properly [7], [62], there should be no information in a collection of keys that allows an attacker to form associations among them.

We note that an active attacker could observe IP addresses when users login to compromised relying parties, thus correlating the $accountID$s and keys being used. However, this is true for current authentication systems as well, and we consider solving this outside of the scope of our work.

P5: If an attacker steals an authenticator and is unable to bypass a PIN or biometric lock on the authenticator software, then it is unable to access any of the user's accounts at relying parties that are managed by the authenticator.

An attacker who steals an authenticator will be unable to authenticate to any of a user's accounts, or even observe authentication actions using malware, since the system will not perform any actions until unlocked with a PIN or biometric lock.

P6: If an attacker steals an authenticator and is able to bypass a PIN or biometric lock on the authenticator software, then it is only able to access the user's accounts at relying parties that are managed by the authenticator until the owner notices the theft and deauthorizes that authenticator.

An attacker in this case may be able to access a user's accounts if the device is storing a valid authenticator certificate. This allows the authenticator to obtain an account certificate and then sign a session certificate. Once the authenticator certificate expires, they will be unable to obtain a new one without possession of a hardware token registered to the account with the CA. The authenticator could require possession of the hardware token to decrypt the synchronized authentication data every time it is opened, preventing the attacker from obtaining the session keys. However, for convenience, the authenticator should allow this data to be unlocked with a PIN or biometric, and the hardware token required only once every period, *e.g.,* 10 days.

Most importantly, a user who notices an authenticator has been stolen can deauthorize K_{auth} for that device by proving possession of a hardware token registered with their account. This prevents the authenticator from obtaining or renewing account certificates, thus preventing it from authenticating to any relying party. The short lifetime of account certificates prevents older account certificates from being used after an authenticator is stolen. Thus, as soon as the theft is noticed, access by the attacker can be revoked.

Threats Outside of our Threat Model: A malicious CA could collude with a relying party to gain access to an account, but none of the information stored at the CA helps them gain access. Likewise, a malicious CA could collude with relying parties to track users by sharing which identifiers belong to the same user. An attacker who both steals an authenticator, bypasses the PIN/biometric on the authenticator software, and steals a hardware token registered to the account can impersonate a user, but they cannot authorize a new authenticator without the secret key.

VI. Overhead Analysis

To assess the scalability of the Let's Authenticate system, we examined both storage and communication requirements for the Certificate Authority.

A. Storage Requirements

Table I shows the storage requirements for the Certificate Authority. We focus our calculations there because the CA will need to store data for all users. This table shows the size of each field that the CA stores, along with how this data scales with the number of authenticators, relying parties, and sessions. All calculations come from our implementation of the CA. We omit the storage for the username and info needed for passwordless FIDO2 for the CA account, since these are orthogonal to our system and don't impose significant storage requirements. All keys are generated using the 2048 bit (256 bytes) RSA algorithm and all certificates are generated using these keys. The geolocation has an average size of 294 bytes, which was averaged from 10 locations geographically spread across the United States.

As an approximation, we show the total storage needed assuming a typical user has 3 authenticators, authenticates with 50 RPs, and has active sessions authorized by all authenticators for all RPs. The total amount required for a typical user is 306 KB.

The storage required for the CA is similar to that required for a company that provides cloud synchronization for a password manager. The authenticator data is similar to the password vault, in that it stores credentials and scales per account the user has with a RP. The primary difference is the session list, which is not typically stored by a password manager and scales per session per RP. This also constitutes over half of the storage. We note that the session list is used only for remote logout. This is a convenience feature for remote logout and also allows a user to determine if a lost or stolen authenticator has been used to access one of their accounts. This feature could be removed without affecting the rest of the system.

TABLE I. STORAGE REQUIREMENTS

Data	Field	Size	Number	SubTotal	Total (3 authenticators, 50 RPs)
Account Info	K_{auth}	256 bytes	1 per authenticator	256 bytes	768 bytes
Authenticator List	$authenticatorName$	256 bytes	1 per authenticator	512 bytes	1536 bytes
	K_{auth}	256 bytes	1 per authenticator		
Map	$domainName$	253 bytes	1 per RP	1277 bytes	64 KB
	$accountName$	256 bytes	1 per per RP		
	$accountID$	256 bytes	1 per RP		
	$K_{session}$	256 bytes	1 per RP		
	$S_{session}$	256 bytes	1 per RP		
Session List	$authenticatorName$	256 bytes	1 per session per RP	1597 bytes	240 KB
	$Cert_{session}$	1047 bytes	1 per session per RP		
	$geoLocation$	294 bytes	1 per session per RP		
Total					306 KB

TABLE II. COMMUNICATION OVERHEAD

API Call	End Point Description	Number of Days between each call	API calls/s per 1M users	Number of RPs per user	Total API calls/s per 1M users
/fido/register	Registration with the CA	365	0.03	1	0.03
/fido/login	Login with CA	365	10	1	1.16
/user/:username/authorize	Authorize an authenticator	365	0.03	1	0.03
/user/:username/deauthorize	Deauthorize an authenticator	365	0.03	1	0.03
/users/:username/data	Pushing authentication data to the CA	1	11.57	5	57.87
/users/:username/data	Checking for updated authentication data	1/24	277.78	1	277.78
/user/:username/account	Get account certificate (High security accounts)	1	11.57	5	57.87
/user/:username/account	Get account certificate (Low security accounts)	30	0.39	50	19.29
Total					591.36

The authenticator likewise needs to store the authenticator data, also about 306 KB, which should not pose a barrier. Relying parties need to store just the $accountID$ and $K_{session}$.

B. Communication Requirements

Table II shows the communication requirements for the Certificate Authority. Using each endpoint, we estimate how often each endpoint is called by the authenticator, factor in the number of RPs a typical user may authenticate with, and thus arrive at an estimate for the number of API calls per second that 1 million users would make. We assume that a typical user has accounts with 5 RPs for high security accounts (banks, work accounts) and 50 RPs for lower security accounts. The estimated number of API calls were calculated as 531.36 calls/second for 1 million users.

We note that a little over half of this is due to a conservative assumption about synchronizing authentication data. We assume that every authenticator synchronizes the data every hour, to allow for up-to-date session lists for remote logout purposes. This could instead be done on demand (when the user manually triggers synchronization), or could be rate-limited based on how busy the CA is.

As with storage overhead, the communications overhead for the CA is similar to that required for a company that provides cloud synchronization for a password manager. The primary difference is our conservative assumption about data synchronization, along with the overhead for obtaining account certificates for each login.

To test the communications overhead of our implementation of the CA we performed some load testing using the K6 open source software.[9] This allowed us to automatically generate 1000 virtual users that would generate requests to our CA endpoints. K6 implements virtual users using efficient parallel communication. Our tests indicate the CA is able to successfully handle 355 API calls/second. Thus we estimate that our implementation could conservatively handle about 660K simultaneous users.

It is important to note we deployed our CA with a single Docker container on a single server running Ubuntu. Deployment of additional instances could be automated during high traffic times. Additionally, load balancing would allow for distribution of the servers. Together, these would drastically increase the load that could be handled.

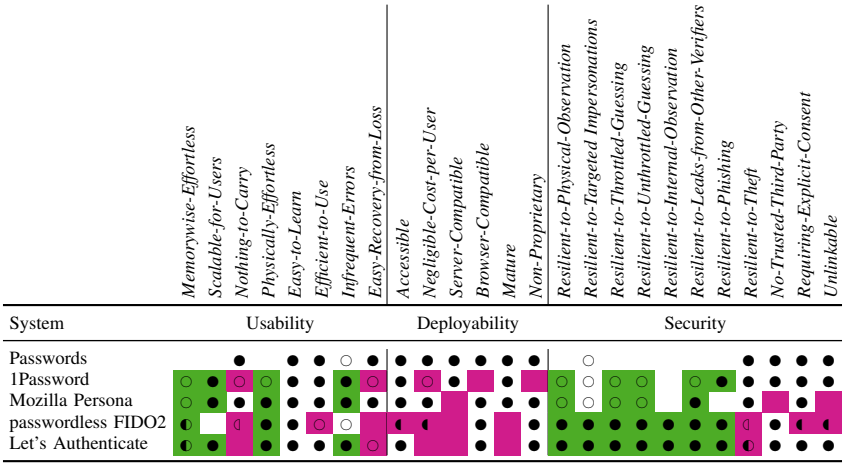
[9]https://www.k6.io/

VII. SYSTEMATIC COMPARISON

We compare authentication schemes in Table III using the framework from *The Quest To Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes*, by Bonneau et al. [5]. Ratings for passwords are taken directly from their work and ratings for 1Password are taken directly from their ratings of LastPass, with one exception. LastPass was originally rated as having a trusted third party because of a leak of their password database, but 1Password uses a PAKE protocol and a secret key that is mixed into the master password on the client side. We rate Mozilla Persona, passwordless FIDO2, and Let's Authenticate using the definitions for the properties in the above paper. The properties are grouped, in order, as usability, deployability, and security/privacy properties. We provide a detailed justification for our ratings of Let's Authenticate, Mozilla Persona, and FIDO2 in the appendix.

Overall, Let's Authenticate rates similarly to 1Password with respect to usability, has significant improvements with respect to security, and lags in deployability. The similarities in usability are due to both using centralized management

TABLE III. RATING OF AUTHENTICATION SYSTEMS



● offers the benefit, ○ almost offers the benefit, ◐ ◖ ◗ depends on the authenticator type or website, (blank) no benefit
better than passwords, worse than passwords

of secrets or keys and having similar user interfaces. Let's Authenticate could thus help users transition from password managers to cryptographic authentication. The security benefits of Let's Authenticate originate in its use of cryptography rather than passwords. This likewise leads to its relatively poor deployability, since servers need to be updated to use Let's Authenticate certificates and the system is less mature.

We also note some significant differences between Let's Authenticate and passwordless FIDO2 hardware and software tokens. Many of these stem from the trade-offs of using certificates versus bare keys. Both systems have strong security properties, but Let's Authenticate adds stronger unlinkability of accounts and ensures consent for logins. Let's Authenticate has the advantage of being more scalable for users, more efficient to use, and providing centralized account recovery. For these properties, FIDO2 requires a separate registration of each token with each website, a user may need to juggle multiple tokens across various websites, and loss of a token is catastrophic unless the user has the foresight to register a backup token. Both FIDO2 and Let's Authenticate have deployability challenges relative to passwords, though FIDO2 likely can overcome these more easily due to strong industry backing.

VIII. DISCUSSION

We discuss several important issues resulting from our development of Let's Authenticate.

A. Reflections on System Design

Designing an authentication system that uses certificates but is both privacy-preserving and avoids trusting a third party is challenging. Our design went through numerous iterations. The result can be viewed as a set of components, each providing essential features:

- Authentication uses a combination of account certificates and session certificates. Session certificates are vital because keeping $S_{session}$ secret eliminates having a trusted third party and ensures that an attacker who compromises the CA cannot impersonate any users at relying parties. One might consider using *only* session certificates that are self-signed, treating the authenticator as essentially a password manager for certificates. However, using account certificates enables a user to revoke access from a lost or stolen authenticator.
- Account identifiers are unique and randomly generated for each relying party. Likewise, account and session keys are unique to each relying party. The mapping between domains and account identifiers is kept secret. Together, these features ensure accounts cannot be linked and no identifying information is given to relying parties.
- The encrypted vault of authentication data enables a user to operate multiple authenticators for the same set of accounts and also enables remote logout.
- The FIDO2 hardware token enables users to prove ownership of an account with a CA and thus deauthorize authenticators that have been lost or stolen. The token also allows a user to maintain access to all of their accounts with relying parties even if they lose all of their authenticators. We allow the user to register backup hardware tokens to protect against loss of a token. We also give the user an emergency kit to protect against losing the secret key, which is needed to decrypt synchronized authentication data, and this kit should be stored in a secure location.
- Our system separates the authenticator from the client device. This is not standard practice for a password manager, but is somewhat similar to a FIDO2 token. This separation enables safe login on public computers and remote logout.

The result of this design is a unique set of trade-offs in the space of password alternatives. As compared to password managers, Let's Authenticate fills more security properties because it replaces passwords with cryptographic secrets and fills identical usability properties. As compared to passwordless FIDO2, Let's Authenticate has similar security properties as a hardware token, but has significant usability improvements. Let's Authenticate is more scalable for users; it requires registration only with the CA, after which other steps are

automated except for an approval step. Let's Authenticate also has easier recovery from loss; if a user loses an authenticator, they can recover all of their accounts in one step by authorizing a new authenticator with their hardware token and initializing the authenticator with the secret key.

One significant challenge for adoption of Let's Authenticate is helping users migrate from an existing account where they use a username and password or federated single-sign on. This would require additional mechanisms at relying parties, along with clear user interfaces that help users make the switch and avoid phishing attacks for their existing passwords.

B. Differences From Passwordless FIDO2

Let's Authenticate uses passwordless FIDO2 for users to authenticate to the CA, after which the user can authorize the use of specific authenticators. Passwordless FIDO2 could be replaced with any other login method, but hardware tokens provide strong protection for the synchronized authentication data. Our design means that users only need to use their hardware keys whenever they want to add or remove an authenticator from their account. This allows Let's Authenticate to avoid carrying and using tokens on a daily basis, a major drawback identified in previous studies [18]. Passwordless FIDO2 tokens also require significant overhead for users, who have to setup backups separately with each RP to guard against loss or theft [33]. Let's Authenticate avoids this overhead by centralizing use of the token to the CA account. Moreover, users can easily recover accounts with relying parties even if they lose all of their software authenticators, and if a user loses a hardware token, they only need to revoke it with the CA, not with every individual relying party.

One possible improvement to the Let's Authenticate design could be to use passwordless FIDO2 when authenticating to RPs in place of session certificates. The primary benefit of this modification would be minimizing changes needed for RPs, which could support both FIDO2 and Let's Authenticate at once. There are two possibilities. (1) The authenticators could store a single FIDO key pair in software, as part of the encrypted and synchronized authentication data. This would require a modification of the FIDO2 specification, which states that "it is expected that a credential private key never leaves the authenticator". However, this modified system would still need account certificates to provide for deauthorization in case an authenticator is lost, and RPs would still need to be modified to receive and validate the account certificate. (2) Each authenticator could use its own FIDO key pair, with the private key stored in hardware as recommended. The system would need to work with the RP to authorize which FIDO public keys are allowed to use an account, to prevent the CA from impersonating the user. This in turn leads to two possibilities if an authenticator is stolen. (2a) A race between the thief and the owner to see who can deauthorize the other's FIDO key first. (2b) A potentially cumbersome method where the hardware token is needed to register and revoke an authenticator's FIDO keys with the RP, ensuring the owner maintains control of the account. Note that registering FIDO2 backup keys at RPs and avoiding account compromise if a key is lost or stolen is an ongoing research problem. As future work we anticipate exploring these option further to determine if we can better leverage passwordless FIDO2 deployment and simplify RP deployment requirements.

C. Centralized Management

An important feature of the Let's Authenticate system is that it enables centralized management of a user's authentication keys, similar to a password manager. Currently, passwordless FIDO2 standards place this management load directly on the user—they have to keep track of which tokens they use with which website, and they have to remember which websites they have registered a backup for in case of loss or theft of a token. An open question is whether passwordless FIDO2 could evolve to likewise include centralized management, or whether tools can use automation to simplify registration and account recovery for users so their cognitive load is lightened.

Any centralized authentication system must help users to manage their account, which can be challenging for users. Loss of access to this account means they lose access to every account with a relying party. Currently, Let's Authenticate allows a user to register backup tokens in case they lose one. It's an open question whether users will be successful in managing their backup tokens. Future work could examine a broad set of account management strategies for centralized authentication systems and illuminate their trade-offs.

A related question is whether users should be required to re-authenticate with their hardware token periodically. The current design of Let's Authenticate requires users to use their hardware token every 10 days. One benefit of a shorter period is that the user is less likely to misplace their hardware token if they must regularly use it. In addition, this short period means that an attacker who is able to steal an authenticator *and* bypass the biometric or PIN protecting the Let's Authenticate software, only has a short window where they can use the stolen credentials. However, a clear downside is that the user must now carry the hardware token with them regularly. An alternative is to only require the hardware token when activating a new authenticator or de-authorizing a lost or stolen authenticator. This improves usability because the user no longer needs to carry their token with them regularly, but could lead to users misplacing tokens that they don't use often and also gives an attacker a larger window of opportunity. We note that the owner of the account can de-authorize any authenticator using a hardware token registered to their account, which reduces harm for lost authenticators and may tip the balance toward preferring the greater usability of only using the hardware token when authorizing or deauthorizing authenticators. These trade-offs need future study and input from users.

Finally, another open question is helping users cope with trust when dealing with a centralized system. Users naturally worry that a breach will result in an attacker gaining access to all of their accounts at once. Users are also justifiably skeptical of claims that a cloud service is secure, given news accounts of breaches, their lack of expertise, and the overall difficulty of ensuring that a system has no design flaws or an implementation has no bugs. This is one advantage of using hardware tokens alone (without centralized management), since this doesn't require trusting a cloud service provider.

IX. Conclusion

This work explores the creation of the Let's Authenticate system for certificate-based user authentication. We used a survey of user preferences to shape our design of Let's Authenticate. Our design demonstrates how a certificate-based authentication system can still preserve user privacy and minimize the amount of trust needed, so that a compromised certificate authority cannot impersonate users. We use a security analysis to show how the system prevents a variety of threats, and we use an overhead analysis to demonstrate the scalability of the system. We also compare Let's Authenticate to several notable systems using a well-known framework to illustrate its advantages.

The bar for any system aiming to replace passwords is high. Overall Let's Authenticate is a promising system for replacing passwords, but further work is needed. First, both short and long term usability studies are needed to refine the usability of the system and assess user attitudes toward adoption. Second, users need help understanding the security benefits of strong encryption [68], and to develop a mental model of how the authentication system works [33]. Finally, deployment should be made as simple as possible for developers.

Acknowledgments

We would like to express our gratitude to Ammon Mugimu and Edward Smart for their contributions in the early concept design stages. We would also like to thank the anonymous reviewers for their helpful feedback and guidance. This research is supported in part by the National Science Foundation under Grant No. CNS-1816929.

References

[1] 1Password, "1password security design."

[2] P. S. Aleksic and A. K. Katsaggelos, "Audio-visual biometrics," *Proceedings of the IEEE*, vol. 94, no. 11, pp. 2025–2044, 2006.

[3] F. Alliance, "Client to authenticator protocol (CTAP)," January 2019.

[4] M. R. Asghar, M. Backes, and M. Simeonovski, "PRIMA: Privacy-preserving identity and access management at internet-scale," in *International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.

[5] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Symposium on Security and Privacy*. IEEE, 2012, pp. 553–567.

[6] J. Brainard, A. Juels, R. L. Rivest, M. Szydlo, and M. Yung, "Fourth-factor authentication: somebody you know," in *Proceedings of the 13th ACM conference on Computer and communications security*, 2006, pp. 168–178.

[7] K. H. Brown, "Security requirements for cryptographic modules," *Federal Information Processing Standards Publication, FIPS 140-2*, pp. 1–53, 1994.

[8] A. Bud, "Facing the future: The impact of apple faceid," *Biometric technology today*, vol. 2018, no. 1, pp. 5–7, 2018.

[9] P. R. Center, "Americans and cybersecurity," January 2017. [Online]. Available: https://www.pewresearch.org/internet/2017/01/26/2-password-management-and-mobile-security/

[10] S. Chiasson, E. Stobert, A. Forget, R. Biddle, and P. C. Van Oorschot, "Persuasive cued click-points: Design, implementation, and evaluation of a knowledge-based authentication mechanism," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 2, pp. 222–235, 2011.

[11] S. Chiasson, P. C. van Oorschot, and R. Biddle, "A usability study and critique of two password managers." in *USENIX Security Symposium*, vol. 15, 2006, pp. 1–16.

[12] J. S. Conners and D. Zappala, "Let's Authenticate: Automated Cryptographic Authentication for the Web with Simple Account Recovery," in *Who Are You?! Adventures in Authentication Workshop*, ser. WAY '19, Santa Clara, California, USA, Aug. 2019, pp. 1–6.

[13] J. Daugman, "How iris recognition works," in *The essential guide to image processing*. Elsevier, 2009, pp. 715–739.

[14] D. De Cock, C. Wolf, and B. Preneel, "The Belgian electronic identity card (overview)." in *Sicherheit*, vol. 77, 2006, pp. 298–301.

[15] S. Drimer, S. J. Murdoch, and R. Anderson, "Optimised to fail: Card readers for online banking," in *International Conference on Financial Cryptography and Data Security*. Springer, 2009, pp. 184–200.

[16] M. H. Eldefrawy, K. Alghathbar, and M. K. Khan, "Otp-based two-factor authentication using mobile phones," in *2011 eighth international conference on information technology: new generations*. IEEE, 2011, pp. 327–331.

[17] M. Fagan, Y. Albayram, M. M. H. Khan, and R. Buck, "An investigation into users' considerations towards using password managers," *Human-centric Computing and Information Sciences*, vol. 7, no. 1, p. 12, 2017.

[18] F. M. Farke, L. Lorenz, T. Schnitzler, P. Markert, and M. Dürmuth, ""you still use the password after all"–exploring fido2 security keys in a small company," in *Symposium on Usable Privacy and Security (SOUPS) 2020)*, 2020, pp. 19–35.

[19] D. Florencio and C. Herley, "A large-scale study of web password habits," in *International conference on World Wide Web (WWW)*. ACM, 2007, pp. 657–666.

[20] D. Florêncio and C. Herley, "One-time password access to any server without changing the server," in *International Conference on Information Security*. Springer, 2008, pp. 401–420.

[21] D. Florêncio, C. Herley, and P. C. Van Oorschot, "An administrator's guide to Internet password research," in *Large Installation System Administration Conference (LISA)*, 2014, pp. 44–61.

[22] N. Haller and C. Metz, "Rfc1938: A one-time password system," 1996.

[23] S. Hammann, R. Sasse, and D. Basin, "Privacy-preserving openid connect," in *Asia Conference on Computer and Communications Security (ASIACCS)*. ACM, 2020, pp. 277–289.

[24] M. Hanson, D. Mills, and B. Adida, "Federated browser-based identity using email addresses," in *W3C Workshop on Identity in the Browser*, 2011.

[25] N. J. Hopper and M. Blum, "Secure human identification protocols," in *International conference on the theory and application of cryptology and information security*. Springer, 2001, pp. 52–66.

[26] Ironkey, 2021, https://www.ironkey.com/en-US/.

[27] M. Jakobsson, L. Yang, and S. Wetzel, "Quantifying the security of preference-based authentication," in *Proceedings of the 4th ACM workshop on Digital identity management*, 2008, pp. 61–70.

[28] D. P. Kormann and A. D. Rubin, "Risks of the passport single signon protocol," *Computer networks*, vol. 33, no. 1-6, pp. 51–58, 2000.

[29] M. Kuhn, "Otpw—a one-time password login package," 1998.

[30] L. Lamport, "Password authentication with insecure communication," *Communications of the ACM*, vol. 24, no. 11, pp. 770–772, 1981.

[31] LastPass, "LastPass technical white paper."

[32] S. G. Lyastani, M. Schilling, S. Fahl, M. Backes, and S. Bugiel, "Better managed than memorized? Studying the impact of managers on password strength and reuse," in *USENIX Security Symposium*, 2018, pp. 203–220.

[33] S. G. Lyastani, M. Schilling, M. Neumayr, M. Backes, and S. Bugiel, "Is fido2 the kingslayer of user authentication? a comparative usability study of fido2 passwordless authentication," in *Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 268–285.

[34] M. Mannan and P. C. Van Oorschot, "Using a personal device to strengthen password authentication from an untrusted computer," in *International Conference on Financial Cryptography and Data Security*. Springer, 2007, pp. 88–103.

[35] T. Martens, "Electronic identity management in estonia between market and state governance," *Identity in the Information Society*, vol. 3, no. 1, pp. 213–233, 2010.

[36] Mozilla, 2021, https://www.mozilla.org/en-US/.

[37] B. C. Neuman and T. Ts'o, "Kerberos: An authentication service for computer networks," *IEEE Communications magazine*, vol. 32, no. 9, pp. 33–38, 1994.

[38] S. Oesch and S. Ruoti, "That was then, this is now: A security evaluation of password generation, storage, and autofill in browser-based password managers," in *USENIX Security Symposium*, 2020.

[39] M. O'Neill, "The security layer," Ph.D. dissertation, Brigham Young University, 2019.

[40] OneSpan, 2021, https://www.onespan.com/products/transaction-signing/cronto.

[41] K. Owens, O. Anise, A. Krauss, and B. Ur, "User perceptions of the usability and security of smartphones as {FIDO2} roaming authenticators," in *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, 2021, pp. 57–76.

[42] B. Parno, C. Kuo, and A. Perrig, "Phoolproof phishing prevention," in *International conference on financial cryptography and data security*. Springer, 2006, pp. 1–19.

[43] A. Parsovs, "Practical issues with tls client certificate authentication." in *Network and Distributed System Security Symposium (NDSS)*, vol. 14, 2014, pp. 23–26.

[44] A. Pashalidis and C. J. Mitchell, "Impostor: A single sign-on system for use from untrusted devices," in *IEEE Global Telecommunications Conference, 2004. GLOBECOM'04.*, vol. 4. IEEE, 2004, pp. 2191–2195.

[45] PassWindow, 2021, https://passwindow.com/.

[46] S. Pearman, S. A. Zhang, L. Bauer, and N. Christin, "Why people (don't) use password managers effectively," in *Symposium on Usable Privacy and Security (SOUPS)*, 2019.

[47] R. Peeters, J. Hermans, P. Maene, K. Grenman, K. Halunen, and J. Häikiö, "n-auth: Mobile authentication done right," in *Computer Security Applications Conference (ACSAC)*. ACM, 2017, pp. 1–15.

[48] D. Recordon and D. Reed, "Openid 2.0: a platform for user-centric identity management," in *Proceedings of the second ACM workshop on Digital identity management*, 2006, pp. 11–16.

[49] K. Reese, T. Smith, J. Dutson, J. Armknecht, J. Cameron, and K. Seamons, "A usability study of five two-factor authentication methods," in *Symposium on Usable Privacy and Security (SOUPS) 2019)*, 2019.

[50] J. Reynolds, N. Samarin, J. Barnes, T. Judd, J. Mason, M. Bailey, and S. Egelman, "Empirical measurement of systemic 2fa usability," in *USENIX Security Symposium*, 2020, pp. 127–143.

[51] J. Reynolds, T. Smith, K. Reese, L. Dickinson, S. Ruoti, and K. Seamons, "A tale of two studies: The best and worst of yubikey usability," in *Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 872–888.

[52] R. L. Rivest, "Reflections on SDSI," given at LampsonFest (Microsoft, Cambridge, MA) in celebration of Butler Lampson's birthday.

[53] R. L. Rivest and B. Lampson, "SDSI – a simple distributed security infrastructure."

[54] A. Ross, J. Shah, and A. K. Jain, "From template to image: Reconstructing fingerprints from minutiae points," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 4, pp. 544–560, 2007.

[55] RSA, "RSA secureID two-factor authentication," 2021, https://www.securid.com/en-us/index.

[56] S. Schechter, A. B. Brush, and S. Egelman, "It's no secret. measuring the security and reliability of authentication via "secret" questions," in *2009 30th IEEE Symposium on Security and Privacy*. IEEE, 2009, pp. 375–390.

[57] M. A. Sirbu and J.-I. Chuang, "Distributed authentication in kerberos using public key cryptography," in *Proceedings of SNDSS'97: Internet Society 1997 Symposium on Network and Distributed System Security*. IEEE, 1997, pp. 134–141.

[58] A. Smith, "What the public knows about cybersecurity," Pew Research Center, 2017, https://www.pewinternet.org/2017/03/22/what-the-public-knows-about-cybersecurity/.

[59] S. L. Smith, "Authenticating users by word association," *Computers & Security*, vol. 6, no. 6, pp. 464–470, 1987.

[60] Sovrin, 2021, https://sovrin.org/.

[61] F. Stajano, "Pico: No more passwords!" in *International Workshop on Security Protocols*. Springer, 2011, pp. 49–81.

[62] P. Svenda, M. Nemec, P. Sekan, R. Kvasnovsky, D. Formanek, D. Komarek, and V. Matyas, "The million-key question—investigating the origins of RSA public keys," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 893–910. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/svenda

[63] H. Tao, "Pass-go, a new graphical password scheme," Ph.D. dissertation, University of Ottawa (Canada), 2006.

[64] T. W. Van Der Horst and K. E. Seamons, "Simple authentication for the web," in *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops-SecureComm 2007*. IEEE, 2007, pp. 473–482.

[65] W3C, "Web authentication: An API for accessing public key credentials," March 2019, https://www.w3.org/TR/webauthn/cross-platform-attachment.

[66] D. Weinshall, "Cognitive authentication schemes safe against spyware," in *2006 IEEE Symposium on Security and Privacy (S&P'06)*. IEEE, 2006, pp. 6–pp.

[67] A. Wiesmaier, M. Fischer, M. Lippert, and J. Buchmann, "Outflanking and securely using the pin/tan-system," *arXiv preprint cs/0410025*, 2004.

[68] J. Wu and D. Zappala, "When is a tree really a truck? exploring mental models of encryption," in *Symposium on Usable Privacy and Security (SOUPS)*, 2018, pp. 395–409.

[69] J. Yan, A. Blackwell, R. Anderson, and A. Grant, "Password memorability and security: Empirical results," *Security & Privacy*, vol. 2, no. 5, pp. 25–31, 2004.

[70] Yubico, 2021, https://www.yubico.com/.

[71] B. Zhu, X. Fan, and G. Gong, "Loxin—A solution to password-less universal login," in *Conference on Computer Communications Workshops*. IEEE, 2014, pp. 488–493.

APPENDIX

A. System Ratings

We include here ratings for Let's Authenticate, Mozilla Persona, and passwordless FIDO2 (1FA) tokens.

Let's Authenticate:

1) Usability: The system is *Quasi-Memorywise-Effortless* due to its use of a FIDO2 hardware token for CA account authentication, when the token requires entry of a PIN. It is *Scalable-for-Users* because most registration and logins require only approval. The system does not support *Nothing-to-Carry* because a user will need to carry a hardware token with them. See §VIII-C for a discussion of how to overcome this. It is *Physically-Effortless* due to needing to enter the master password periodically. Because the user interface combines elements of FIDO2 with password managers, and studies show these are easy to learn, we anticipate our system is also *Easy-to-Learn*, though more work is needed to ascertain this. We judge our system *Efficient-to-Use* due to its centralized management, and there are *Infrequent-Errors* due to automation of all cryptographic operations. It has *Quasi-Easy-Recovery-from-Loss* because a user needs to manage backup hardware tokens only for their CA account, as opposed to for every relying party, and must keep track of a printed secret key to initialize new authenticators.

2) Deployability: The system may or may not be *Accessible* depending on the type of hardware token, which depends on its form, PIN entry, etc. Screen readers can help with additional interactions (*e.g.,* approving a login, revoking a lost authenticator). The system does not have *Negligible-Cost-per-User* due to cost of a hardware token and the per-user cost incurred by a certificate authority. It is not *Server-Compatible*,

and it is *Browser-Compatible* because a user may always use a smartphone authenticator and we have demonstrated proof-of-concept browser extensions for most major browsers. The system is not *Mature* but is *Non-Proprietary*.

3) Security: The system is *Resilient-to-Physical-Observation*, because no one is able to observe a user entering a secret due to its use of hardware tokens. It is *Resilient-to-Targeted-Impersonation*, *Resilient-to-Throttled-Guessing*, and *Resilient-to-Unthrottled-Guessing* because we use a hardware token when authenticating to the CA, and also use cryptographic secrets when authenticating with websites. The only avenue for a targeted impersonation is if someone has physical access to an authenticator *and* is able to bypass the PIN or biometric on the authenticator software *and* has access to a printed recovery kit *and* has access to a hardware token registered to the account. It is *Resilient-to-Internal-Observation*, due to its use of a hardware token. It is *Resilient-to-Leaks-from-Other-Verifiers*, since relying parties only validate certificates. It is *Resilient-to-Phishing*, since website logins cannot be phished and account with the CA is secured with a hardware token. It is *Resilient-to-Theft* depending on the method used for encrypting authenticator data. If the system uses a secret key that is printed for a recovery kit, then it has full protection against theft of a hardware token since the recovery kit is also needed to initialize a new authenticator. If the system uses a FIDO2 hardware token to generate the secret used to encrypt the authenticator data, then its resilience to theft depends on whether a PIN is required for the hardware token. It has *No-Trusted-Third-Party* because a malicious or compromised CA cannot gain access to a user's accounts. Logins satisfy *Requiring-Explicit-Consent* and accounts are *Unlinkable* due to use of a unique identifier and public keys for each service.

Persona:

4) Usability: Persona is *Quasi-Memorywise-Effortless* because a user must remember a password for their email account. It is *Scalable-for-Users*, like other federated systems. Because a user only needs to verify ownership of their email account, there is *Nothing-to-Carry*, it is *Physically-Effortless*, *Easy-to-Learn*, and *Efficient-to-Use*. Since there is only one password to type, it should have *Infrequent-Errors* and the system should offer easy*Recovery-From-Loss* since password reset is commonly deployed for email accounts.

5) Deployability: Persona is *Accessibile* because the user interaction is typing a password. Likewise, there is *Negligible-Cost-per-User* for identity providers and websites. Persona does require changes to servers, so it is not *Server-Compatible*, but it does have a cross-platform JavaScript Library, enabling it to be *Browser-Compatible*. It was *Mature* during its heyday, due to Mozilla's support and is open source or *Non-Proprietary*.

6) Security: Persona has *Quasi-* support for four properties—*Resilient-to-Physical-Observation*, *Resilient-to-Targeted-Impersonation*, *Resilient-to-Throttled-Guessing*, and *Resilient-to-Unthrottled-Guessing*. These properties all hinge on the fact that passwords are no longer used with websites, so these attacks cannot be performed there, but may be successful against the master password. It is not *Resilient-to-Internal-Observation*, due to its use of an email password. It is *Resilient-to-Leaks-from-Other-Verifiers*, since relying parties only validate certificates. It is not *Resilient-to-Phishing*, since email authentication can be spoofed, but is *Resilient-to-Theft* like other password-based services. It cannot satisfy *No-Trusted-Third-Party* due to reliance on identity providers, but logins do offer *Requiring-Explicit-Consent*. Accounts are not *Unlinkable* due to use of certificates certifying email addresses, unless users go to the significant cost of establishing a separate email address for each account.

Passwordless FIDO2:

1) Usability: FIDO2 hardware/software tokens, when used in passwordless mode (1FA) are *Quasi-Memorywise-Effortless* when the token requires entry of a PIN. They are not *Scalable-for-Users* because the user must separately register a token (and a backup if desired) at each website. Whether there is *Nothing-to-Carry* depends on the type of token. Based on recent studies [33], [18] they are *Physically-Effortless* and *Easy-to-Learn*. They are *Quasi-Efficient-to-Use* and subject to *Quasi-Infrequent-Errors* because a user may need to juggle several different authenticators and remember which goes with which website, depending on website support. They do not support *Easy-Recovery-from-Loss*, since a user must manage backup tokens separately for each relying party on their own.

2) Deployability: A token may or may not be *Accessible* or have *Negligible-Cost-Per-User*, depending on its form, PIN entry, etc. They are not *Server-Compatible* since changes to the back end are required, but are *Browser-Compatible* due to increasing support for the required APIs in browsers. The system is not yet *Mature*, though support is growing, and it is *Non-Proprietary* because it is based on an open standard.

3) Security: Tokens use cryptographic credentials for login, so they are *Resilient-to-Physical-Observation*, *Resilient-to-Targeted-Impersonation*, *Resilient-to-Throttled-Guessing*, *Resilient-to-Unthrottled-Guessing*, *Resilient-to-Internal-Observation*, *Resilient-to-Leaks-from-Other-Verifiers*, and *Resilient-to-Phishing*. They may be *Quasi-Resilient-to-Theft*, depending on the form of the token, and there is *No-Trusted-Third-Party*. Logins with the scheme may be *Requiring-Explicit-Consent*, depending on whether the website asks for it. Whether the system is *Unlinkable* depends on how manufacturers use attestation, because each token may have a unique attestation key. The WebAuthn spec advises manufacturers of authenticators to assign non-unique attestation keys in batches of 100,000, but there is no way to enforce this and a manufacturer may simply be careless.

Note that we disagree with the ratings for passwordless tokens given in [33]. They rate FIDO2 as *Scalable-for-Users*, but overlook the difficulty of having to setup a token separately for each website where it is used. They rate it as *Efficient-to-Use* and having *Infrequent-Errors*, but overlook the difficulty of juggling authenticators among websites. They rate it as *Browser-Compatible*, but not all browsers have the necessary protocols built in for communication with hardware and software authenticators, and we judge it not to be *Mature* yet due to these difficulties and lack of adoption by websites. They also overlook the possibility of privacy leaks that lead us to rate it as only potentially *Unlinkable*.

B. Survey

In this survey, we are interested in learning your opinions about ways to login to websites. New systems are being developed to either replace passwords or improve how they work.

Password Replacement Perceptions:

1) How important is it to you that websites adopt new systems that replace passwords? *(Likert, Extremely Important to Not at all Important)*
   (Q2–Q10 randomized, Likert, Like a Great Deal to Dislike a Great Deal)
2) Some new systems store all of your website credentials for you so that you no longer have to memorize passwords. How much do you like or dislike this feature?
3) Some new systems replace passwords with cryptography that is so strong it would take all the computing power in the world thousands of years to break your credentials. How much do you like or dislike this feature?
4) Some new systems require you to register an account only once, instead of having to register separately with each website you visit. How much do you like or dislike this feature?
5) A new system might require you to use your smartphone whenever you need to login to a website. How much do you like or dislike this feature?
6) A new system might require you to use a small hardware token, similar in size to a small USB drive, whenever you need to login to a website. How much do you like or dislike this feature?
7) A new system might require you to use a browser extension or a separate program on your device whenever you need to login to a website. How much do you like or dislike this feature?
8) A new system might require you to have a single secret, but if you lose or forget that secret, you lose access to all of your website accounts, unless you previously registered a backup device with the system. How much do you like or dislike this feature?
9) A new system might make it easy for you to recover your accounts if you trust a company to hold some of your secrets, but if that company was malicious it could also login to your accounts. How much do you like or dislike this feature?
10) A new system might require you to register a backup device with every website you use in order to provide easy recovery if you lose your primary device. How much do you like or dislike this feature?

Demographics:

11) To which gender identity do you most identify?
    - ○ Male
    - ○ Female
    - ○ Transgender Male
    - ○ Transgender Female
    - ○ Gender variant / nonconforming
    - ○ Other
    - ○ Prefer not to answer
12) What is your age?
    - ○ Under 18
    - ○ 18–24
    - ○ 25–34
    - ○ 35–44
    - ○ 45–54
    - ○ 55-64
    - ○ 65–74
    - ○ 75–84
    - ○ 85 or older
    - ○ Prefer not to answer
13) What is the highest level of education you have attained?
    - ○ o Less than high school
    - ○ High school graduate
    - ○ Some college
    - ○ 2 year degree
    - ○ 4 year degree
    - ○ Professional degree
    - ○ Doctorate
    - ○ Prefer not to answer
14) On a scale of 1 to 5, how would you rate your current technological expertise?
    For the purposes of this survey, we're primarily concerned with your computer and web-based skills. We've defined three points on the scale as follows. These tasks represent some of the things a person at each level might do.
    Beginner (1 to 2): Able to use a mouse and keyboard, create a simple document, send and receive e-mail, and/or access web pages
    Intermediate (3): Able to format documents using styles or templates, use spreadsheets for custom calculations and charts, and/or use graphics/web publishing
    Expert (4 to 5): Able to use macros in programs to speed tasks, configure operating system features, create a program using a programming language, and/or develop a database.