A Hidden-Password Online Password Manager

Maliheh Shirvanian*
Visa Research
mshirvan@visa.com

Christopher Robert Price
University of California Irvine
crprice@uci.edu

Mohammed Jubur
University of Alabama at Birmingham
mjabour@uab.edu

Nitesh Saxena
University of Alabama at Birmingham
saxena@uab.edu

Stanislaw Jarecki
University of California Irvine
sjarecki@uci.edu

Hugo Krawczyk
Algorand Foundation
hugo@ee.technion.ac.il

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 HIPPO, a cloud-based password manager that does not learn or store master passwords and account passwords. HIPPO is based on the cryptographic notion of device-enhanced password authenticated key exchange proven by Jarecki et al. to resist online guessing attacks and dictionary attacks. We introduce the HIPPO protocol design and report on a full implementation of the system.

1 INTRODUCTION

Decades after the introduction of passwords, password-protected systems still experience various attacks, including shoulder surfing [19], online guessing (brute force) attacks [9], and offline dictionary attacks [16]. 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 venues of attacks due to the memorability issue (e.g., writing down, storing electronically, and less frequent updates). Moreover, users often tend to re-use the same password to access multiple web services, which increases the security risks since the compromise of any one of the services also compromises the accounts with other services.

To address these issues, online password managers save the users' (encrypted) web service *account passwords* on a password file on a cloud service [1, 3, 5]. These password managers help to decrease the possibility of password breaches by generally 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 of password managers [2, 4, 6, 15].

To address this fundamental vulnerability, we introduce HIPPO[1], a secure cloud-based password manager that is built on top of a cryptographic primitive called Device-Enhanced Password Authenticated Key Exchange (DE-PAKE) [13]. HIPPO does not store but generates a secure and a randomized account password for each website upon receiving the master password from the user. Hence, it implicitly enforces high-entropy account passwords.

The randomized account password rwd is generated as a *key-ed* function of the master password using an oblivious-pseudo random function (OPRF) protocol [13] defined as F_k(pwd). Two parties involved in the protocol: (1) a client from which the user initiates an authentication attempt on the bases of a master password pwd, and (2) a cloud-based password manager that holds a unique key k per each service the user authenticates to. OPRF ensures that the password manager does not learn or store rwd or pwd or anything derived from them.

Contributions: The contributions are as follows:

1. A Hidden-Password Online Password Manager: We introduce our password manager HIPPO, built on top of the DE-PAKE cryptographic primitive. HIPPO 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 key-ed function of the master password. The high-entropy account password is registered with each service and is reconstructed during the authentication session using an oblivious pseudo random function password hardening approach that receives the master password from the user and the key from the password manager server.

2. Design and Implementation of HIPPO: We designed HIPPO as a cloud service responding to the password inquiries from the users. Our implementation consists of the server-side Node.js service and the client-side Chrome browser extension. This client-server architecture provides secure password generation via HIPPO. The account password is reconstructed on the client and is automatically entered in the password field.

2 BACKGROUND

Device-Enhanced Password Authenticated Key Exchange. The design of HIPPO is built on top of the Device Enhanced Password Authenticated Key Exchange (DE-PAKE) protocol introduced in

*Work Done at UAB.

SAC '21, March 22–26, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8104-8/21/03.
https://doi.org/10.1145/3412841.3442131

[1] HIPPO denotes HIdden-Password Password manager Online. It is a strong password manager, just like hippo is a strong animal.

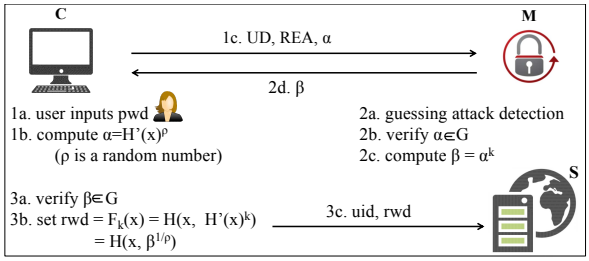

Figure 1: HIPPO protocol reconstructs account password to authenticate user

[13]. DE-PAKE is an extension of the notion 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 pwd into a high-entropy random string rwd by leveraging the device, and then uses this random string as a password input to any password authenticated key exchange protocol (PAKE). The authors of [13] 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 the need for secure communication between the device and the client.

Related Work on Storeless Password Managers. Hash-based password managers (e.g., [12, 17, 21]) represent an alternative approach to traditional password managers that generates the account password as a function of the master password on the fly. These type of password managers do not store the account passwords (unlike traditional online password managers), rather they compute the account password as the hash of the master password, once the master password is entered on the website forms. Therefore, they address the issues with the leakage of the password file. However, they are still vulnerable to offline attacks upon the compromise of the web services since they store a doubled-hash of the master password for verification on the server.

SPHINX [18] is another password manager introduced based on DE-PAKE primitive. SPHINX is a device-based password manager that transforms a human-memorable password into a random password with the aid of a smartphone. Upon receiving the master password from the user, SPHINX runs the password hardening protocol with the smartphone to reconstruct the service password on the client. SPHINX crucially relies on a smartphone to reconstruct the password. The absence of such a device due to the loss of connectivity or being out of the battery power, can cause serious usability issues preventing the user from logging in (similar to two factor authentication and device-based password managers [7, 10, 11, 20]). Besides, log in to web services from the smartphone itself would be problematic since the cryptographic key would be stored on the same device (the phone) on which the master password is entered. Our approach carries the security properties of the underlying DE-PAKE protocol. However, it addresses several concerns related to the deployment of the password manager as an online service, such as reliability of the service.

3 OUR APPROACH

3.1 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 temporary remembers a random number ρ of size τ and blinds the C's OPRF input by computing $\alpha = H'(uid, domain, \text{ctr}, \text{pwd})^{\rho}$. 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 α to the manager. UD privately identifies the user account (i.e., while hiding the username and the service domain name 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 α 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 β to C.

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

- *3a.* C verifies that β is an element of G; aborts if not.
- *3b.*C deblinds its OPRF input by computing $\beta^{1/\rho}$. C discards ρ at this step and sets $\text{rwd} = F_k(uid, domain, \text{ctr}, \text{pwd}) = H(uid, domain, \text{ctr}, \text{pwd}, H'(uid, domain, \text{ctr}, \text{pwd})^k)$.
- *3c.* C transfers rwd to the server S.

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

3.2 Threat model and Security Properties

The formal and practical security of HIPPO derives from the DE-PAKE formal framework and security proofs of [13]. HIPPO 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 of HIPPO as they follow from the results of [13]. Please refer to [13] for details.

The security definition from [13] 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., [14]).

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 [13], when applied to our setting, imply the following security properties for HIPPO (below, Dict denotes a dictionary from which 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 pwd, runs HIPPO with M on input pwd, and uses the resultant value rwd in an authentication session with S. Only if the latter authentication succeeds, does the attacker learn pwd. This means that to be successful A needs to perform an online attack of order |Dict| against *both* S and M, and that offline attacks are ineffective.

(2) **Resistance to Attacks upon Compromise of M:** In case server M is compromised, [13] 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 HIPPO is for A to use the compromised M's key (an OPRF key) to compute rwd for each value of pwd in Dict and test each obtained rwd in online interaction with S. Thus, even with a compromised M, the attacker still needs online interactions with S in the order of |Dict|. Also here, offline attacks are ineffective. These properties are to be contrasted with password managers that store a list of passwords encrypted under pwd and where the compromise of the password manager typically leads to an offline-only attack on pwd and all encrypted passwords. In the case of HIPPO, even if M is fully controlled by the attacker, *nothing* (in the strongest information theoretic sense) is learned about pwd or on the individual rwd.

(3) **Resistance to Attacks upon Compromise of S:** When a server S is compromised, [13] shows an upper bound of $q_M/|\text{Dict}|$ on the attacker's probability of success. It implies that the best attack against HIPPO in this case is an exhaustive online attack against M using guessed values pwd to obtain rwd that can then be validated against the compromised state of S. Note that an exhaustive attack on rwd could also be possible in this case; however, since rwd is a high-entropy secret such attack is infeasible. Moreover, learning rwd directly from a server S (e.g., breaking the server's TLS communication), does not compromise either pwd or any other password rwd as 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 |Dict| is possible. In it, one lists all possible pwd values and uses the internal (OPRF) key at M to calculate rwd for each pwd. Then rwd is tested against the compromised state at S.

(5) **Resistance to Phishing Attacks:** An additional important security property attained by the HIPPO implementation is security against phishing attacks obtained thanks to the use of the server's domain in the derivation of rwd.

(6) **Full cryptographic security with random UT:** HIPPO accommodates an optional value UT that can be set by the user to 0 or to a high-entropy random value. In the latter case (in which UT needs to be carried to all the user's client machines) and as long as UT is not disclosed, *no feasible attack*, online or offline, or upon the compromise of both M and S, exists against the master password pwd and its derivatives rwd as any such attack would require to guess UT. On the other hand, if UT is eventually disclosed, HIPPO still guarantees all the above properties.

4 SYSTEM IMPLEMENTATION

4.1 HIPPO Client

We consider NIST P-256 as the group G over which the operations run. We assume that the user has set a recovery email address REA on the client and has verified it through standard verification processes (e.g., by providing a code or clicking a link emailed from the password manager server to REA). For the sake of simplicity, the user only sets one recovery email.

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 [17]), 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) $\alpha = H'(uid, domain, \text{ctr}, \text{pwd})^\rho$ 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., β). If the received value satisfies the ECC curve equation, β is raised to $1/\rho$ to unblind $H'(uid, domain, \text{ctr}, \text{pwd})^k$. This value is then mapped to rwd to provide the generated password. The generated password is then populated in the form.

Reading and Replacing the Password from/to Forms: This functionality is a fork of the chrome port of Stanford's PwdHash extension [17]. 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.

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, HIPPO incorporates the Secure WebSocket protocol for the client-manager communication. This choice is made to follow the browsers' content security policy that prevents mixed content. The HIPPO Chrome browser extension provides the TLS encrypted WebSocket communication with the manager to exchange messages.

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 ρ and computes α encoded into (x, y) pair coordinates. Upon receiving the computed β from the manager it performs inverse exponentiation and maps the generated point to a hex string using a SHA-256 hash function as $H(uid, domain, \text{ctr}, \text{pwd}, \beta^{1/\rho})$. This hexstring is encoded to rwd by applying password constraints as mentioned earlier.

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, HIPPO 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 where ctr is managed by the server or a separate service or app designed for this purpose.

Extension Options Page: The client browser extension has an options page for configuring user's preferences such as recovery email, ctr and UT.

4.2 HIPPO 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.

Secure WebSocket Server: Although the HIPPO 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 accepts Secure WebSocket connections.

Account Look-up: HIPPO 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 HIPPO 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 protected by HIPPO. 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), HIPPO computes the key-ed hash of the user id and domain name as UD, such that UD=SHA256-HAMC(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.

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 SHA256-HAMC($key = mk, input$ = (UD, REA)).

ECC Computation: As part of the HIPPO protocol, the manager responds to the OPRF message received from the client. The manager receives the computed $\alpha = H'(uid, domain\text{ctr}, \text{pwd})^{\rho}$ 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 [18], however, other alternatives robust to side channels would be possible [8]. 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 β encoded into a (x, y) coordinate to the client.

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.

Rate Limiting: Since HIPPO is implemented as an online service with no need for user authentication it may be a target for an active attacker guessing 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 used two standard rate limiting approaches, namely connection count per time interval and GeoIP matching.

REFERENCES

[1] [n. d.]. 1Password: Simple, Convenient Security. ([n. d.]). https://1password.com/.
[2] [n. d.]. 9 Popular Password Manager Apps Found Leaking Your Secrets. https://bit.ly/3h46lXX. ([n. d.]).
[3] [n. d.]. Dashlane Password Manager. ([n. d.]). https://www.dashlane.com/.
[4] [n. d.]. LastPass CEO reveals details on security breach. https://cnet.co/2ANDkz5. ([n. d.]).
[5] [n. d.]. LastPass remembers all your passwords across every device for free! ([n. d.]). https://lastpass.com/.
[6] [n. d.]. Password manager OneLogin hacked, exposing sensitive customer data. https://zd.net/3dKKlPJ. ([n. d.]).
[7] Nora Alkaldi and Karen Renaud. 2016. Why Do People Adopt, or Reject, Smartphone Password Managers? (2016).
[8] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. 2013. Elligator: elliptic-curve points indistinguishable from uniform random strings.
[9] Joseph Bonneau. 2012. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE.
[10] Emiliano De Cristofaro, Honglu Du, Julien Freudiger, and Greg Norcie. 2013. A comparative usability study of two-factor authentication. *arXiv preprint arXiv:1309.5344* (2013).
[11] Nancie Gunson, Diarmid Marshall, Hazel Morton, and Mervyn Jack. 2011. User perceptions of security and usability of single-factor and two-factor authentication in automated telephone banking. *Computers & Security* 30, 4 (2011), 208–220.
[12] J Alex Halderman, Brent Waters, and Edward W Felten. 2005. A convenient method for securely managing passwords. In *Proceedings of the 14th international conference on World Wide Web*. ACM.
[13] Stanislaw Jarecki, Hugo Krawczyk, Maliheh Shirvanian, and Nitesh Saxena. 2016. Device-enhanced password protocols with optimal online-offline protection. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM.
[14] Stanislaw Jarecki, Hugo Krawczyk, Maliheh Shirvanian, and Nitesh Saxena. 2018. Two-factor authentication with end-to-end password security. In *the International Conference on Practice and Theory of Public Key Cryptography*.
[15] Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Song. 2014. The emperor's new password manager: Security analysis of web-based password managers. In *23rd USENIX Security Symposium (USENIX Security 14)*. 465–479.
[16] Arvind Narayanan and Vitaly Shmatikov. 2005. Fast dictionary attacks on passwords using time-space tradeoff. In *Proceedings of the 12th ACM conference on Computer and communications security*. ACM.
[17] Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C. Mitchell. 2005. Stronger password authentication using browser extensions. In *USENIX Security Symposium*.
[18] Maliheh Shirvanian, Stanislaw Jarecki, Hugo Krawczykz, and Nitesh Saxena. 2017. SPHINX: A password store that perfectly hides passwords from itself. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 1094–1104.
[19] Furkan Tari, Ant Ozok, and Stephen H Holden. 2006. A comparison of perceived and real shoulder-surfing risks between alphanumeric and graphical passwords. In *Proceedings of the second symposium on Usable privacy and security*. ACM.
[20] Luren Wang, Yue Li, and Kun Sun. 2016. Amnesia: a bilateral generative password manager. In *Distributed Computing Systems (ICDCS), IEEE 36th International Conference on*.
[21] Ka-Ping Yee and Kragen Sitaker. 2006. Passpet: convenient password management and phishing protection. In *Proceedings of the second symposium on Usable privacy and security*.