

# Building and Testing a Hidden-Password Online Password Manager

Mohammed Jubur<sup>1</sup>, Christopher Robert Price<sup>2</sup>, Maliheh Shirvanian, Nitesh Saxena<sup>3</sup>, *Senior Member, IEEE*, Stanislaw Jarecki<sup>4</sup>, and Hugo Krawczyk<sup>5</sup>

**Abstract**—The most commonly adopted password management technique is to store web account passwords on a password manager and lock them using a master password. However, current online password managers do not hide the account passwords or the master password from the password manager itself, which highlights their real-world vulnerability and lack of user confidence in the face of malicious insiders and outsiders that compromise the password management service especially given its online nature. We attempt to address this crucial vulnerability in the design of online password managers by proposing a cloud-based password manager that does not learn or store master passwords and account passwords. We introduce the protocol design and report on a full implementation of the system. Our implementation provides several security features, including enforcement of a unique and secure password per each service, robustness to online password guessing attacks against the password manager and the web service, robustness to password dictionary attacks upon compromise of the password manager and the web service, and security against phishing attacks. Furthermore, to assess users' perceptions of the security and usability of our password manager, we conducted a lab-based study. The findings from the study suggest that our system is close to being practical for everyday use and is viewed by users as both usable and more secure/trustworthy.

**Index Terms**—Password management, store-less password manager, hidden-password online password manager (HIPPO), security, privacy, usability study, user perception, online authentication, phishing-resistant design, device-enhanced password authenticated key exchange (DE-PAKE)/oblivious pseudo random function (OPRF) protocol, human-centred cybersecurity.

## I. INTRODUCTION

DECADES after the introduction of passwords, password-protected systems still experience various attacks, including shoulder surfing [1], online guessing (brute force) attacks

[2], and offline dictionary attacks [3]. Several of these attacks depend solely on the entropy of the passwords itself. Users tend to pick memorable passwords; however, such easier passwords generally imply easier attacks, while secure and random passwords open up other avenues of attacks due to the memorability issue (e.g., writing down, storing electronically, and less frequent updates). Moreover, users often tend to reuse the same password to access multiple web services, which increases security risks since the compromise of any one service also compromises the accounts with other services. Many of the recent data leakage attacks on popular web services either stem from the poor choice of passwords or result from the attacks on other services due to the reuse of passwords (e.g., [4], [5], [6], [7], [8]).

To address these issues, online password managers save the users' (encrypted) web service *account passwords* on a password file on a cloud service [9], [10], [11]. These password managers help to decrease the possibility of password breaches by suggesting complex and unique account passwords for each web service. However, current online password managers do not protect the “account passwords” or the “master password” from the password manager itself, which exposes them to malicious insider as well as outsider attacks that compromise the password management service—especially given its online nature. Reports show several instances of the exploitation of this crucial vulnerability in password managers [12], [13], [14], [15], [16].

To address this fundamental vulnerability, we introduce a secure cloud-based password manager built on top of the Device-Enhanced Password Authenticated Key Exchange (DE-PAKE) primitive [17]. Our proposed system does not store the account passwords on the password manager; instead, it generates a secure, randomized account password for each website upon receiving the master password from the user, thus implicitly enforcing high-entropy account passwords.

Recent studies on password guessing underscore the criticality of high-entropy passwords. [18] provides a rigorous analysis of guessing curves for probabilistic password models, while [19] demonstrates how machine-learning approaches, such as random forests, can considerably speed up guessing attacks. Researchers have also found that real-world passwords often follow distributions approximated by Zipf's law [20], indicating that users frequently choose popular (thus vulnerable) passwords. These findings strongly reinforce the need for a password manager that generates robust, unpredictable account passwords without storing them in a vulnerable location. In summary, our approach addresses these vulner-

Received 17 June 2024; revised 5 January 2025, 8 March 2025, and 5 May 2025; accepted 18 June 2025. Date of publication 26 June 2025; date of current version 23 July 2025. This work was supported in part by NSF under Grant OAC-2139358 and Grant CNS-2201465. The associate editor coordinating the review of this article and approving it for publication was Dr. Kaiping Xue. (*Corresponding author: Mohammed Jubur*.)

This work involved human subjects or animals in its research. Approval of all ethical and experimental procedures and protocols was granted by the Institutional Review Board (IRB).

Mohammed Jubur is with the College of Engineering and Computer Science, Jazan University, Jazan 82817, Saudi Arabia (e-mail: mjabour@jazanu.edu.sa).

Christopher Robert Price and Stanislaw Jarecki are with the Department of Computer Science, University of California at Irvine, Irvine, CA 92697 USA (e-mail: crprice@uci.edu; sjarecki@uci.edu).

Maliheh Shirvanian is with Netflix Inc., Los Gatos, CA 95032 USA (e-mail: maliheh21@gmail.com).

Nitesh Saxena is with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: nsaxena@tamu.edu).

Hugo Krawczyk is with Amazon Web Services, New York, NY 10036 USA (e-mail: hugokraw@gmail.com).

Digital Object Identifier 10.1109/TIFS.2025.3583459

abilities by offering a secure, non-storing, and user-friendly password manager. **Our main contributions are:**

- 1. *A Hidden-Password Online Password Manager (HIPPO)*: We introduce our password manager, built on top of the DE-PAKE cryptographic primitive. Our solution does not store or learn the master password and/or the web account passwords; rather, it computes a unique secure account password for each service as a keyed function of the master password. The high-entropy account password is registered with each service and reconstructed during authentication using an OPRF password hardening approach that receives the master password from the user and the key from the password manager server.
- 2. *Design and Implementation*: We designed our password manager as a cloud service that responds to password inquiries from users. Our implementation consists of a server-side Node.js service and a client-side Chrome browser extension, enabling secure password generation. The account password is reconstructed on the client and automatically entered in the password field, offering resistance to phishing attacks (via domain binding), master password guessing attacks (via throttling), privacy (by hiding account IDs from the manager), and easy account password updates (changing key generation parameters without changing the master password). Empirical tests confirm an overall execution plus communication time below 400ms.
- 3. *Usability Evaluation*: To measure users' perceptions of the security and usability of our proposed password manager, we conducted a lab-based study. Given prior research indicating users' discomfort with online password managers due to potential password leaks, these studies were crucial. While we are confident our solution technically addresses this concern, we aimed to understand if it also alleviates users' apprehensions. In our study, we evaluated the authentication using our password manager and compared it with LastPass. The findings revealed that participants felt more secure using our password manager than LastPass and had fewer privacy concerns. This suggests that our solution not only addresses several security concerns associated with password leaks in other online password managers but also enhances the overall usability of password authentication. We found that the usability of our approach closely mirrors that of other (potentially less secure) online password managers, as users simply need to input a username and master password to log into the web service.

*Extension to Earlier Submission*: We initially introduced our password manager in a poster paper accepted at the 2021 ACM Symposium on Applied Computing Computer Security Track (SEC@SAC21). That paper discussed the motivation, introduced the concept, and presented a proof-of-concept implementation. In this extended submission, we delve deeper into the idea, refine the design and implementation, and present two studies evaluating our system's usability.

## II. BACKGROUND

### A. Device-Enhanced Password Authenticated Key Exchange (DE-PAKE)

Our design is built on top of the Device-Enhanced Password Authenticated Key Exchange (DE-PAKE) protocol introduced in [17]. DE-PAKE is an extension of Password Authenticated Key Exchange (PAKE) with four parties: user  $U$ , client  $C$ , server  $S$ , and a device  $D$ . DE-PAKE securely transforms a user-memorable password  $\text{pwd}$  into a high-entropy random string  $\text{rwd}$  by leveraging the device, and then uses this random string as a password input to any PAKE. The authors of [17] studied the composition of their password-to-random (PTR) protocol with any PAKE protocol, giving rise to DE-PAKE that is resistant to online guessing attacks and offline dictionary attacks (under server and device compromise), without needing secure communication between the device and the client.

### B. Related Work on Password Managers

Password managers are usually available as cloud services (or desktop versions with cloud synchronization) which allow users to pick randomized high-entropy account passwords without the need to memorize them. On several commercial password managers such as LastPass and Dashlane, these account passwords are stored on the password manager and are locked under a master password [9], [10] and/or a secret key that the user needs to protect (e.g., on 1Password [11]). While password managers provide fast and easy login, all the stored passwords are exposed if the password manager is compromised or if the master password/secret key is leaked. Although 1Password is not truly a cloud-based password manager because it relies on a locally stored secret managed by the user—it cannot be compromised merely by breaching the cloud service (similar to our approach). However, it also degrades usability since the user must manage a local secret (unlike our approach).

Hash-based password managers (e.g., [21], [22], [23]) compute the account password as a hash of the master password on the fly. While they do not store the account passwords, they remain vulnerable to offline dictionary attacks if the web service itself is compromised, because the stored password is a deterministic hash-based derivation of the master password.

### C. SPHINX Vs. Our Approach

SPHINX [24] is a device-based password manager that leverages DE-PAKE primitives similarly to our system. It transforms a human-memorable password into a random password using a trusted smartphone to perform the core cryptographic operations. While SPHINX addresses many server-related threats, it critically depends on a single device (the smartphone) for secure operation. If the phone is unavailable due to connectivity issues or a drained battery, usability suffers significantly.

In contrast, our solution is *cloud-based* and does not rely on a single, trusted client device. This design choice removes constraints such as smartphone availability and battery charge,

allowing users to seamlessly reconstruct their passwords from multiple devices. No single device is essential to the secure operations of our manager, mitigating the risk that a lost or compromised device locks the user out of all services. Additionally, our system accommodates multi-device usage by securely synchronizing client-maintained parameters (ctr and UT) across the user's authorized devices (discussed later in Section III and IV), improving overall accessibility and user experience.

#### D. Passkeys

Built on the WebAuthn standard, passkeys offer a password-less authentication method that enhances security and user experience by using public-key cryptography. Unlike traditional password managers that store and autofill user-generated passwords, passkeys eliminate the need for passwords altogether. This approach mitigates risks tied to password reuse and phishing attacks, as there are no passwords to be compromised [25]. Nonetheless, since many services do not yet fully support passkeys, a robust password manager remains very important. We envision future compatibility or integration of passkeys with our system for users desiring a hybrid approach.

#### E. LastPass as a Baseline

LastPass [10] is among the most popular password management services, supporting phones, tablets, and desktops on major operating systems and browsers. It offers both personal and business versions. Users can store credentials on the LastPass cloud, which is encrypted using AES-256. The key for encryption is derived from the user's master password and username [26]. LastPass enables password generation with various customizations (length, uppercase letters, numbers, symbols, etc.), making it convenient to create complex passwords like "T8@nW5Y2\$r9".

However, even a service as robust as LastPass remains susceptible to breaches when its infrastructure is compromised. A 2022 incident [27] reportedly exposed encrypted user data, reflecting the reality that no service can guarantee perfect security. Recent investigations also link a LastPass breach to the theft of \$4.4 million in cryptocurrency [28], underscoring the real-world risks that motivate the need for solutions like ours.

In our user study (Sections V–VI), we compare participants' perceptions of our password manager to LastPass, exploring how a non-storing, OPRF-based model might address real-world threats more effectively.

### III. OUR APPROACH

#### A. A Bird's Eye View

Our approach is an online password manager that securely responds to the password reconstruction requests without storing the user's master passwords or the reconstructed account passwords registered with the web services. The design of the protocol embraces the DE-PAKE protocol, however, it answers several challenges associated with the use of an online service as the manager.

An essential component of our password manager is the Oblivious Pseudo Random Function (OPRF) defined in [17] that maps a human-memorable master password **pwd** to a high-entropy account password **rwd**. The OPRF protocol runs between the password manager service **M** and the client terminal **C** from which the user **U** authenticates to a web service **S** to reconstruct  $\text{rwd} = F_k(x)$ , where  $k$  is provided by **M** and  $x$  is provided by **C**.

Our protocol offers other essential security and privacy features such as identifying the users by a user identifier **UD** without learning the username and domain name of the services, registering a verified user Recovery (Return) Email Address **REA**, and detecting online guessing attacks against **M** through limiting the attempt rates for a given (**UD**, **REA**).

#### B. Finer Details

##### 1) Protocol Vocabulary:

- The protocol runs between the user **U** authenticating from a client machine **C**, a password manager server **M**, and a web service **S** to which the user has registered an account.
- The protocol assumes a group  $G$  of prime number  $p$  with generator  $g$  and a security parameter  $\tau = |p|$ .
- The protocol defines two hash functions  $H$  and  $H'$ , mapping a string into elements of  $\{0, 1\}^\tau$  and  $G$ , respectively.
- A master key  $mk$  is securely stored on the password manager server **M**. This key serves as the root secret for generating the OPRF keys (i.e.,  $k$ ) used to transform the user's master password **pwd** into a randomized account password **rwd**. The master key itself is never exposed to the client or any external party, ensuring the integrity of the overall protocol.
- Each user account is identified using an account identifier **UD** that is computed as  $\text{UD} = \text{HMAC}(\text{key} = \text{UT}, \text{input} = (\text{uid}, \text{domain}))$ . **UT** is an optional token picked by the user and should be carried to any client. This value is considered to be 0 if it is not set.
- Each user account registers and verifies access to a user's Recovery (Return) Email address **REA**. During the initial setup, the user is prompted to confirm **REA** by responding to a verification code or link sent to that address, ensuring that only the legitimate account holder can receive notifications or fallback codes.
- Each user account keeps a counter **ctr** that can be incremented to update **rwd** without changing **pwd**.
- The protocol assumes a key generation function  $f_{mk}(\text{REA}, \text{UD})$  on the server to compute OPRF key  $k$  from a master key  $mk$  and a account identifier **UD** (e.g., an HMAC function with  $mk$  as the key and  $(\text{REA}, \text{UD})$  as the input).
- The protocol uses the OPRF function  $F$  mapping **pwd** to **rwd** and defined as  $F_k(x) = H(x, H'(x)^k)$  in which  $x$  is the input from **C** and  $k$  is the OPRF key input from **M** associated with each account. In our instantiation  $x$  is set to  $(\text{uid}, \text{domain}, \text{ctr}, \text{pwd})$ .
- The protocol uses the OPRF function  $F$  mapping **pwd** to **rwd**, defined as  $F_k(x) = H(x, H'(x)^k)$ , where  $x$  is the input from **C** and  $k$  is the OPRF key provided by **M**. In our instantiation,  $x$  is set to  $(\text{uid}, \text{domain}, \text{ctr}, \text{pwd})$ .



**Notation Clarification:** In this expression,  $H'(x)^k$  denotes exponentiating the hash output  $H'(x)$  to the power  $k$  in a prime-order cyclic group  $G$ . It *does not* mean applying the hash function  $k$  times. Exponentiation in such a group is a common practice in cryptographic protocols built on group operations.

2) *Master Password Vs. UT, and Temporary Device Usage:*

**Implementation Notes: Master Password vs. UT.** Our protocol differentiates:

- **Master Password (pwd):** A user-memorable secret typed for each new device or session.
- **User Token (UT):** An optional high-entropy secret that the user can store or transfer once to a trusted device (e.g., in a secure file or hardware enclave).

If *pwd* alone is compromised (e.g., by phishing), an attacker cannot derive the randomized account password (*rwd*) without also obtaining UT. In our design, UT never needs to be typed repeatedly, so it remains hidden from typical keylogging or phishing vectors.

**UT = 0 Scenario.** By default,  $UT = 0$  for simpler onboarding. In this case, *pwd* plus user metadata (e.g., domain, REA) drives the OPRF. Although the protocol is still protected against offline dictionary attacks, we *strongly recommend* configuring a random, non-zero UT (similar in spirit to 1Password's "Secret Key") in higher-risk environments. Without  $UT \neq 0$ , *rwd* relies chiefly on *pwd*, potentially weakening security if *pwd* is guessed or leaked.

**Temporary Usage on Untrusted Devices.** Our protocol accommodates short-term or partial usage in untrusted/public environments:

- 1) *Minimal Ephemeral State:* The client can store only non-persistent data (blinding factors, session tokens), discarding them at logout.
- 2) *User Re-Enters UT:* Because UT is known only to the user, they can simply re-enter it on the untrusted device for that session, without permanently saving it.
- 3) *Session Revocation + Rate-Limiting:* Combined with REA notifications, this allows rapid invalidation if suspicious behavior is detected.

Thus, a user may log in from a temporary device *without* permanently exposing UT. The manager enforces tight rate-limiting to prevent credential-guessing abuses in ephemeral sessions.

3) *Protocol Instantiation:* Figure 1 shows the steps taken by each party to reconstruct *rwd* and to authenticate the user.

**Step 1:** The client initiates the protocol as follows:

- **1a.** U inputs *pwd* on C. *pwd* is picked by the user and along with *uid*, *domain*, and *ctr* is considered as the C's input into the password hardening OPRF.
- **1b.** C picks and temporarily remembers a random number  $\rho$  of size  $\tau$  and blinds the C's OPRF input by computing  $\alpha = H'(uid, domain, ctr, pwd)^\rho$ . Blinding protects *pwd* from attackers listening to the C-M channel and to those compromising M.
- **1c.** C queries M by transferring UD, REA and  $\alpha$  to the manager. UD privately identifies the user account (i.e., while hiding the username and the service domain name

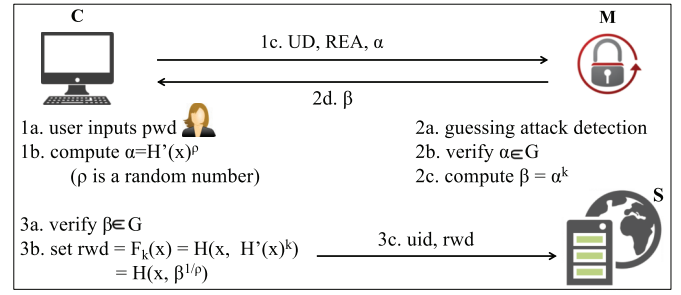


Fig. 1. The protocol reconstructs an account password to authenticate user.

from M to protect the user's privacy). REA is used to send notification in case M detects a suspicious guessing activity (through the rate limiting approach). Since REA is used in rate limiting it is necessary in every OPRF invocation.

**Step 2:** The manager continues the protocol upon receiving the query following the steps discussed next.

- **2a.** M verifies the validity of the connection based on the rate limiting policy and aborts if any anomaly is detected. If no record is found for UD (which happens in the very first attempt) M creates a record for UD and asks the user to verify access to REA (e.g., by sending an email for the user to return a code or click a link).
- **2b.** M takes its OPRF role by verifying that  $\alpha$  is an element of  $G$  and aborts if verification fails.
- **2c.** M uses master key *mk* to generate the OPRF key  $k$  and to compute  $\beta = \alpha^k$ .
- **2d.** M responds to the client by transferring  $\beta$  to C.

**Step 3:** C brings the protocol to a conclusion upon receiving the response from M, as follows:

- **3a.** C verifies that  $\beta$  is an element of  $G$ ; aborts if not.
- **3b.** C deblinds its OPRF input by computing  $\beta^{1/\rho}$ . C discards  $\rho$  at this step and sets  $rwd = F_k(uid, domain, ctr, pwd) = H(uid, domain, ctr, pwd, H'(uid, domain, ctr, pwd)^k)$ .
- **3c.** C now submits the high-entropy password *rwd* to the target web service S. S verifies *rwd* against its stored hash and, on success, issues the usual session credential to complete user authentication.

The setup phase to register the account password *rwd* with a service and the login phase follows the same steps and differs only in the client's interaction with the server as per the service password update and login process.

### C. Threat Model and Security Properties

The formal and practical security of our work derives from the DE-PAKE formal framework and security proofs of [17]. Our password manager is an implementation of that proven scheme where the password manager M plays the role of the device in that model. Here we provide an informal account of the main security properties as they follow from the results of [17]. Please refer to [17] for details.

Because *pwd* is typed frequently, an attacker may obtain it via phishing or keylogging. However, our design also relies on a user token UT, which is never routinely typed, thus

remaining protected on trusted devices. Therefore, simply having  $\text{pwd}$  alone does not suffice to derive  $\text{rwd}$ . When  $\text{UT} = 0$  (the default for convenience), the protocol still prevents *offline* dictionary attacks unless both  $M$  and  $S$  are compromised simultaneously (as shown below). Nevertheless, we strongly recommend configuring a non-zero  $\text{UT}$  (similar to 1Password's Secret Key) for high-risk scenarios.

The security definition from [17] captures “best possible” security in the sense that the only effective attacks are those that are unavoidable, namely, exhaustive online dictionary attacks against  $S$  and/or  $M$ , with offline dictionary attacks only possible upon the compromise of *both*  $S$  and  $M$ . The model does not consider defenses against client compromise or the leakage of the user's master password. Such defenses for any password manager can be obtained via second factor authentication (cf., [29]).

The DE-PAKE model considers an active attacker  $A$  that controls all communication channels and may compromise the server  $S$  and/or manager  $M$ . Security is quantified as a function of the number of online interactions between  $A$  and  $M$  and between  $A$  and  $S$ , where the number of such interactions is denoted by  $q_M$  and  $q_S$ , respectively. The results of [17], when applied to our setting, imply the following security properties (below,  $\text{Dict}$  denotes a dictionary from which  $\text{pwd}$  is chosen).

- 1) **Security against online and offline attacks:** It is shown that the probability of  $A$  to compromise the security of user authentication and key exchange is at most  $\min(q_M, q_S)/|\text{Dict}|$ . In practical terms, this means that the best possible attack is one where the attacker guesses a value of  $\text{pwd}$ , runs the protocol with  $M$  on input  $\text{pwd}$ , and uses the resultant value  $\text{rwd}$  in an authentication session with  $S$ . Only if the latter authentication succeeds, does the attacker learn  $\text{pwd}$ . This means that to be successful  $A$  needs to perform an online attack of order  $|\text{Dict}|$  against *both*  $S$  and  $M$ , making offline attacks ineffective unless both  $M$  and  $S$  are compromised.
- 2) **Resistance to Attacks upon Compromise of  $M$ :** In case server  $M$  is compromised, [17] shows that  $A$ 's probability to break the protocol security when  $M$  is compromised is at most  $q_S/|\text{Dict}|$ . This means that the best possible (and inevitable) attack against the protocol is for  $A$  to use the compromised  $M$ 's key (an OPRF key) to compute  $\text{rwd}$  for each value of  $\text{pwd}$  in  $\text{Dict}$  and test each obtained  $\text{rwd}$  in online interaction with  $S$ . Thus, even with a compromised  $M$ , the attacker still needs online interactions with  $S$  in the order of  $|\text{Dict}|$ . Also here, offline attacks are ineffective. These properties are to be contrasted with password managers that store a list of passwords encrypted under  $\text{pwd}$  and where the compromise of the password manager typically leads to an offline-only attack on  $\text{pwd}$  and all encrypted passwords. In our case, even if  $M$  is fully controlled by the attacker, *nothing* (in the strongest information theoretic sense) is learned about  $\text{pwd}$  or on the individual  $\text{rwd}$ .
- 3) **Resistance to Attacks upon Compromise of  $S$ :** When a server  $S$  is compromised, [17] shows an upper bound of  $q_M/|\text{Dict}|$  on the attacker's probability of success. It

implies that the best attack against the protocol in this case, is an exhaustive online attack against  $M$  using guessed values  $\text{pwd}$  to obtain  $\text{rwd}$  that can then be validated against the compromised state of  $S$ . Note that an exhaustive attack on  $\text{rwd}$  could also be possible in this case; however since  $\text{rwd}$  is a high-entropy secret such attack is infeasible. Moreover, learning  $\text{rwd}$  directly from a server  $S$  (e.g., breaking the server's TLS communication), does not compromise either  $\text{pwd}$  or any other password  $\text{rwd}$  as  $\text{rwd}$ 's are random and computationally independent from each other as long as the OPRF key is not disclosed.

- 4) **Resistance to Attacks upon Compromise of  $M$  and  $S$ :** When *both*  $M$  and  $S$  are compromised, an inevitable offline attack of order  $|\text{Dict}|$  is possible. In it, one lists all possible  $\text{pwd}$  values and uses the internal (OPRF) key at  $M$  to calculate  $\text{rwd}$  for each  $\text{pwd}$ . Then  $\text{rwd}$  is tested against the compromised state at  $S$ .
- 5) **Resistance to Phishing Attacks:** Another critical security property is protection against phishing attacks, achieved by binding the domain name to the derivation of  $\text{rwd}$ . Any phished domain yields a different  $\text{rwd}$ , thereby preventing credential theft.
- 6) **Full cryptographic security with random  $\text{UT}$ :** Our design accommodates an optional value  $\text{UT}$  that can be set by the user to 0 or to a high-entropy random value. In the latter case (in which  $\text{UT}$  needs to be carried to all the user's client machines), and as long as  $\text{UT}$  is not disclosed, no feasible attack, online or offline, or upon the compromise of both  $M$  and  $S$ , exists against the master password  $\text{pwd}$  and its derivatives  $\text{rwd}$ . An attacker would need to guess both  $\text{pwd}$  and  $\text{UT}$ . Conversely, if  $\text{UT}$  is ever disclosed, our protocol still guarantees the above properties.
- 7) **Master Password Leakage and Temporary Usage:** Although the threat model does not fully consider client compromise, we note that temporary usage on untrusted devices further reduces the risk of  $\text{UT}$  exposure. If  $\text{pwd}$  is phished or guessed, the attacker still cannot compute  $\text{rwd}$  unless they also capture  $\text{UT}$ . When using temporary sessions,  $\text{UT}$  remains stored briefly (often encrypted) and can be revoked after the session. Combined with rate-limiting and REA-based notifications, this ensures that local compromise of a temporary device does not grant ongoing attacker access to the user's credentials. Hence, neither a single-factor breach ( $\text{pwd}$  alone) nor temporary device compromise compromises the user's entire password ecosystem.

#### IV. SYSTEM IMPLEMENTATION

In this section, we discuss the implementation of HIPPO, as well as the challenges related to online attacks, phishing, user privacy, and service reliability. Our security analysis in Section III-C shows the security of the system in case of an attack. We assume both client and server follow best practices to address common security threats, such as securely storing secrets. For example, secret values such as  $\text{mk}$  and  $\text{UT}$  are stored securely on the server and the client

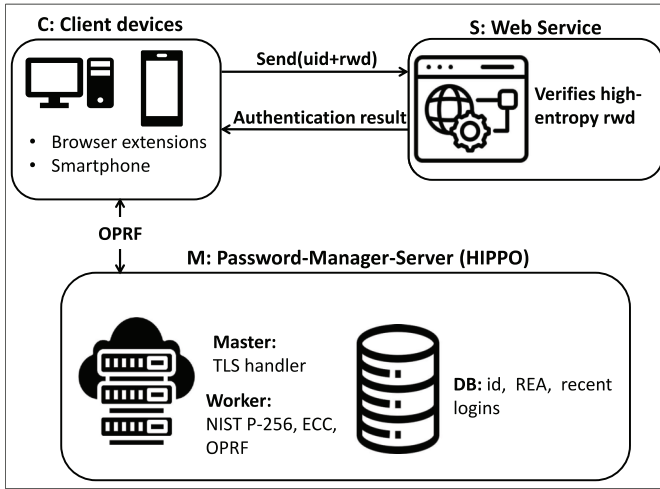


Fig. 2. HIPPO workflow. Clients derive site-specific random passwords via an OPRF with the cloud server; the derived  $rwd$  is sent to the target web service.

applications. Similarly, the server deploys denial of service attack prevention to avoid attacks on the server. Rate limiting, discussed later in this section, is one of these mitigation mechanisms.

#### A. Password Manager Client

Figure 2 shows the high-level specification. We consider NIST P-256 as the group  $G$  over which the operations run. The user registers a single recovery-email REA on the client and verifies it via a one-time code/link sent by the server. HIPPO's cryptographic core is independent of user interfaces: the current Chromium extension and a compact Android/iOS wrapper both initiate the same OPRF-over-WebSocket flow and utilize the returned  $rwd$ ; only the local trigger/GUI differs, meaning that no protocol changes are necessary.

The client runs when a website is accessed. It attempts to dynamically locate a password web form. The client waits for the special user input, either the “F2” or “@@” key sequences (in-line with [21]), and then activates a callback when the user finishes typing in the password field by relinquishing focus on that element. Upon activation of the callback, a TLS encrypted WebSocket connection with the server is attempted, with which the client negotiates a message containing (1)  $\alpha = H'(uid, domain, ctr, pwd)^\rho$  as an elliptic curve  $(x, y)$  coordinate pair, (2) the user identification UD computed using the web form username and current website domain name, and (3) the recovery email address REA. The client waits for the response from the server (i.e.,  $\beta$ ). If the received value satisfies the ECC curve equation,  $\beta$  is raised to  $1/\rho$  to unblind  $H'(uid, domain, ctr, pwd)^k$ . This value is then mapped to  $rwd$  to provide the generated password. The generated password is then populated in the form.

1) *Reading and Replacing the Password from/to Forms:* This functionality is a fork of the chrome port of Stanford's PwdHash extension [21]<sup>1</sup>. The password is received and replaced from/to website forms similar to the Stanford's PwdHash Google Chrome port, with several modifications

to adopt recent website forms. The mapping of  $H(x, H'(x)^k)$  to  $rwd$  is also based on PwdHash HashedPassword function from the same port. The extension applies password character length and constraints and populates password field with the permuted password.

To come up with the most practical mapping we reviewed 50 most visited websites [30] that require username and password. We reviewed their password policies for those that explicitly mention their policy in their website, and for those website that do not have documented their policies, we attempted signing up and creating an account and tried different password combination to realize password length and character requirements. All the studied websites have some sort of password policy in place, by defining a minimum password length, and recommending use of lower case, uppercase, numbers, and special characters. Here we summarize our analysis:

2) *Strength Metric:* Several of the websites have a form of password strength metric implemented that rejects easy to guess passwords (e.g., password123, 1qaz2wsx). A password mapping scheme can simply avoid generating passwords available in previous leaked password databases or those that contain the username.

3) *Length:* The minimum number of required characters are between 4-12: 28/50 websites have a minimum length of 8, and 13 websites had a minimum length of 6 characters. Only 1-2 websites have a minimum number of 10-12 characters. Therefore, a mapping to 8-10 characters seems to satisfy most of the websites' password length requirements.

4) *Alphanumeric:* None the studied websites prohibits users from selecting passwords that contain alphanumeric. In fact 38/50 websites, recommend/require using lower case, uppercase, and numbers. Two of the websites reject passwords containing two consecutive repeat characters. Since alphanumeric seems to be a common practice, the password mapping function must include alphanumeric passwords in the generated randomized passwords.

5) *Symbols:* 25/50 websites recommend use of symbols and no website rejects passwords containing special characters. However,  $\_$  is not accepted by one website,  $<$  and  $>$  are not accepted by another website, one of the websites only accepts  $!@\$ \% ^ \& *$  and another website only accepts  $!@\$ \% ^ \& * \sim$ . Considering these few restrictions, a mapping that includes  $!@\$ \% ^ \& *$  as special characters in the generating  $rwd$  satisfies most requirements.

Since the mapping is not the core of our password manager, we did not intend to regenerate it, however, several improvements were made to migrate PwdHash code to the current website designs. In summary, our design can work with the diverse password formatting requirements of the web services, and other engineering options such as inputting the master password on the extension are also possible in practice. Similar to other password managers, password generation can also have an option to let users select length and character requirements.

6) *WebSocket Client:* Since most of the web-services to which the user logs in (e.g., banks, social media, and email services) establish a SSL channel and serve the users through HTTPS, we incorporate “Secure” WebSocket protocol for the

<sup>1</sup>The prefix of @@ and F2 is the part of this porting.



client-manager communication. This choice is made to follow the browsers' content security policy that prevents mix content. The Chrome browser extension provides the TLS encrypted WebSocket communication with the manager to exchange messages.

7) *Elliptic Curve Cryptography (ECC) Operations*: Upon password field loss of focus, extension reads the password field data and maps it to a point on NIST P-256 curve using a hash into elliptic curve function. It then picks the random 256-bit  $\rho$  and computes  $\alpha$  encoded into  $(x, y)$  pair coordinates. Upon receiving the computed  $\beta$  from the manager it performs inverse exponentiation and maps the generated point to a hex string using a SHA-256 hash function as  $H(uid, domain, ctr, pwd, \beta^{1/\rho})$ . This hexstring is encoded to  $rwd$  by applying password constraints as mentioned earlier.

8) *Password Updates*: To update the passwords without changing  $pwd$ , the user can set a  $ctr$  on the client and increment it to change the associated  $rwd$  for a specific account. Updating  $ctr$  changes the input to the OPRF protocol, and thereby, changes the account password  $rwd$ . Therefore, even if the user keeps the master password unchanged and only modifies  $ctr$ , The password manager generates a new high-entropy account password that can be registered with the service. The use of  $ctr$  also increases the security of the system against online guessing attacks on password manager server since the attacker not only need to guess the master password but also  $ctr$  to successfully authenticate to a service. In our design,  $ctr$  is managed on the client extension, however, one can assume a design that  $ctr$  is managed by the server or a separate service or app designed for this purpose. In case the user decides to change  $pwd$ , all  $rwd$ 's originated from that  $pwd$  needs to be updated. While deploying automated update feature is out of the scope of this work, similar to the interesting feature offered by password managers such as [10], it is possible to update passwords by running password update scripts from the password manager browser extension.

9) *Extension Options Page*: The client browser extension has an options page for configuring users' preferences. For development purposes, we designed the options page so that the user can input the password manager server WebSocket listening address and port. This setting can also be hard-coded in the extension code. We also allow users to pick a recovery email and set it in the extension. The extension also has an option to show the user the randomized password  $rwd$  registered with the web-service. This option can be used if the user wishes to log in from clients that do not support our password manager (e.g., email client applications).  $ctr$  and  $UT$  are also set on this page.

10) *Synchronization Mechanism*: A key advantage of our cloud-based approach is that users often access their password manager from multiple devices (e.g., a personal laptop, a work computer, or a mobile device). To ensure consistent and secure state management of client-maintained parameters—particularly  $ctr$  and  $UT$ —our system allows these parameters to be stored, encrypted, and synchronized across trusted devices. This mechanism prevents desynchronization and provides a seamless transition: for instance, if a user increments  $ctr$  on their personal laptop to update the password

for a certain domain, the updated  $ctr$  value can be securely propagated to their work computer. Likewise, if  $UT$  is in use, it is similarly shared among the user's authorized devices, preserving privacy and security guarantees.

## B. Password Manager Server

The server is built using the NodeJS framework providing the TLS encrypted WebSocket handler for external client connections. The ECC implementation is supported by the Stanford SJCL/JSBN library. In addition to the core framework, the following functionality is provided through third party modules: the WebSocket protocol implementation, the database for storing user notification/account recovery preferences, the email protocol implementation for sending notification emails/recovery instructions to users, and GeoIP for identifying abnormalities in the user's connection patterns.

1) *Secure WebSocket Server*: Although our protocol does not require and rely on confidentiality of the client-manager channel, Secure WebSocket communication was essential as per the browser's content security policy. We acquired a key and certificate issued by InCommon RSA Server CA on our manager and run Apache v2.4.6 with SSL enabled to accept Secure WebSocket connections.

2) *Service Reliability*: To offer a higher reliability, the server runs in a master-slave configuration, where the master performs connection handling and delegates ECC operations to the slaves. The slaves act as individual computational nodes, receiving a potential  $(x, y)$  coordinate pair, testing coordinate pair on a chosen elliptical curve, generating an OPRF key, exponentiating the curve configuration with that key, and finally returning it to the master. The master receives multiple slave responses and picks the response with the highest slave commitment to avoid potentially damaged slaves. The master continuously handles external client requests and delegates ECC computation to alternating slave groups to avoid blocking the service with ECC computational load. The master validates the client requests before delegating to the slaves, and upon a threshold of invalidity, applies the rate limiting.

3) *Account Look-up*: Password manager identifies the users' account to 1) generate the OPRF key, and 2) to prevent guessing attacks by applying rate limiting policies. This identification does not have any authentication characteristics as our password manager does not need to authenticate the user per se. To uniquely identify each user account, we define a user identification parameter  $UD$  per each account. All records related to an account are stored in a database and are tagged with  $UD$ .  $UD$  is set on the client and is transferred to the password manager server as part of the initial message (step 1c in Figure 1).  $UD$  can simply be the user id with a web-service. However, since revealing the user id and the domain name may raise privacy concerns (as the manager learns this information), the password manager computes the key-ed hash of the user id and domain name as  $UD$ , such that  $UD = \text{SHA256-HMAC}(key = UT, input = (uid, domain))$ . The key to the hash function (defined as  $UT$ ) is optionally set by the user (0 if not set). If  $UT$  is set the user should populate it on other clients.

TABLE I  
THE PROTOCOL COMPONENT EXECUTION TIME WITH SJCL LIBRARY

Get Random Value	SHA256	Big Integer Addition	Big Integer Division	Big Integer Exponentiation	Big Integer Mod Exponentiation	Big Integer Mod Inverse	Curve Point Exponentiation
0.1750ms	0.4450ms	0.0050ms	0.0100ms	0.0050ms	0.0200ms	0.2500ms	26.1200ms

TABLE II  
THE OVERALL PROTOCOL EXECUTION TIME

	Password Manager Server	Password Manager Client Extension
Hash into Elliptic Curve	—	1.59 ms
Membership Check	0.22 ms	0.25 ms
Point Multiplication	26.12 ms	71.25 ms
Overall Protocol Time	389.44 (including websocket communication)	

4) *Key Generation*: The OPRF key for each account is generated as a function of a master-key  $mk$ , UD and REA. UD and REA are transferred from the client to the manager (as shown in step 1c of Figure 1) and the password manager holds  $mk$ . We compute OPRF key  $k$  as  $\text{SHA256-HMAC}(\text{key} = mk, \text{input} = (\text{UD}, \text{REA}))$ .

5) *ECC Computation*: As part of the the protocol, the manager responds to the OPRF message received from the client. The manager receives the computed  $\alpha = H'(uid, domain, ctr, pwd)^p$  encoded into a  $(x, y)$  coordinate representing a point on the NIST P-256 elliptic curve. Our implementation of Hash-into-Elliptic-Curve is in line with that suggested in [24], however, other alternatives robust to side channels would be possible [31]. The manager then checks if the received  $(x, y)$  pair is a valid point on the curve. Then it computes the value of  $\beta = \alpha^k$  and transfers  $\beta$  encoded into a  $(x, y)$  coordinate to the client.

6) *Phishing Prevention*: The client inputs the domain name of the web-service in the OPRF function. This design prevents phishing attacks by involving the domain name  $domain$  in the generation of  $rwd$ . Therefore, a phished domain name generates a different  $rwd$ .

7) *Rate Limiting*: Since our design is implemented as an online service with no need for user authentication it may be a target for an active attacker who guesses the client's OPRF input (i.e.,  $(uid, domain, ctr, pwd)$ ) to reconstruct  $rwd$  and to attempt to log in to the web-service. To avoid such attacks, we developed two standard rate limiting approaches. In the first approach, the connection count per interval is monitored and connections are closed for those that reach and exceed a threshold. That is, the password manager server validates the client connection by a configurable threshold of acceptable login attempts for an UD. A default threshold of 10 logins from an IP address in 5 seconds was defined on the server. The IP address and the time-stamp of the connection attempts for UD is recorded on the database. In the second approach, an accepted connection has the GeoIP location of that connection matched with previous GeoIP locations. If the location does not match upon a threshold an invalid attempt is detected. Once a suspicious activity is detected a notification email is sent to the user's recovery email address REA. REA is transferred from the client to the password manager server (as shown in step 1c of Figure 1) and can be cross-checked by

the server with the one stored in the database associated with UD. Other notification services such as SMS or two factor authentication are other possible choices.

8) *Database Server*: We use ArangoDB, a scalable NoSQL solution, for storing the recent login attempts (latest timestamp and IP address/GeoIP from which the user transfers the request) and the recovery email associated with a UD.

9) *Used Libraries*: Node.js v8.1.2 on CentOS 7 (dual Intel Xeon CPU E5-2687W v3 @ 3.10GHz, 3.87 GB RAM) forms the backbone. Core Node.js libraries (`crypto`, `http`, `https`, `fs`) plus 3rd-party npm modules (e.g., `WebSocket` v3.0.0, `ArangoDB` v5.6.1, `nodemailer` v4.0.1, `mmdb-reader` v1.1.0) provide connectivity, database services, email notifications, and GeoIP throttling. Elliptic-curve computations rely on Node.js `crypto`, plus Stanford's JSBN and SJCL.

### C. Protocol Performance Measurement

We first evaluate the performance of the password manager server in the execution of the primary functions of the protocol. The execution time is estimated for the server running on a virtual machine with 2 x Intel Xeon CPU E5-2687W v3 @ 3.10GHz and 3.87 GB of memory. Table I shows the execution time of different functions using SJCL library. We also evaluated the same functions with ECURVE library, however, we achieved the best performance on the most time consuming functions with SJCL<sup>2</sup>. The most costly computation is the elliptic curve multiplication, which is reasonably low: less than 70 ms on the client and less than 25 ms on the server. Next, we evaluated the execution time of hashing into an elliptic curve on the client and group membership test and elliptic curve point exponentiation on both the client and the server. The timing is shown in Table II. We noticed that the most costly computation on the server and the client is related to point multiplication (computation of  $\alpha = H'(x)^p$ ,  $\beta = \alpha^k$  and  $\beta^{1/p}$ ). This computation contributes to about 160 ms of 389.4 ms overall protocol time. The execution time of other functions such as RNG and SHA256 is negligible. Having detailed the system architecture, cryptographic protocols, and performance measurements, we next present a user study that compares our cloud-based password manager against LastPass. This study provides insights into how users perceive and interact with our approach, focusing on essential tasks (e.g., installation, configuration, password updates).

## V. USER STUDY COMPARING OUR PASSWORD MANAGER WITH LASTPASS

This section presents our comparative user study of our cloud-based password manager and LastPass. The primary

<sup>2</sup>Higher Performance libraries would improve the protocol's overall performance; however, at the time of evaluation, SJCL and ECURVE were the few JavaScript libraries offering low-level operations required by our protocol implementation.



aim is to evaluate how users perceive and interact with each manager regarding essential tasks such as installation/configuration, password updates, and login procedures. While our system supports advanced features like multi-device usage and a fallback mechanism for untrusted or temporary devices, this study focuses solely on everyday password management behaviors. Future work will expand the scope to multiple devices and advanced scenarios.

In parallel, we conducted a broader survey of major password managers (LastPass, 1Password, Bitwarden) to confirm that their fundamental usability workflows are largely identical: (1) installing an extension or app, (2) creating a master password, and (3) storing credentials in a zero-knowledge vault for autofill. For instance, Seiler-Hwang et al. [32] found that 1Password's mobile interface received a lower SUS score (52.6, rated "not acceptable") than LastPass (mid-60s, "marginal") in a lab study with 60 participants, yet both products followed the same "create master password, store credentials, autofill" routine. Meanwhile, Arias-Cabarcos et al. [33] observed that 1Password and LastPass had comparable setup overhead and master-password complexity in an enterprise context. Bitwarden, though open-source and more minimalistic, still follows the same pattern of installing an extension, setting up a master password, and auto-filling site logins [34], [35]. Hence, focusing on LastPass still captures the *key* usability tasks relevant to other managers—particularly regarding installation, password updates, and perceived security/trust in a "vault-based" approach.

#### A. Objectives and Scope

Although our password manager supports multi-device usage and incorporates a fallback/recovery mechanism (e.g., a one-time secure code) to re-establish trust on new or untrusted machines, these capabilities were not explicitly examined in this user study. Instead, the comparative study primarily focused on:

- 1) **Core Usability:** Assessing the ease of installation/configuration, password updates, and login processes for both our password manager and LastPass.
- 2) **Security Perceptions:** Exploring participants' confidence in storing and protecting their credentials, as well as their trust in each manager's overall security framework.
- 3) **Adoption Willingness:** Determining participants' likelihood of using these password managers for various personal and professional online accounts.

Future evaluations will include a thorough examination of multi-device portability and fallback features, analyzing alternative device scenarios (e.g., temporary or untrusted devices) in greater depth.

#### B. Study Design

The core idea of our study was to evaluate user perception, acceptance, and thoughts on LastPass and our password manager. The study design follows similar password manager studies [24], [36], [37]. We recruited 20 participants through word-of-mouth from diverse backgrounds. Participants used

both password managers to log into predefined accounts on Gmail, Twitter, and Dropbox, chosen for their popularity and varied login procedures.

Gmail requires username first, then password; Twitter and Dropbox require both in the same form. Password update procedures also vary: Gmail involves second-factor authentication, while Twitter and Dropbox require old and new passwords, with Twitter requiring the new password twice. After the study, participants answered questions about their experience using both quantitative and qualitative methods.

Due to Covid-19, the study was conducted remotely. Each participant spent approximately 90 minutes to complete the study. The study was approved by our University's Institutional Review Board (IRB), and standard ethical procedures were followed. Participation was voluntary, with participants informed about the study and given the option to discontinue at any time.

#### C. Remote Study Setup

We conducted our study using a Google site [38],<sup>3</sup> with four pages: a home page with a welcome message, introduction, and instructions; a pre-test questionnaire; and two pages with task instructions for the password managers. After completing tasks, participants answered survey questions about their perceptions. Quantitative and qualitative data were collected using embedded Google Forms [39], allowing participants to enter their answers on the same page as the task instructions.

#### D. Preparation

As our study was conducted remotely, we communicated with participants via emails and WhatsApp. We shared usernames and passwords for pre-created accounts and provided the study site link. Zoom [40], TeamViewer [41], and Skype [42] were used for remote screen sharing and real-time observation, ensuring accurate data collection and providing assistance when needed.

#### E. Study Protocol

Our study consists of five stages, as shown in Figure 3. The concrete steps followed in each stage are outlined below.

**Stage 1: Start:** An examiner shared the study site link, usernames & passwords, and Zoom meeting link. The examiner ensured participants received all necessary links and could share their screens.

**Stage 2: Welcome:** Participants received a brief introduction and instructions about the study.

- **Introduction:** We explained password managers' general concept, security, and usability purposes, including LastPass and our proposed password manager, through a short video (3 minutes).
- **Instructions:** We outlined the tasks: filling out the pre-test survey, using LastPass with predefined accounts, and using our proposed password manager. Participants were

<sup>3</sup><https://sites.google.com/view/pmstudy-2021/home>

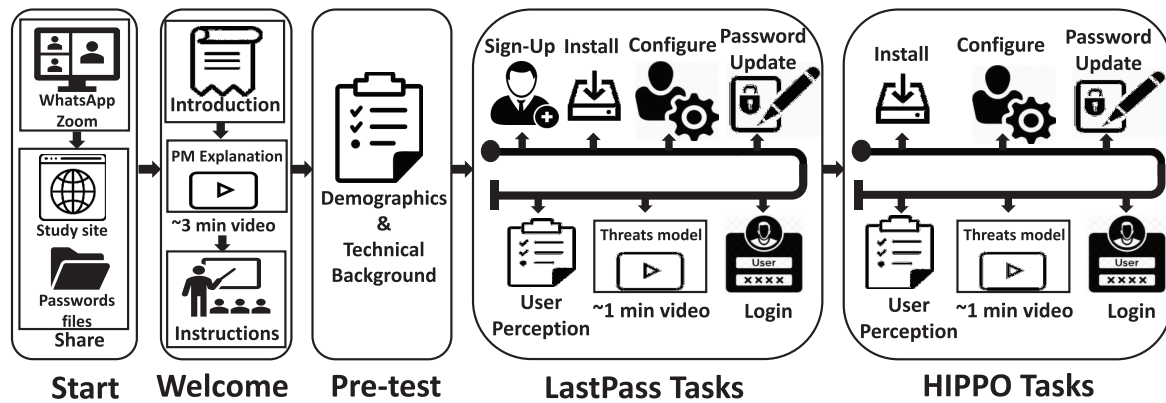


Fig. 3. Study Process.

instructed to assume they were logging into their own accounts.

**Stage 3: Pre-test:** Participants filled out a survey about their demographics and technical backgrounds.

**Stage 4: LastPass Tasks:** Participants used LastPass to complete the following tasks:

- 1) **Sign up:** Install the LastPass extension and sync with their accounts.
- 2) **Password Update:** Follow instructions to generate and update passwords using LastPass, enabling the master password reprompt feature.
- 3) **Login Using LastPass:** Log in using LastPass and repeat the login process three times for each service.
- 4) **Post-test Questionnaire:** Watch a short video about LastPass's potential threats, then complete a questionnaire.

**Stage 5: Our Password Manager Tasks:** Participants used our password manager to complete similar tasks:

- 1) **Installation and Configuration:** Install the extension, configure the manager, and set a recovery email, inputting the offset key, server address, and port number.
- 2) **Password Update:** Follow instructions to generate and update passwords using our password manager, including steps to activate and use the manager.
- 3) **Login Using Our Password Manager:** Log in using our password manager, repeating the login process three times for each service.
- 4) **Post-test Questionnaire:** Watch a short video about our password manager's potential threats, then complete a questionnaire.

#### F. Post-Test

We utilized the following measures to understand participants' perceptions of LastPass and our password manager and to answer the study questions.

**Tasks Easiness and Satisfaction Scores:** Participants rated their agreement with statements related to the easiness of tasks and their satisfaction with the authentication method on a Likert-type scale (1 = "strongly disagree" to 5 = "strongly agree"). The statements included:

leftmargin=\*

- 1) **Installation and configuration:** Ease and satisfaction with installing and configuring LastPass/our password manager.
- 2) **Password update:** Ease and satisfaction with updating passwords using LastPass/our password manager.
- 3) **Login:** Ease and satisfaction with logging into accounts using LastPass/our password manager.

**System Usability Scale (SUS):** Usability was measured using the 10-item System Usability Scale (SUS) developed by Brooke [43], with scores ranging from 0 to 100, where higher scores indicate better usability.

**System Specific Questions:** Participants rated their agreement with statements about satisfaction, trust, perceived security, and willingness to use the system on a scale from 1 to 5. Questions were grouped into two categories: system security & privacy and portability.

#### • Perception of security & privacy:

- Concern about LastPass/our password manager collecting too much information.
- Belief that passwords are safe from LastPass/our password manager itself.
- Concern about LastPass/our password manager leaking passwords to another party.
- Feeling that passwords are more secure using LastPass/our password manager.

- **Portability:** Confidence in logging in from multiple computers being convenient with LastPass/our password manager.

**Qualitative Feedback:** Participants reported any technical problems and provided open-ended feedback on their opinions, suggestions, and willingness to use LastPass/our password manager in the future.

**Statistical Analysis:** We used the Wilcoxon Signed-Rank Test (WSRT) with a 95% confidence level to measure differences in means between groups (e.g., usability, security). Bonferroni correction was applied for multiple comparisons. The effect size of WSRT was calculated as  $r = Z / \sqrt{N}$ , where  $Z$  is the z-statistic value and  $N$  is the number of observations.

## VI. USER STUDY RESULTS AND ANALYSIS

This section presents the findings from our remote user study, which compared user perceptions of usability, security, and overall acceptance between our cloud-based password manager and LastPass. Although our system supports advanced features such as multi-device usage and fallback mechanisms for untrusted devices, these capabilities were not directly evaluated in this study. Instead, we focused on common everyday tasks—installation, configuration, password updates, and logins—to gather insights into the core usability and perceived security of both password managers. Participants were recruited via social networks through word-of-mouth. We first present demographic and technical background information from the pre-test results. We then discuss user perceptions of ease of use, satisfaction, and security from the post-test questionnaire, followed by statistical analysis of the collected data.

Our broader survey suggests that mainstream password managers—1Password, Bitwarden, and LastPass—share similar workflows based on a “master password + autofill” approach. Existing literature confirms that these products mainly differ in interface design and certain advanced features, such as 1Password’s Secret Key or Bitwarden’s open-source ecosystem, rather than in fundamental usage steps [33], [44]. Munyendo et al. [44] noted that 19 of 22 LastPass users who switched to Bitwarden cited trust and simplicity concerns, despite minimal differences in actual usage patterns. Thus, our user testing, while limited specifically to LastPass versus our password manager, likely yields insights that generalize well to other mainstream password managers.

Another notable factor is that negative security incidents can overshadow otherwise user-friendly designs. The 2022 LastPass breach significantly affected users’ security perceptions and consequently their perceived ease of use, as reported by Security.org [45]. Similarly, participants in our study emphasized “trust in the product” as equally critical to usability, echoing findings from Munyendo et al. [44], where users shifted from LastPass to Bitwarden primarily for enhanced transparency and security. Hence, even though everyday usability tasks remain comparable, security controversies profoundly shape users’ perceptions and their willingness to adopt or continue using a password manager.

### A. Pre-Test Results

1) *Demographics*: The study involved 20 participants distributed across three age groups: 18–24 years (5%), 25–34 years (75%), and 35–44 years (20%). The participants were equally divided between undergraduate and graduate students, representing diverse educational backgrounds such as education, engineering, healthcare, and science. 25% were female and 75% male. Regarding education level, 30% were pursuing or had completed a Bachelor’s degree, 55% a Master’s degree, and 15% a Doctoral degree.

2) *Technical Background and Password Management Experience*: Participants rated their computer proficiency as Fair (25%), Good (45%), or Excellent (30%). Similar ratings were

observed for general computer security skills: Fair (30%), Good (45%), and Excellent (25%).

When choosing passwords, 15% preferred easy-to-remember options, 45% chose passwords difficult for others to guess, 10% selected strong randomized passwords, and 30% reused similar passwords across accounts. When creating passwords for new accounts, 45% reused existing passwords, 40% modified existing ones, 10% created entirely new passwords, and only 5% utilized a password manager-generated password. While 85% reported no known password leaks or theft, 15% had experienced compromised passwords.

Among participants, 55% had experience with password managers, with 45.5% using them daily, 45.5% occasionally, and 9.1% rarely. Password manager types were evenly used: browser-based (36.4%), device-based (36.4%), and online/cloud-based (36.4%), with overlaps between these categories.

Password managers were commonly used for email (90.9%), social media (81.8%), work (63.6%), and banking services (45.5%), often with multiple services simultaneously. Reasons cited for using password managers included convenience (54.5%), enhanced security (36.4%), and personal preference (9.1%). Interestingly, 72.7% had not used built-in password generators, while 27.3% used them, predominantly for social media (100%), email (66.7%), and work (33.3%).

### B. Post-Test Results

The post-test questionnaire results are summarized in Table III and detailed below.

1) *Usability: Easiness and Satisfaction*: Participants rated ease and satisfaction for installation/configuration, password updates, and login tasks.

#### Installation and Configuration:

- **Easiness**: LastPass: 4.15 (0.65), our password manager: 3.95 (0.97)
- **Satisfaction**: LastPass: 4 (0.77), our password manager: 3.95 (0.97)
- Slightly higher ease and satisfaction scores for LastPass reflect its streamlined installation process, though differences were not statistically significant.

#### Password Update:

- **Easiness**: LastPass: 3.85 (0.96), our password manager: 3.55 (0.97)
- **Satisfaction**: LastPass: 3.6 (0.97), our password manager: 3.85 (0.91)
- Differences were minor and statistically insignificant.

#### Password Login:

- **Ease**: LastPass: 4.25 (0.7), our password manager: 3.35 (0.91); significant difference ( $p = 0.003$ ,  $r = 0.46$ ).
- **Satisfaction**: LastPass: 3.6 (0.87), our password manager: 3.5 (0.81); no significant difference.
- LastPass was notably easier for logging in due to its simpler workflow.

**System Usability Scale (SUS)**: SUS scores were comparable (LastPass: 61.25, our password manager: 58), both indicating “OK” usability according to established benchmarks



TABLE III  
OVERVIEW OF POST-STUDY QUESTIONNAIRE SCORES

	LastPass	our password manager	Statistics	Effect Size
Easiness				
Installation and Configuration	4.15 (0.65)	3.95 (0.97)	0.377	0.46
Password update	3.85 (0.96)	3.55 (0.97)	0.15	
Password login	4.25 (0.7)	3.35 (0.91)	0.003	
Satisfaction				
Installation and Configuration	4 (0.77)	3.95 (0.97)	0.858	0.46
Password update	3.6 (0.97)	3.85 (0.91)	0.326	
Password login	3.6 (0.87)	3.5 (0.81)	0.295	
SUS	61.25 (15.07)	58 (15.36)	0.432	0.46
Security & Privacy				
Collecting too much information	3.65 (1.06)	1.75 (0.7)	0.0001	0.58
Passwords are safe from the PM itself	2.15 (0.85)	4.2 (0.75)	0.0001	0.60
Leaking password to another party	3.8 (1.08)	1.85 (0.73)	0.0001	0.58
Passwords are secure using the PM	2.85 (1.01)	4.25 (0.54)	0.001	0.55
Generating a strong randomized password	4.05 (0.74)	4.45 (0.67)	0.021	0.37
Login from multiple PC	3.45 (1.07)	3.75 (0.99)	0.217	

TABLE IV  
TECHNICAL PROBLEMS WITH THE PASSWORD MANAGERS

	LastPass	our password manager
No problems	15%	55%
Few problems	75%	40%
Some problems	10%	5%
Many problems	0%	0%

(Bangor et al. [46], Lewis et al. [47]). Differences were not statistically significant (WSRT,  $p = 0.432$ ).

2) **Security & Privacy: Privacy:** Participants significantly preferred our password manager regarding concerns about information collection (LastPass: 3.65, our password manager: 1.75;  $p < 0.0001$ ,  $r = 0.58$ ) and password leakage (LastPass: 3.8, our password manager: 1.85;  $p < 0.0001$ ,  $r = 0.58$ ).

**Security:** Participants rated our password manager significantly higher for password safety from the manager itself (our password manager: 4.2, LastPass: 2.15;  $p < 0.0001$ ,  $r = 0.60$ ), general password security (our password manager: 4.25, LastPass: 2.85;  $p = 0.001$ ,  $r = 0.55$ ), and generating strong randomized passwords (our password manager: 4.45, LastPass: 4.05;  $p = 0.021$ ,  $r = 0.37$ ).

3) **Portability:** Participants felt slightly more confident in their ability to conveniently log in from multiple computers using our password manager (3.75) compared to LastPass (3.45), though this difference was not statistically significant.

4) **Quantitative Results: Technical problems:** Participants rated technical issues encountered during the tasks on a scale from “No problems” (1) to “Many problems” (4). On average, participants faced significantly fewer technical problems with our password manager (1.5; SD = 0.59) compared to LastPass (2.1; SD = 0.3), according to the Wilcoxon Signed-Rank Test ( $p = 0.003$ ,  $r = 0.47$ ). Table IV summarizes the frequency of reported technical problems.

Many participants specifically reported compatibility issues between LastPass and Dropbox during password updates, exemplified by comments such as:

- “While using a password with required master password and I am changing the password, I find that LastPass saves the different password than what I chose to be a password which leads to reset the password again.”

- “Could not update the generated password for Dropbox automatically. I needed to update manually.”
- “Password generate/fill didn’t work in Dropbox. Had to manually copy and paste.”

Participants also experienced specific issues with our password manager, primarily related to longer response times and unfamiliar interfaces:

- “Take a longer time to compute the password.”
- “The response from the server is taking too long, and I think the program is amazing and just needs some UI enhancements to eliminate any technical issues.”
- “Sometimes our password manager had trouble starting up or connecting to the socket after pressing F2. After a couple tries, the connection was successful.”
- “Unorganized page ‘console’.”

Some connection delays with our password manager were traced to an external incident involving OVHcloud, affecting numerous online services [48].

**Disadvantages of the password managers:** Participants highlighted key disadvantages summarized in Table V. LastPass was primarily criticized for security concerns due to past breaches and vulnerabilities stemming from centralized cloud storage. In contrast, our password manager was perceived as less user-friendly due to additional complexity and cognitive effort required during use.

**Willingness to (not) use the password managers:** When asked about their willingness to adopt the password managers, 55% indicated a preference for our password manager, compared to 40% for LastPass. Participants who preferred LastPass highlighted ease of use, primarily for social media. In contrast, participants preferring our password manager cited higher perceived security, expressing willingness to use it for sensitive accounts such as banking and work-related services. Approximately 25% expressed reluctance toward using password managers altogether, citing general distrust in password manager technology.

**Open-ended Question:** At the study’s conclusion, participants provided additional comments:

- “It’s a nice experience, and useful tools will be in place.”
- “I believe that our password manager is more secure.”
- “I learned new things.”

TABLE V  
DISADVANTAGES OF THE PASSWORD MANAGERS

LastPass	Our Password Manager
Past data breaches eroded trust, prompting migration to another manager.	Not user-friendly: requires extra steps and waiting for our password manager to connect.
Cloud compromise would breach all stored secrets.	Users must understand a multi-step flow and wait for password generation.
Per-device install/config; single vault is a single point of failure.	Higher cognitive load: more effort needed to learn and use.
Prone to compromise when master vault is exposed.	(Same as above.)

- “our password manager looks promising and I would like to try it.”

### C. Defense Mechanisms Against Phishing and Online/Offline Attacks

Beyond usability metrics, password managers must address threats such as phishing, online/offline password guessing, and accidental credential leaks. Table VI summarizes how our password manager, 1Password, Bitwarden, and LastPass each tackle: noitemsep, leftmargin=\*

- 1) Phishing prevention (domain checking vs. domain binding),
- 2) Online/offline attack resilience (rate-limiting, cryptographic hardness),
- 3) Additional secrets or ephemeral usage for untrusted devices.

a) *Phishing Attacks*: Our password manager explicitly binds the derived password `rwd` to the legitimate domain, ensuring a phishing site yields mismatched credentials. 1Password prompts users to confirm domain correctness before autofilling [49], while Bitwarden enforces domain matching but can allow user overrides [35]. LastPass also checks domain matching and autofill prompts; historically it allowed broader user overrides, but it has tightened these checks after reported phishing and autofill concerns. Research suggests forced domain binding is more robust at preventing credential reuse on malicious sites [32], [33].

b) *Online Vs. Offline Password Attacks*: All four managers—our password manager, 1Password, Bitwarden, and LastPass—employ zero-knowledge encryption so that an *offline* dictionary attack typically requires compromising both the manager’s server (M) and the user’s data at S. In our password manager, the DE-PAKE-based OPRF [17] (plus UT) helps resist brute-force guessing. 1Password relies on a mandatory Secret Key in addition to a master password, Bitwarden uses high PBKDF2 (or Argon2) iteration counts [50], and LastPass employs PBKDF2 with varying iteration levels (older accounts may have lower defaults if not updated). For *online* attacks, our password manager features rate-limiting and REA alerts, 1Password and Bitwarden use 2FA/lockouts, and LastPass also supports 2FA plus lockouts after multiple failed attempts.

c) *Separation of Secrets and Untrusted Devices*: 1Password splits the master password from a Secret Key, while Bitwarden usually relies on a single master password. Our protocol likewise separates `pwd` from an UT. For ephemeral usage (e.g., on a public PC), our password manager stores only minimal data locally. When needed, the user *re-enters*

their UT—which they alone know—into the untrusted device for that session, discarding it upon logout. Bitwarden offers local caching for offline use but lacks a second secret by default, and 1Password’s Secret Key must be re-entered on each new device [45]. LastPass, for its part, has no built-in second secret beyond the master password, though 2FA is optional. Ephemeral sessions in LastPass rely on simply logging in/out without permanently storing the vault, but no additional fallback or UT-like token is enforced.

### D. Key Insights and Discussion

In our user study, participants carried out typical tasks related to password management (installation, configuration, password creation/updates, and logins), using a single device environment. Although our system supports multi-device usage and a fallback mechanism for untrusted devices, these features were not formally tested here. Consequently, the following insights primarily concern everyday, single-device workflows.

- **Usability vs. Security Trade-off**: Participants found LastPass marginally easier overall, but perceived our password manager as more secure (Tables III, VI). In particular, domain binding and the optional UT gave our password manager an edge in anti-phishing and offline-attack mitigation. Yet some users found UT slightly cumbersome, echoing the learning curve in 1Password’s Secret Key model.
- **Privacy and Trust**: Many participants expressed greater confidence in our password manager for data collection and leak risks. By design, our password manager does not store or learn master passwords, which contributed to fewer privacy concerns. This correlates with findings that last year’s breach impacted LastPass’s trust perception [45].
- **Technical Challenges**: LastPass faced more site-specific conflicts (e.g., Dropbox), whereas our password manager required a certain level of technical familiarity (browser developer tools, server connections). In the broader ecosystem, 1Password and Bitwarden also encounter occasional autofill failures, per references [32], [35]. No one solution is universally frictionless.
- **Willingness to Adopt**: A notable segment indicated they would adopt our password manager for critical accounts (banking/work), citing its unique combination of OPRF, domain binding, and optional two-secret design. They found LastPass more convenient for simple tasks (social media), aligning with user transitions from LastPass to

TABLE VI  
COMPARISON OF DEFENSE MECHANISMS IN OUR PASSWORD MANAGER, 1PASSWORD, BITWARDEN, AND LASTPASS

	Our Password Manager	1Password	Bitwarden	LastPass
<b>Phishing Defense</b>	Domain-bound rwd	User domain checks at autofill	Domain match w/ user override	Domain checks + autofill prompts <sup>a</sup>
<b>Offline Attack</b>	DE-PAKE OPRF + UT	Master pwd + Secret Key	PBKDF2/Argon2 <sup>b</sup>	Master pwd + PBKDF2 <sup>c</sup>
<b>Online Attack</b>	Rate-limiting, REA alerts	2FA, lockout after tries	2FA, iterative key stretching	2FA, lockout after multiple failed attempts
<b>Extra Secret</b>	UT	Mandatory Secret Key	Single master password	None by default; single master password
<b>Untrusted Devices</b>	User re-enters UT on new device	Must re-enter Secret Key on each new device	Local offline cache, single secret	Ephemeral usage via login/logout; no second secret

<sup>a</sup> Historically, LastPass allowed auto-fill that could be overridden; security incidents prompted improvements to domain checking [35].

<sup>b</sup> Open-source with configurable iteration counts [50].

<sup>c</sup> Older accounts might have lower iteration counts unless updated [45].

other managers due to trust or advanced security preferences [44].

- **Limitations (Sample Size and Single Device):** Our study involved 20 participants operating in a single-device setup. While these findings highlight important usability and security observations, the limited sample size and device context may affect generalizability. Future work will include larger, more diverse participant pools, formal evaluations of multi-device usage, and fallback testing for untrusted device scenarios.

**Concise Capability Summary.** Our comparative analysis (Table VI) and user study identified three distinct advantages of HIPPO (our password manager) over mainstream password managers (LastPass, Bitwarden, 1Password):

- 1) **Zero server-side storage:** No server-side storage or reconstruction capability for master or derived passwords, yielding breaches harmless in terms of credential exposure.
- 2) **Per-domain, high-entropy credentials:** Each login credential (rwd) is dynamically generated from the user's master password and optionally UT, strictly bound to the legitimate domain, effectively preventing phishing, reuse, and credential autofill attacks.
- 3) **Enhanced security without compromising usability:** The optional second secret (UT), combined with robust rate-limiting, significantly increases defense against both offline and online attacks compared to standard PBKDF2/Argon2 mechanisms. Crucially, our password manager's SUS score (58) aligns closely with commercial offerings (61), demonstrating minimal impact on user convenience.

**Cloud-Based Architecture and Fallback Mechanism.** Although we designed our password manager to support multi-device usage and a fallback/recovery feature (e.g., a one-time secure code for re-establishing trust on an untrusted device), these capabilities were not actively tested in this study. Further investigations are planned to determine how users respond to advanced recovery workflows and whether multi-device synchronization and temporary usage affect their overall security perceptions.

**Passkeys and WebAuthn.** As passwordless methods gain momentum—particularly passkeys aligned with WebAuthn—some accounts may eventually forgo traditional passwords entirely. However, not all services currently support such protocols. For the foreseeable future, many users will continue to

rely on passwords for at least a subset of their online accounts. A robust password manager thus remains relevant and can potentially integrate with passkey-based solutions, combining the benefits of emerging passwordless authentication with our password manager's security guarantees.

Overall, these findings reveal how participants navigate usability and security trade-offs when choosing a password manager, and how trust, privacy, and technical compatibility influence their decisions. By expanding future studies to include larger sample sizes, multiple devices, fallback testing, and integration with passkeys, we aim to further validate and refine our system's architecture under more varied real-world conditions.

## VII. CONCLUSION

We introduced a cloud-based password manager that enforces the use of a unique and high-entropy password for each web service constructed as a pseudo-random function of a key stored on the password manager and a user-memorable master password. Our password manager does not learn or store the master password and/or the web account password, but rather, it generates the high-entropy account password on the fly once the master password is entered into the client machine. In no case does the manager learn the master or account passwords even while processing these values. We built a robust implementation of the system and measured the performance of this implementation, which shows the underlying protocol incurs minimal overhead. To assess the usability and security of our solution from the user's perspective, we conducted a remote user study. In this study, participants used our cloud-based password manager to authenticate to various web services and provided their feedback. The results showed that participants had fewer privacy concerns about our system and perceived it as more secure compared to LastPass. These findings indicate that our password manager offers a user-friendly and secure solution for managing passwords. It addresses privacy concerns and provides a higher level of security compared to traditional password management methods. The positive user feedback and perceived trust in the system validate the effectiveness of our approach in improving password management and security.

## REFERENCES

- [1] F. Tari, A. A. Ozok, and S. H. Holden, "A comparison of perceived and real shoulder-surfing risks between alphanumeric and graphical passwords," in *Proc. 2nd Symp. Usable Privacy Secur.*, 2006, pp. 1–56.



- [2] J. Bonneau, "The science of guessing: Analyzing an anonymized corpus of 70 million passwords," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 538–552.
- [3] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," in *Proc. 12th ACM Conf. Comput. Commun. Secur.*, Nov. 2005, pp. 364–372.
- [4] Reuters, (2017). *Yahoo Says All Three Billion Accounts Hacked in 2013 Data Theft*. Accessed: Dec. 31, 2024. [Online]. Available: <https://reut.rs/2UqGoZ4>
- [5] (2014). *The Ebay Breach Explained*. Accessed: Dec. 31, 2024. [Online]. Available: <https://bit.ly/3cQpkSJ>
- [6] *The Equifax Data Breach*. Accessed: Jun. 28, 2025. [Online]. Available: <https://www.epic.org/privacy/data-breach/equifax/>
- [7] L. Constantin. (2015). *Researcher Says Adult Friend Finder Vulnerable to File Inclusion Vulnerabilities*. Accessed: Dec. 31, 2024. [Online]. Available: <https://shorturl.at/yYDIK>
- [8] (2017). *Hackers Are Using Uber's 57 Million Account Data Breach to Steal Passwords*. Accessed: Dec. 31, 2024. [Online]. Available: <https://www.thedailybeast.com/hackers-are-using-ubers-57-million-account-data-breach-to-steal-passwords>
- [9] *Dashlane Password Manager*. Accessed: Jun. 28, 2025. [Online]. Available: <https://www.dashlane.com/>
- [10] *LastPass Remembers All Your Passwords Across Every Device for Free!*. Accessed: Jun. 28, 2025. [Online]. Available: <https://lastpass.com/>
- [11] *1Password: Simple, Convenient Security*. Accessed: Jun. 28, 2025. [Online]. Available: <https://1password.com/>
- [12] (2021). *9 Popular Password Manager Apps Found Leaking Your Secrets*. Accessed: Dec. 31, 2024. [Online]. Available: <https://bit.ly/3h46lXX>
- [13] *Password Manager Onelogin Hacked, Exposing Sensitive Customer Data*. Accessed: Jun. 28, 2025. [Online]. Available: <https://zd.net/3dKKIPJ>
- [14] Z. Li, W. He, D. Akhawe, and D. Song, "The emperor's new password manager: Security analysis of web-based password managers," in *Proc. 23rd USENIX Secur. Symp. (USENIX Secur.)*, 2014, pp. 465–479.
- [15] L. Whitney. (2011). *Lastpass Ceo Reveals Details on Security Breach*. Accessed: Apr. 30, 2025. [Online]. Available: <https://cnet.co/2ANDkz5>
- [16] *What Happened When The DEA Demanded Passwords From LastPass*. Accessed: Feb. 18, 2020. [Online]. Available: <https://www.forbes.com/sites/thomasbrewster/2019/04/10/what-happened-when-the-dea-demanded-passwords-from-lastpass/?sh=22c9f61f7ebe>
- [17] S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena, "Device-enhanced password protocols with optimal online-offline protection," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, May 2016, pp. 177–188.
- [18] P. Liu, J. Blocki, and W. Bai, "Confident Monte Carlo: Rigorous analysis of guessing curves for probabilistic password models," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2023, pp. 626–644. [Online]. Available: <https://ieeexplore.ieee.org/document/10179365>
- [19] D. Wang, Y. Zou, Z. Zhang, and K. Xiu, "Password guessing using random forest," in *Proc. 32nd USENIX Secur. Symp.*, 2023, pp. 965–982. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity23/presentation/wang-ding-password-guessing>
- [20] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian, "Zipf's law in passwords," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 11, pp. 2776–2791, Nov. 2017. [Online]. Available: <https://ieeexplore.ieee.org/document/7961978>
- [21] B. B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell, "Stronger password authentication using browser extensions," in *Proc. USENIX Secur. Symp.*, Jul. 2005, pp. 1–2.
- [22] J. A. Halderman, B. Waters, and E. W. Felten, "A convenient method for securely managing passwords," in *Proc. 14th Int. Conf. World Wide Web*, 2005, pp. 471–479.
- [23] K.-P. Yee and K. Sitaker, "Passpet: Convenient password management and phishing protection," in *Proc. 2nd Symp. Usable Privacy Secur.*, 2006, p. 32.
- [24] M. Shirvanian, S. Jarecki, H. Krawczyk, and N. Saxena, "SPHINX: A password store that perfectly hides passwords from itself," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 1094–1104.
- [25] A. Hetler. (2024). *Passkey Vs. Password: What is the Difference?*. Accessed: Dec. 31, 2024. [Online]. Available: <https://www.techtarget.com/whatis/feature/Passkey-vs-password-What-is-the-difference>
- [26] I. LogMeIn. (2021). *What Makes Lastpass Secure?*. Accessed: Apr. 20, 2021. [Online]. Available: <https://support.logmeininc.com/lastpass/help/what-makes-lastpass-secure-lp070015>
- [27] K. Toubba. (2022). *Notice of Recent Security Incident: The Lastpass Blog*. Accessed: Dec. 31, 2024. [Online]. Available: <https://blog.lastpass.com/2022/12/notice-of-recent-security-incident/>
- [28] L. Abrams. (2023). *Lastpass Breach Linked to Theft of \$4.4 Million in Crypto*. Accessed: Dec. 31, 2024. [Online]. Available: <https://www.bleepingcomputer.com/news/security/lastpass-breach-linked-to-theft-of-44-million-in-crypto/>
- [29] S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena, "Two-factor authentication with end-to-end password security," in *Proc. Int. Conf. Pract. Theory Public Key Cryptogr.*, Jan. 2018, pp. 431–461.
- [30] (Jan. 2021). *Top Websites in the us by Traffic*. [Online]. Available: <https://www.semrush.com/blog/most-visited-websites/>
- [31] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange, "Elligator: Elliptic-curve points indistinguishable from uniform random strings," in *Proc. ACM SIGSAC Conf. Comput. Commun. Sec.*, 2013, pp. 967–980.
- [32] S. Seiler-Hwang, P. Arias-Cabarcos, A. Marín, F. Almenares, D. Díaz-Sánchez, and C. Becker, "I don't see why I would ever want to use it," Analyzing the usability of popular smartphone password managers," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2019, pp. 300–312. [Online]. Available: <https://dl.acm.org/doi/10.1145/3323642.3323677>
- [33] P. Arias-Cabarcos, A. Marín, D. Palacios, F. Almenárez, and D. Díaz-Sánchez, "Comparing password management software: Toward usable and secure enterprise authentication," *IT Prof.*, vol. 18, no. 5, pp. 34–40, Sep. 2016. [Online]. Available: <https://ieeexplore.ieee.org/document/7501652>
- [34] A. Hutchinson, J. Tang, A. J. Aviv, and P. Story, "Measuring the prevalence of password manager issues using in-situ experiments," in *Proc. USEC*, 2024, pp. 1–27.
- [35] S. Team. (2025). *Securden 2025: Comparative Report on Password Managers*. Accessed: Feb. 24, 2025. [Online]. Available: <https://www.securden.com/blog/index.html>
- [36] A. Karole, N. Saxena, and N. Christin, "A comparative usability evaluation of traditional password managers," in *Proc. Int. Conf. Inf. Secur. Cryptol.*, Jan. 2011, pp. 233–251.
- [37] S. Chiasson, P. C. van Oorschot, and R. Biddle, "A usability study and critique of two password managers," in *Proc. USENIX Secur. Symp.*, Jul. 2006, pp. 1–16.
- [38] G. Inc. (2021). *How to Use Google Sites*. Accessed: Apr. 10, 2021. [Online]. Available: [https://support.google.com/sites/answer/6372878?hl=en&ref\\_topic=7184580](https://support.google.com/sites/answer/6372878?hl=en&ref_topic=7184580)
- [39] Google Inc.(2021). *Create Beautiful Forms: Free Online Surveys for Personal and Professional Use*. Accessed: Dec. 31, 2024. [Online]. Available: <https://www.google.com/forms/about/>
- [40] Z. V. C. Inc. (2021). *Zoom Video Communications*. Accessed: Apr. 13, 2021. [Online]. Available: <https://zoom.us/>
- [41] TeamViewer.(2021). *Remote Access & Support*. Accessed: Apr. 13, 2021. [Online]. Available: <https://www.teamviewer.com/en-us/>
- [42] *Skype—P2P VoIP*. Accessed: Apr. 13, 2021. [Online]. Available: <http://www.skype.com/>
- [43] J. Brooke, "SUS-A quick and dirty usability scale," *Usability Eval. Ind.*, vol. 189, no. 194, pp. 4–7, 1996.
- [44] C. W. Munyendo, P. Mayer, and A. J. Aviv, "I just stopped using one and started using the other": Motivations, techniques, and challenges when switching password managers," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2023, pp. 3123–3137, doi: [10.1145/3576915.3623150](https://doi.org/10.1145/3576915.3623150).
- [45] B. Cruz. (2024). *Security.org 2024 Password Manager Industry Report*. Accessed: Feb. 24, 2025. [Online]. Available: <https://www.security.org/digital-safety/password-manager-annual-report/>
- [46] A. Bangor, P. Kortum, and J. Miller, "Determining what individual SUS scores mean: Adding an adjective rating scale," *J. Usability Stud.*, vol. 4, no. 3, pp. 114–123, 2009.
- [47] J. R. Lewis, "The system usability scale: Past, present, and future," *Int. J. Hum.-Comput. Interact.*, vol. 34, no. 7, pp. 577–590, Jul. 2018.
- [48] M. Rosemain and R. Satter. (2021). *Millions of Websites Offline After Fire at French Cloud Services Firm*. Accessed: Apr. 29, 2021. [Online]. Available: <https://shorturl.at/dvyQ1>
- [49] K. Kim. (2024). *1Password Vs. Bitwarden: 2024 Review*. Accessed: Feb. 24, 2025. [Online]. Available: <https://me.pcmag.com/en/password-managers/23767/1password-vs-bitwarden-which-password-manager-should-you-choose>
- [50] L. Millares. (2025). *Bitwarden Vs. 1Password: Key Comparisons*. Accessed: Feb. 24, 2025. [Online]. Available: <https://www.techrepublic.com/article/bitwarden-vs-1password/>