SPHINX: A Password Store that Perfectly Hides from Itself

Maliheh Shirvanian*, Stanislaw Jarecki†, Hugo Krawczyk‡ and Nitesh Saxena *
*University of Alabama at Birmingham Email: maliheh@uab.edu, saxena@cis.uab.edu
†University of California at Irvine Email: stasio@ics.uci.edu
‡IBM Research Email:hugo@ee.technion.ac.il

***Abstract*—Password managers (aka stores or vaults) represent a security technique that allows a user to store and retrieve (*usually* high-entropy) passwords for her multiple password-protected services by interacting with a "device" serving the role of the manager (e.g., a smartphone or an online third-party service) on the basis of a single (low-entropy) master password. Existing password managers work well to defeat offline dictionary attacks upon web service compromise, assuming the use of high-entropy passwords is enforced. However, they are vulnerable to leakage of all passwords in the event the device is compromised or malicious, due to the need to store the passwords encrypted under master password and/or the need to input the master password to the device (as in smartphone managers). Evidence exists that password managers can be attractive attack targets.**

In this paper, we introduce a novel approach to password management, called SPHINX, which remains secure even when the password manager itself has been compromised. In SPHINX, the information stored on the device is *information theoretically independent* of the user's master password — an attacker breaking into the device learns *no information* about the master password or the user's individual passwords. Moreover, an attacker with full control of the device, even at the time the user interacts with it, learns *nothing* about the master password — the password is not entered into the device in plaintext form or in any other way that may leak information on it. Unlike existing managers, SPHINX produces strictly high-entropy passwords and makes it compulsory for the users to register these passwords with the web services, which defeats online guessing attacks and offline dictionary attack upon service compromise. As an important added advantage over existing managers, that require some form of secure channels between the device and the client machine from which the user logs in, SPHINX can work with fully *unsecured channels* offering extra layer of security. The design and security of SPHINX is based on the device-enhanced PAKE model of Jarecki et al. that provides the theoretical basis for this construction and is backed by cryptographic proofs of security.

While SPHINX is suitable for different device platforms, in this paper, we report on its concrete instantiation on smartphones given their popularity and trustworthiness as password managers (or even two-factor authentication). We present the design, implementation and performance evaluation of SPHINX, offering prototype browser plugins, smartphone apps and transparent device-client communication. We also report on a lab-based usability study of SPHINX, which indicates that users' perception of SPHINX security and usability is high, and the overall user experience is significantly better than current smartphone managers, and satisfactory when compared to regular password-based authentication. We also provide a comprehensive comparative evaluation of SPHINX with other password managers. Finally, we discuss how SPHINX may be extended in the future to an online service for the purpose of back-up or as an independent password manager.

I. Introduction

The central role of passwords for authentication and for gaining access to resources, from casual website visits to national security, is well known. Equally well known are the major security vulnerabilities of such mechanisms spawned by the limitations of human memory and the consequent low entropy of passwords (e.g., [17], [20], [41], [47]). Such low-entropy passwords are vulnerable to both online guessing and offline dictionary attacks that build on password dictionaries from which a significant portion of passwords are chosen. Candidate passwords for authenticating a user to a server can be tested by an attacker through online interactions with the server. Even more seriously, an attacker breaking into a server can mount an offline attack that uses information stored on the server to test the different passwords in the dictionary. Such offline dictionary attacks are an increasingly important concern (see, e.g. [14], [15]), especially in light of frequent attacks against major commercial vendors, as recently experienced, e.g., by PayPal [2], LinkedIn [8], Blizzard [3] and Gmail [7]. The offline attacks are particularly devastating as a single server break-in may lead to extraordinary numbers of compromised passwords [6], [16]. Furthermore, since many users re-use their passwords across multiple services, compromising one service often compromises user accounts at other services.

Numerous approaches have been proposed by researchers and practitioners to improve the security of passwords from the client-side or user-side alone (i.e., without making any changes to a persistent server that uses traditional password-based authentication). One broad class of such approaches is referred to as *password managers* (also known as password stores or vaults) and forms the central focus of this paper.

Traditional password managers (e.g., [1], [12]) allow the user to store and retrieve (*usually* high-entropy) passwords, denoted by rwd, for her multiple password-protected services by interacting with a "device" serving the role of the manager (a smartphone or an online third-party service)[1] on the basis of a single (low-entropy) master password, denoted pwd. These rwd's are usually stored on the device encrypted under pwd. In case of online password managers, the user provides pwd to the service, which then unlocks and sends rwd to the user (over a protected channel). In case of smartphone managers, the user enters pwd directly on the device, the device unlocks

[1]Password managers that store rwd's on the client machine (browser) itself (e.g., [11]), make it very hard for the user to move from one client to another, and may actually be equivalent to online managers to enable syncing of stored passwords across multiple clients through an online server.

rwd and displays it on the screen, and the user then manually copies pwd over to the client machine.

These password managers clearly alleviate the memorization burden on the user, and work well to defeat offline dictionary attacks upon web service compromise, assuming the use of high-entropy rwd's is enforced. However, they are vulnerable to leakage of rwd's in the event the device is compromised or is itself malicious, due to: (1) the storing of the passwords rwd's encrypted under pwd, and/or (2) the need to input the master password to the device (smartphone managers). In the first case, after retrieving encrypted rwd's from the storage unit of the password manager, the attacker can launch an offline dictionary attack against pwd [2]. Such attacks are a serious concern in light of recent breaches against commercial online managers [46] and exfiltration approaches discussed in the literature [37]. In the second case, pwd is directly exposed to the attacker. With the advent of mobile computing, malware that can compromise mobile devices is becoming a major threat [4], [10], [49], and thus existing smartphone managers open up a significant vulnerability of exposing pwd upon entry and/or leaking rwd's upon offline dictionary attack.

Cracking-resistant password encoding strategies have been proposed in the literature to render offline dictionary attacks ineffective [23]. They introduce the notion of outputting decoy passwords to an attacker who compromises the manager and attempts to decrypt the passwords with a wrong master password. Since the attacker is not aware of the correct password, any attempt to login with the decoy passwords can be prevented on the server, and raise an alert. However, such a scheme seems to be vulnerable to an attack presented in a very recent work [29], based on differences in the distribution of the passwords.

Other advanced password management solutions have been proposed that do not require storage of rwd's [30], [42], [48]. For example, PwdHash [42], maps pwd to a rwd by hashing (pwd, domain) pair and registering it as a strong password with the server. PwdHash deterministically transforms a user's password into a more complex password but this transformation does not protect against offline dictionary attacks at a compromised web service. Moreover, if a user uses the same memorable password pwd with PwdHash for different services, the compromise of a single server leads to the discovery of pwd via an offline dictionary attack and then to the (deterministic) calculation of all the user's passwords derived from pwd. In our case, the compromise of a server does not help the attacker learn either the randomized password rwd used for that server or the underlying password pwd. We provide a brief review and analysis of many existing password managers in Section VII details appear in Appendix A.

In this paper, we introduce, build and study SPHINX, a new password manager that offers a high level of security even in case the password manager itself is compromised. SPHINX's most appealing features are: (1) the information stored in the device is *information theoretically independent* of the user's master password pwd; hence, an attacker breaking into the device learns *no information* on pwd or the user's individual passwords rwd's; and (2) an attacker with full control of the device, even at the time the user interacts with it, learns *nothing* about pwd; pwd is never entered into the device in plaintext form or in any other way that may leak information on it. The above properties hold unconditionally, even against a computationally unbounded attacker.

Moreover, SPHINX produces strictly high-entropy rwd's and enforces the users to register these passwords with the web services, while current password managers may let the users choose and register low-entropy passwords thereby still opening up the vulnerability to online guessing attacks and offline dictionary attacks upon web service compromise. As an added advantage over existing managers that require some form of secure channels between the device and the client machine from which the user logs in, SPHINX can work with fully *unsecured channels* offering additional layer of security. Given the numerous vulnerabilities of PKI to certification failures and man-in-the-middle (MitM) attacks (either due to programmatic errors or human mistakes), e.g., [25], [28], [44], relying upon secure channels in existing online managers may be problematic, a situation SPHINX completely avoids.

The design and security of SPHINX is based on the device-enhanced password authenticated key exchange (DE-PAKE) model of Jarecki et al. [32] that provides the design and theoretical basis for this construction and is backed by cryptographic proofs of security. The core technique is the use of an efficient oblivious pseudo random function (OPRF) scheme [26], [27] that transforms a human-memorable password into a random password with the aid of a device without the need to store the passwords on the device and without the device learning anything about the password even when computing on it. Specifically, when using SPHINX, for each service with which the user has an account, the device stores a unique key k. This key is used to map the user-memorized password pwd (input by the user into the client machine) into a randomized password $\mathsf{rwd} = F_k(\mathsf{pwd}|\mathsf{domain})$ using the (oblivious) PRF F_k based on a simple protocol between the device and the client. The details of the protocol are described in the next section.

In sum, SPHINX offers the following *simultaneous* combination of security features:

1) Resistance to online guessing attacks, due to the use of high-entropy and mutually independent passwords rwd registered with web services.
2) Resistance to offline dictionary attacks under server compromise (or server being malicious), due to the server storing a salted one-way hash of rwd, a randomized input. [32]
3) Resistance to phishing attacks, due to the use of website domain in the computation of rwd.
4) Resistance to offline dictionary attacks under device compromise (or device being malicious), in particular hiding the user's master password information theoretically.
5) Resistance to eavesdropping and man-in-the-middle attacks on the device-client channel *without* the need to establish a secure channel (the properties of the OPRF protocol executed over this channel ensure that no additional protection is needed).

[2] If *all* rwd's are *fully* random then with careful enough encryption an offline dictionary attack in this case may be avoided; yet concern (2) would remain in this case too.

TABLE I: Security properties of SPHINX in contrast to current managers. Highlighted cells represent the unique advantages offered by SPHINX compared to other managers.

Resistance to:	SPHINX	Current Managers
Offline dictionary attacks under server compromise (or server being malicious)	Yes, by enforcing random independent passwords per site.	Only if manager enforces high-entropy per-site passwords.
Phishing attacks	Yes, by incorporating domain name.	Only the hash-based managers [30], [42], [48].
Offline dictionary attacks under device compromise (or device being malicious)	Yes, *perfect security* (information theoretic secrecy).	No, passwords are stored or entered on the manager.
Eavesdropping and/or man-in-the-middle attacks on the device-client channel	Yes, without the need to establish a secure channel.	Enforced by external mechanisms or physical security assumptions$*$.

$*$ *Current online managers require confidential and authenticated channels, while current smartphone managers require confidential channels.*

The last two security properties are unique to SPHINX, not offered by any existing password managers. The first security property is provided by default in SPHINX, while this property may or may not be provided by existing managers depending upon whether or not high-entropy passwords are enforced by the manager. The contrasts are summarized in Table I. SPHINX also increases security against online guessing attacks by making use of a strictly random password registered with the service.

Security is not the only attractive feature of SPHINX. SPHINX also strives to provide a level of user experience close to that of password-only authentication but without the burden of remembering multiple passwords, and with full-entropy per-site derived passwords.

Detailed Contributions: While SPHINX is suitable for different types of devices, in this paper, we report on its concrete instantiation developed on personal smartphones given their popularity and trustworthiness as password managers (or even two-factor authentication) as suggested in the study of [33].

1) ***A Novel Password Manager*** (Section II): We introduce SPHINX, a novel cryptographic password manager application that *perfectly hides passwords from itself*. SPHINX is a novel application of the general device-enhanced password key exchange (DE-PAKE) framework fron [32]. DE-PAKE is a broad modular cryptographic primitive with several possible applications. SPHINX is one such important applications (not studied in [32]). using an instantiation of DE-PAKE that works *with no modification* on the current web services that use password-only authentication (in particular, allowing for the use of SPHINX with typical TLS-enabled services that used the password-over-TLS protocol). This practical application was not studied by the authors of [32].
2) ***System Design and Implementation*** (Section III): We present the design, implementation and performance evaluation of a full smartphone-based SPHINX system offering a prototype browser (Chrome) plugin and a device (Android) app[3]. As a main component of our design, we highlight and address the challenges associated in building transparent and robust bidirectional browser-device communication.
3) ***Usability Study*** (Sections IV and V): We report on a usability study of our SPHINX smartphone manager conducted in lab settings with participants of different educational backgrounds but *none skilled in computer security*. The study follows a methodology in line with that of prior notable studies of password managers [24], [34]. Our results suggest that users' perception of SPHINX security and usability is high, and that SPHINX user experience is satisfactory when compared to regular password authentication using a human-memorable password (a baseline used in our study). We also show, based on simple inspection analysis of required user interaction flows, that SPHINX smartphone manager imposes lesser cognitive load on the users than traditional smartphone managers.
4) *Comparative Analytical Exposition* (Sec. A): We provide a comprehensive analytical evaluation of SPHINX comparative to other password management approaches based on an adaptation of an existing elaborate framework of security, usability and deployability metrics [21]. Our analysis shows that SPHINX is a preferred choice assuming the presence of a device during the login process.

Scope of the Work: Just like any other smartphone password manager (and even currently deployed two-factor authentication mechanisms), our studied SPHINX instantiation assumes the availability of the smartphone during the authentication process. Based on this assumption, it simultaneously provides security properties 1-4 above, and a satisfactory level of user experience when compared to regular password-only authentication.

Like *any* password management approach, providing protection in the event of client compromise is beyond the scope of SPHINX. While some protection is provided by SPHINX in the form of anti-phishing defenses and the ability to block remote adversarial activity by requiring user confirmation on the device upon SPHINX invocation, comprehensive defenses based on two-factor authentication (TFA) techniques would entitle server-side changes which password managers avoid. Integrating SPHINX with a TFA solution is possible but we leave this as an item of future work.

A further extension of our work could support a SPHINX online instantiation that replaces or complements the smartphone-based instantiation (e.g., as a main password manager or as a backup option to the smartphone-based instantiation). Fortunately, the security properties of SPHINX, particularly its resistance to device compromise and security against active man-in-the-middle attackers on client-device channel without the need for secure channels, make the online variant very appealing. Although we discuss how our work can be extended in the future to such an online case, building and testing the full implementation is beyond the scope of the current paper.

[3]A brief video, demonstrating the implemented prototype can be found at https://sites.google.com/site/stopsmanager/demo

II. Our Approach

A. Background

We first review the notion of Device-Enhanced Password-Authenticated Key Exchange (DE-PAKE) introduced in [32], a cryptographic primitive which gives rise to our SPHINX application. DE-PAKE [32] securely transforms a user-memorable password into a full-entropy random string (strong password) by leveraging a secondary device, and then uses this random password as an input to any password-based authenticated key exchange protocol (PAKE) [18]. In [32], authors developed such a "password-to-random" (PTR) protocol functionality and studied its composition with any PAKE protocol, giving rise to a DE-PAKE protocol that is resistant to online guessing attacks, offline dictionary attacks (under server and device compromise), without the need for secure authenticated communication between device and the authentication terminal (client).

In DE-PAKE protocol model, there are four parties: user, client, server, and device, individually denoted U, C, S, and D, respectively. U's goal is to authenticate itself to S via C by making use of a simple password and a personal device D. The protocol has two phases: initialization and authenticated key exchange. In the initialization phase, U chooses a password from a given dictionary Dict. The initialization phase also includes the device-client communication that establishes the state stored at D, as well as interaction with S producing a user's state $\sigma_{\mathsf{S}}(\mathsf{U})$ that S stores while U only remembers its password. After initialization, the link between U and D is subject to the same (unrestricted) man-in-the-middle adversarial activity as in the links between U and S. In the authenticated key exchange phase, U interacts with D, C and S over insecure (adversary-controlled) channels to authenticate itself to S and establish session keys to protect subsequent communication with S (e.g., to download emails).

The PTR protocol design [32] follows the "password hardening" approach of Ford and Kaliski [26] (hence termed FK-PTR), but *without* the need for authenticated channels between D and C. DE-PAKE model assumes a fully capable man-in-the-middle attacker active on all the links between all parties and one is allowed to compromise servers and devices at will. The idea in the Ford-Kaliski (FK) scheme is for C to interact with one or more auxiliary servers in order to map the user's (non-random) password into a (random) secret (the "hardened password"), taken from a dictionary unknown to the attacker. This secret is then used by U to authenticate to S. DE-PAKE adopts this idea but replaces the auxiliary servers with a single device, where the device-client channel is *unauthenticated*.

B. SPHINX Overview and Features

SPHINX is a novel compelling application of the DE-PAKE approach [32]. It is a password manager, transparent to any existing web service that deploys password authentication. As shown in Figure 1, U registers a hardened randomized password rwd with S, but only remembers a memorable password pwd that could be the same for multiple accounts (we use the terms master password and memorable password interchangeably). For each server S with which the user U has an account, the device D stores a unique key k. The key is used to map the user-memorized password pwd into a randomized password $\mathsf{rwd} = F_k(\mathsf{pwd}|\mathsf{domain})$ using the (oblivious) PRF F_k. pwd and rwd are never stored in C and (in contrast to the current password managers – see Section VII) neither rwd nor pwd is ever stored in or exposed to D. Instead, D and C run the PTR protocol to *obliviously* compute rwd at the login time. The details of the protocol are described in the next subsection. SPHINX offers several key security guarantees simultaneously (individually as well as combined), namely:

1) **Resistance to offline dictionary attacks under server compromise (or server being malicious)**: In SPHINX, the server stores the salted-hash of the randomized password rwd, and hence the compromise of a server does not help the attacker to learn rwd (or pwd). Note that a dictionary attack is *infeasible* since rwd does not belong to a dictionary known to the attacker (unlike other managers which may not necessarily use randomized passwords).
2) **Resistance to phishing attacks**: SPHINX combines pwd with the domain name of the web service to compute rwd to provide protection under a phishing attack. The phisher can not even perform an offline dictionary attack to learn rwd or pwd.
3) **Resistance to offline dictionary attacks under device compromise (or device being malicious)**: SPHINX does not leak any information about pwd or rwd to the device or a potential malicious code running on the device, since neither the pwd nor rwd is stored or entered on the device. Therefore, any offline attack against the device (remote or physical) does not reveal any information about the pwd or rwd, even during a user's active session.
4) **Resistance to eavesdropping and man-in-the-middle attacks on the device-client channel**: SPHINX does not require a secure channel between the device and client, but is secure against eavesdropping and MITM attack over device-client channel.

As far as online guessing attacks are considered, the security provided by smartphone-based SPHINX is equivalent to that of the high-entropy rwd's. Other smartphone managers have the same level of security against online guessing attack, however, rwd's may or may not be high-entropy in other managers. When considering the online instantiation of SPHINX, the security level against online guessing attack is equivalent to that of the master password pwd, in line with other online managers.

Security is not the only compelling feature of SPHINX. It also offers the following usability advantages:

1) **Use of a human-memorable password**: In SPHINX, the user simply remembers pwd but registers a strong randomized password rwd with the service. Moreover, pwd can be reused over multiple accounts without compromising security, while the randomized password rwd is unique to each service.
2) **Easy password updates:** Rather than asking the user to frequently change the password and memorize the updated password (as is a common practice today), only the key on the device can be changed, which

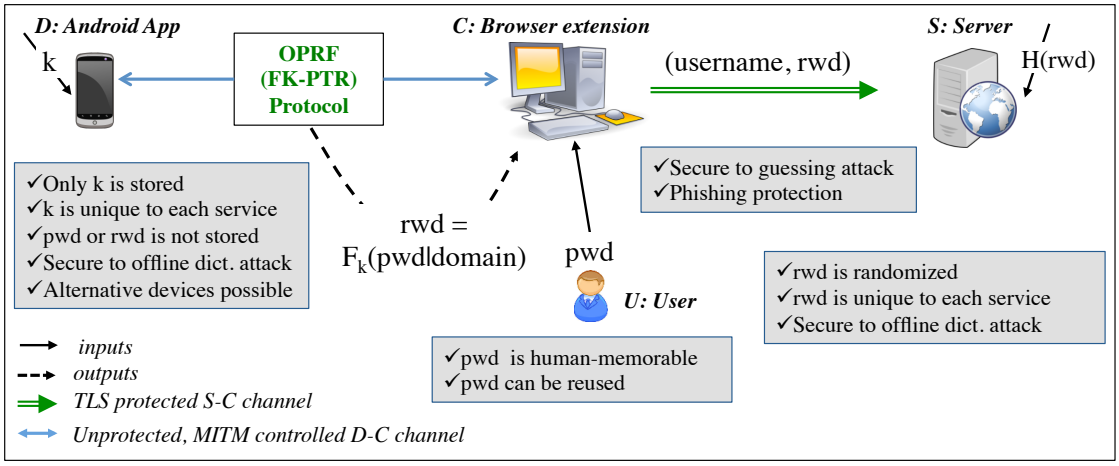

Fig. 1: A high-level overview of the SPHINX System. U enters memorable password pwd and approves the communication on the D (explicit consent), D and C run an OPRF protocol (instantiated as FK-PTR) to construct a randomized password rwd, C sends rwd to S over SSL/TLS to authenticate to the service. A smartphone-based instantiation of SPHINX is shown, since this is the instantiation developed and tested in the paper. However, the device can be replaced with an online service.

Setup

- *Group G.* The scheme works over a cyclic group G of prime order q, $|q| = \tau$, with generator g.
- *Hash functions* H, H' map arbitrary-length strings into elements of $\{0,1\}^\tau$ and G, respectively, where τ is a security parameter.
- *OPRF.* For a key $k \leftarrow Z_q$, we define function F_k as $F_k(x) = H(x, (H'(x))^k)$.
- *Parties.* User U, Client C, Device D, Server S.

Initialization Phase

- D chooses and stores OPRF key $k \leftarrow Z_q$;
- U chooses and remembers a memorable password pwd $\leftarrow$ Dict;
- C and D interact to construct "randomized password" rwd $= F_k$(pwd|domain) on input pwd from U and k from D;
- Server S stores one-way hash of the "randomized password" rwd.

Login Phase

- **Client-Device Interaction (FK-PTR)**
  1) C chooses $\rho \leftarrow Z_q$; sends $\alpha = (H'(\text{pwd}|\text{domain}))^\rho$ to D.
  2) D checks that the received $\alpha \in G$ and if so it responds with $\beta = \alpha^k$.
  3) C sets rwd $= H(\text{pwd}|\text{domain}, \beta^{1/\rho})$.
- **Client-Server Interaction**

  U authenticates to S using the "randomized password" rwd submitted over SSL/TLS channel.

Fig. 2: SPHINX Protocol Details

provides an improved level of usability.

3) **Use of multiple types of devices:** Since no information about pwd or rwd is learned by the device and the device-client channel does not need to be a secure channel, a personal device could be a smartphone, a wearable device (e.g., smartwatch) or even an online service (as will be discussed in Section VI).

All the security and usability features are offered by SPHINX simultaneously, while other password management schemes only provide a partial subset of these properties, especially no other scheme provides the last two properties, as will be elaborated in Section VII.

C. SPHINX Protocol and System Details

In the standard password-only authentication schemes, U authenticates to S using pwd. In the SPHINX protocol as shown in Figure 2, C runs an instance of password authentication protocol not on pwd, but on value rwd $= F_k(\text{pwd}|\text{domain})$ where F is a pseudorandom function (PRF) and k is a key held by D. Before authenticating to S, C contacts D (through a client application) and obtains the rwd value using a special-purpose (oblivious) PRF-evaluation protocol. Without knowledge of k, the value rwd has full entropy in the range set of function F and hence dictionary attacks do not apply against rwd (not even if the server is compromised). Since D holds only the PRF key k and S holds only information related to pseudorandom value rwd $= F_k(\text{pwd}|\text{domain})$, we can ensure that offline attacks against pwd are also infeasible when S and/or D are compromised. In the case of D compromise, the user's password is not exposed. Rather, security reduces to the security of the memorable password (the attacker who obtains the device can attempt to guess the memorable password only in an online attack against S).

The PRF protocol used in SPHINX is defined in [32] as *PTR*, and is designed in a way that neither the device or a MITM learn anything about the password (pwd or rwd) and no one other than the device learns anything about the key k. In addition, no verification information is transmitted between the D and C and no such information is stored at D. Note that

no authenticated or secret channel between C and D can be assumed as this would require pairing C and D, or knowing D's public key in advance, or storing pwd-related information at D (the latter would open pwd to an offline dictionary attack upon compromising D).

The implementation of the SPHINX protocol is based on an instantiation of the DE-PAKE primitive [32]. This instantiation assumes a cyclic group G of prime order q, $|q| = \tau$, with generator g. At initialization, U chooses and remembers password pwd while D chooses and stores $k \leftarrow Z_q$. To retrieve rwd, C first blinds pwd by raising the hashed value $H'(\mathsf{pwd}|\mathsf{domain})$ to a random exponent ρ, and sends it to D. This perfectly hides pwd from D and from any eavesdropper on the $\mathsf{U} - \mathsf{D}$ link. D checks that the received value is in the group G and if so it raises it to the secret exponent k. Now, C can de-blind this value by raising it to the power $1/\rho$ to obtain $H'(\mathsf{pwd}|\mathsf{domain})^k$. Finally, C hashes this value with pwd to obtain rwd.

Note that D contains no information related to pwd hence an attacker interacting with D or even breaking into it, learns nothing about pwd. Also, C does not run any test on the value reconstructed in the FK-PTR protocol. Hence, an attacker that interacts with C in the role of D does not learn anything about pwd from watching the behavior of C. These "obliviousness" and minimality properties of FK-PTR are essential to achieve PTR security. The security of SPHINX directly follows from the security of PTR and DE-PAKE. For formal security arguments underlying PTR and DE-PAKE, we refer the reader to [32].

III. Design, Implementation & Performance Evaluation

In this paper, we instantiate the SPHINX system using the smartphone as the device serving the role of the password manager. The resulting SPHINX system has two essential components (as shown in Figure 1), namely, the SPHINX browser extension and the SPHINX Android application communicating with each other over an unprotected channel to reconstruct the hardened randomized password from the memorable password.

A. SPHINX Browser Extension

Step 1–Reading the password: The browser extension listens to the keyboard events generated on login-page and gets activated once a predefined "@@" password prefix or the "F2" function key is entered in the password field. This design decision is similar to the approach of PwdHash [42] and allows the user to choose whether to use the service for a particular website or not (i.e., only passwords that are preceded by the prefix undergo the protocol). After getting activated, the extension reads the input password. We note that the additional password prefix is only a design choice and can be discarded from the design at no additional security/usability cost. For example, an alternative design choice is to ask the user to enable SPHINX for each service at the enrollment time.

Step 2–Hashing the password into the elliptic curve: The entered password is input in a "Hash-into-Elliptic-Curve" function. We call this function H' (Figure 2). We implemented H' using the Stanford Elliptic Curve Cryptography and Core JavaScript libraries for curve and field computation, and CryptoJS library for SHA-256 computation. The Hash-into-Elliptic-Curve function maps the password into a point on NIST P-256 curve. In this implementation, SHA-256 of the input and the iteration counter is computed and truncated into an element in Z_q, and the computed value is considered as the x coordinate of a point on the curve if the y value associated with it is a quadratic residue (i.e. x and y satisfy the curve equation). Otherwise, the same computation is repeated until a curve element is obtained. Such a point on the curve is the output of the hash function[4]. To add resistance against phishing attacks, password is concatenated with the domain name of the website and then is input into H'.

Step 3–FK-PTR OPRF protocol: After computing the hash, the extension follows its role in the OPRF function to blind the password. The OPRF function [31] is defined as $F_k(x) = H(x, (H'(x))^k)$ with input x from the client and k from the device. The OPRF works over group G of prime order p, which in our implementation is an elliptic curve NIST P-256 group. The input to the OPRF function is the password concatenated with the domain name of the visited page. As described in Figure 2, the extension picks a random number $\rho \in Z_q$ and raises hash value of the input to the power ρ (note that the use of the blinding factor ρ hides the password with information-theoretic security), and sends it to the device (we call this value α). In response, the extension receives $\beta = \alpha^k$ from the device. After checking the group membership of β, the extension reconstruct the randomized password by raising the received value to the power of $\rho^{-1} \in Z_q$ and then computing SHA-256 hash of the calculated value. We used Stanford Random Number Generator JavaScript API and CryptoJS SHA-256 to generate the random number ρ. The communication channel between the extension and the device will be discussed in more detail in Section III-C.

Step 4–Entering the randomized password: The final step performed by the extension is in line with the PwdHash implementation. The output of the OPRF is encoded to a random combination of letters, numbers and symbols matching the password requirement of the visited website, and is re-entered in the password field of the login page.

B. SPHINX Android Application

Step 1–Starting the FK-PTR protocol: In the first step the Android app receives $\alpha = H'(pwd|domain)^\rho$ from the client, checks the group membership of α, and computes $\beta = \alpha^k$. The OPRF key k is picked by the device at the initialization phase and is stored on the device. All elliptic curve functions in our app are based on Java Security and Spongy Castle libraries.

Step 2.1–Explicit Consent Mode: In this optional step, which we consider as the primary design option for SPHINX an alert message is displayed on the device to make the user aware of the ongoing login process. The user is required to confirm the alert before the protocol continues to the next step. We call this alert box the explicit consent on the device. This design choice is made to ensure that the device does not

[4] An alternative, robust to side channels, is to use a hashing-into-the-curve mechanism such as Elligator 2 [19].

respond to unauthorized requests (without user's awareness), preventing an attacker who obtains the user's master password from authenticating to the service. We evaluated this option in our usability study presented in Section V.

Step 2.2–Zero-Interaction Mode: In the second option, user can disable the explicit consent requirement, for the application to run on the device with zero interaction with the user, without seeking for user's approval. This design choice is assumed to be more usable and transparent to the user (we refer to this feature as "Physically Effort-less" in Section VII). Using this option, an attacker who has obtained the master password (e.g., with a shoulder surfing attack), might be able to login to the service (we refer to this feature as "Resilient-to-Physical-Observation" in Section VII). Our hypothesis is that since this model does not require any extra action from the user's side, compared to the password-only authentication model, its usability should be similar to password-only systems, therefore, we did not evaluate this option in our study.

Step 3–Completing the FK-PTR OPRF Protocol: In this step the device sends β to the client to complete its role in the OPRF.

C. Device-Client Channel

In our first implementation of the SPHINX system for Google Chrome browser, which was used in our lab-based study (described Section IV), we used the WebSocket protocol to establish communication between the device and the Chrome Extension. For the client to initiate the WebSocket communication by sending α, the device needs to be set-up as an HTTP server, the client being the HTTP client. We set up an HTTP server on the device using NanoHttpd Java application [9], adapted for Android.

In later implementation of SPHINX we decided to use Google Cloud Messaging (GCM) to provide a more stable connection between the device and the client. Note that since our application does not require the device-client channel to be secure, trusting on GCM would not at all affect the security of our approach.

We required a bi-directional GCM channel between the device and the client to run the FK-PTR OPRF protocol. That is, both the Chrome browser extension and the Android app should be able send and receive messages (in GCM terminology that is upstream and downstream messages). For upstream and downstream messaging, GCM requires an XMPP server and a client that is conventionally a smartphone or a Chrome extension.

We implemented the XMPP server on the Android device using Smack library. The application on the Android device creates an XMPP connection (using "Project ID" and "API Key" assigned to the GCM project on Google developer website) and listens to the receiving messages (i.e., α). The application can also send XMPP messages (i.e., β) to the client that is defined by the "Registration ID". To our knowledge, this is the first implementation of GCM that makes a bi-directional connection between two typically considered GCM clients (a mobile phone and a browser extension) without the need for any additional relaying server.

D. Performance Evaluation

The overall execution time of the SPHINX and performance of different major tasks on the Android device (LG G3 smartphone) and the client-side Chrome extension (on MacBook Air laptop with a 1.3 GHz Intel Core i5 processor and 4GB of memory) is evaluated over $10,000$ iterations, and the averaged results are reported in Table III. The total execution time for both the WebSocket and GCM implementation are shown. We excluded the time of human interaction with the system (manual pwd entry, and explicit consent on the device) from the evaluation. We also excluded authentication to the service, as this is the same in all schemes. Communication between C and D is timed for a 10Gbps WiFi Internet.

Based on our evaluation, for all parties, the most costly computation is Elliptic Curve exponentiation (71.00 ms on the extension, and 52.30 ms on the Android app with the mentioned libraries). The overall execution time of SPHINX protocol is around 500 ms and 400 ms, for WebSocket and GCM communication, respectively (excluding human interaction), which seems reasonably efficient. The total execution time including the interaction of human and the device are reported in lab-based usability study described in the following two sections.

IV. Usability Study Design

To analyze the effectiveness of the developed smartphone-based SPHINX system from the point of view of security (as perceived by the users), usability and adoption potential, we conducted a formal lab-based study, where participants logged into a popular web email service (Gmail) in a controlled observable environment with credentials (master password and device) provided to the participants. The design details and the flow of the study are presented in this section. The study was approved by our University's IRB. The participation in the study was voluntary, and standard ethical procedures were fully followed, e.g., participants being informed, given choice to discontinue, and not deceived.

A. Goals and Methodology

We first reasoned as to the optimal scheme in our study that should be used as a baseline for comparison with SPHINX. The candidate schemes we analyzed were: password-only, SPHINX in Zero-Interaction Mode, SPHINX in Explicit Consent Mode, and current device-based password manager. Table II shows the steps a user follows to authenticate by her username and password to a typical server in four different approaches. Compared to password-only systems, SPHINX in Zero-Interaction Mode does not impose any additional burden to the user, and SPHINX in Explicit Consent Mode requires the confirmation on the device. In contrast, current device-based password managers require the user to input master password on the device, and then copy the password from the device to the terminal manually. Just based on inspection of the user task flows, it is clear that current device-based password manager will have much lower usability compared to the other three approaches, and hence we ruled it out as the candidate for comparison. SPHINX in Zero-Interaction Mode also seems very close to password-only in terms of usability, so we do not use these schemes for comparison purposes either. We were left

TABLE II: Users' Task Flow in Different Password Managers (usability level seems to *decrease* from *left to right*). Following the inspection of these flows, our study was designed to compare SPHINX Explicit Consent Mode with Password-Only, with the goal of identifying how close the former is to the latter in terms of user experience.

Password-Only	SPHINX Zero-Interaction Mode	SPHINX Explicit Consent Mode	Device-based Password managers
1. Enter username 2. Enter password	1. Enter username 2. Enter master password (on client) [Password is transferred to the webpage automatically]	1. Enter username 2. Enter master password (on client) 3. Click confirm on the device [Password is transferred to the webpage automatically]	1. Enter username 2. Enter master password (on device) 3. Read and enter the password from device to terminal

TABLE III: Performance Analysis of the Implemented SPHINX Protocol for NIST P-256 Curve and 128 Bit OPRF Key; $\star$ The total time excludes users' interaction with the system.

	Task	Delay
Device	Group Membership ($\alpha \in G$)	0.36 ms
	Scalar Multiplication ($\beta = \alpha^k$)	71.00 ms
Client	EC-Hash ($H'(\mathsf{pwd}\|domain)$)	2.23 ms
	Scalar Multiplication ($H'(\mathsf{pwd}\|domain)^\rho$)	54.23 ms
	Inverse ($1/\rho$)	0.23 ms
	SHA256 Hash (H)	0.07 ms
	$H(\mathsf{pwd}\|domain, \beta^{1/\rho})$	52.30 ms
Total Time$\star$	Websocket	510.00 ms
	GCM	400.54 ms

with SPHINX in Explicit Consent Mode and password-only scheme, which appeared to be two ideal schemes to compare via our usability study.

Thus, we designed our study to compare the usability of SPHINX (Explicit Consent Mode) with the usability of traditional password-only login as the baseline or control condition (i.e., login in the absence of SPHINX or any other password manager). Our hypotheses were that the SPHINX login would be near-transparent to the users, except of typing in a memorable password (i.e., almost as easy as the password login), and security perception of SPHINX would be stronger for the users (i.e., providing a better sense of security when using SPHINX compared to the password-only login). To test these hypotheses, our study aims to answer the following aspects:

- **Transparency:** How close is the SPHINX user experience to a password-only authentication (login with a plain password in the absence of SPHINX)?
- **Security and Trust:** How much do the users trust the system? Do they feel that the system might expose their password?
- **Necessity:** Would the users be willing to adopt the system in practice?
- **Portability:** Can users easily login from multiple terminals?
- **Efficiency:** What delay the system incurs in logging in the users?
- **Comfort and Learnability:** How comfortable are users with the system? Can users quickly learn to use the system?

B. Study Flow

We recruited 25 participants from the members of our university. A similar number of participants are well-established in usability study research of password managers and mobile-based security applications [24], [34], [39], [40], [45]. After a brief introduction about the study, the participants were navigated by an examiner to a computer desk, and were provided an Android phone that had SPHINX app installed and a laptop that had the SPHINX Chrome extension installed. They were asked to consider this laptop as their home/personal (primary) computer. The examiner supervised and observed the participants throughout the study, and took notes of their questions and their performance. Upon completion of the study, the examiner filled out a form about success and errors that occurred in each assigned task. To assure that all users received equal guidance, all information and the instructions were shown in QuestionPro online survey format. The participants could read the instructions related to each task and answer questions related to the task online. All answers were stored on QuestionPro server for further analysis. We also logged the execution time of the protocol, excluding the manual password entry and including the approval on the phone.

The study was composed of three phases: the pre-study questionnaire, the main study tasks, and the post-study questionnaire. Analysis of the participants' answers, error rates, and behaviors in the study helped us to: (1) demonstrate the usability of SPHINX, (2) compare the usability of SPHINX with the password-only authentication, and (3) investigate possible security issues arising from usability problems. The study took about 30 minutes for each user. To value participants' time and effort, we offered a $5 Starbucks gift card to each. Figure 3 provides the high-level flow of our study.

C. Pre-Study Phase

The quantitative/qualitative pre-study questions were grouped into three categories, displayed in three consecutive pages, as summarized below (the questions asked are included in Appendix A2):

Q1. Demographics: The participants were asked to fill out a demographic questionnaire. These questions polled for each participant's age, gender and education.

Q2. Technical Background: The participants were asked about their general computer and security skills, and technical background to uncover their initial attitude towards security.

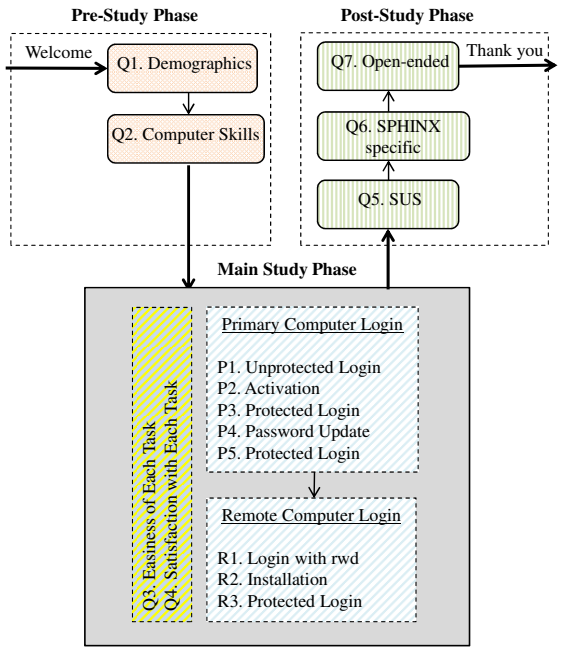

Fig. 3: Lab Study Design Flow

D. Main Study Phase

The main part of the experiment was an *experience-oriented study*. The study methodology is in line with that of prior studies of password managers [24], [34]. The participants were provided with a short description of SPHINX i.e., what it is, how it is activated, and how it is used. Then, they were given a username and a password to authenticate to Gmail (this service was chosen for its popularity among users), and a set of tasks to perform. Each task was shown in a web-page with an instruction followed by two multiple choice usability questions related to users' comfort in performing the task. After finishing each task and answering the two questions, users were instructed to move to the next available task.

1) Primary Computer Login: Following is the list of the first five tasks that was performed from the laptop provided to the participants (primary computer):

P1. Unprotected Login*:* As a baseline for our study, we asked the participants to login to Gmail using the given username and password (i.e., without using SPHINX). The username was the name of our research lab and the password was their name followed by a four digit number. We asked the participants to perform this task once. This task captures traditional password-only authentication.

P2. Activation*:* To evaluate the usability of the SPHINX registration process, we asked the users to update their password. The updated password was used as their master password in the next task. To activate SPHINX, the new password should start with "@@" or the user should click on the "F2" key before entering the password. After entering the password, user should accept a prompt displayed on the phone to assure that SPHINX is run only by the user who possesses the phone (and not by a remote attacker). We suggested them to set the extension to show the randomized password once. They could take a note of the randomized password in order to be able to login without SPHINX extension on a remote computer.

P3. Protected Login*:* To evaluate the usability of SPHINX login process, we asked the users to login to Gmail with their new password (established in the Activation task). After entering the master password, user accepts the prompted alert on the phone. Then the protocol continues and fills the password field automatically with the randomized password. The participants repeated this task 5 times.

P4. Password Update*:* To evaluate the usability of password update when using SPHINX, we asked the participants to update their SPHINX protected password to a new SPHINX protected password.

P5. Protected Re-Login*:* To gather more data, and also to provide participants with another chance to try SPHINX with a different master password (one established in the Password Update task), we asked them to follow the protected login process and login to Gmail with the updated password.

2) Remote Computer Login: After completing the primary computer login experiment, the participants were asked to move to another computer, which did not have the SPHINX plugin installed. They were told to consider this computer as their office or friend's computer, and instructed to perform the following tasks:

R1. Direct Login with Randomized Password*:* To evaluate usability of the system in case the SPHINX Chrome browser extension is unavailable or users' phone is unavailable (e.g., out of battery) users were asked to login to Gmail using their randomized password.

R2. Plugin Installation*:* An email was sent to the participants containing a link to the SPHINX browser extension. They were asked to download and install the extension on the remote computer.

R3. Protected Re-Login*:* Finally, after installing and configuring SPHINX on the remote computer, participants were asked to login using SPHINX (i.e., using their master password) as in the case of login from their primary computer.

To record participants' opinion about the system, each task above was followed by two questions that the participants had to answer on a 5-point Likert scale before moving to the next task.

Q3. Task Easiness*:* How easy it was to execute this task, Extremely easy, Very easy, Neither easy nor difficult, Very difficult, Extremely Difficult?

Q4. Task Satisfaction*:* How satisfied are you with SPHINX at performing this task, Extremely satisfied, Very satisfied, Neither satisfied nor dissatisfied, Very dissatisfied, Extremely dissatisfied?

E. Post-Study Phase

The post-study phase consists of the following set of questionnaires (for precise questions, see Appendix A2):

Q5. System Usability Scales*:* At the end of the experiment, in the first set of post-study questions, users were asked to fill

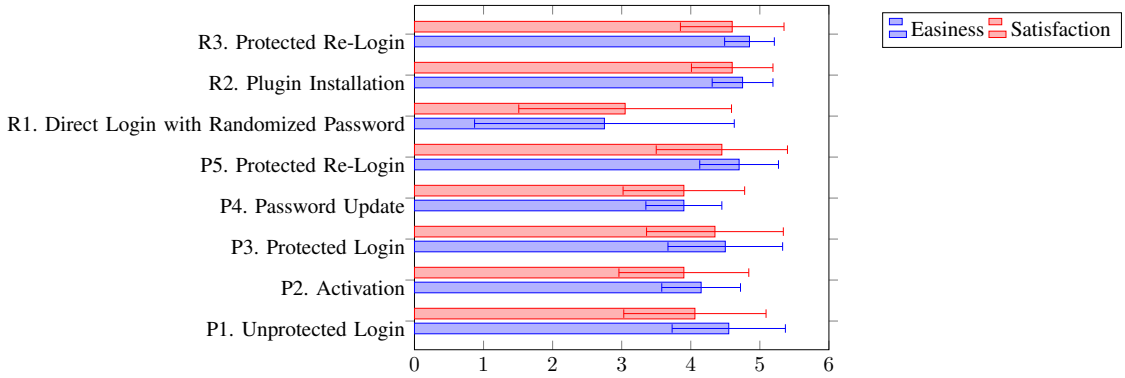

Fig. 4: Main Study Task Easiness & Satisfaction Scores (out of 5)

out the System Usability Scale (SUS) questionnaire [22] for password-only systems, as well as for SPHINX. SUS has been designed to measure the usability of an engineered system with respect to effectiveness and efficiency, and users' experience and satisfaction.

Q6. SPHINX Specific Usability Questions: In the second questionnaire, we asked more specific questions about SPHINX to figure out how users' experience was in comparison to password only systems, and how confident they were about system's security and reliability. This questionnaire included: two "Transparency", five "Security and Trust", two "Necessity", and one "Portability" questions.

Q7. Open Ended Questions: The study concluded with six open-ended questions (Appendix A2).

V. Study Results and Analysis

A. Pre-Study Analysis

The 25 participants were from the age groups of 18-24 years (40%) and 25-34 years (60%), with equal number of undergraduate and graduate students from *diverse educational background*, including: education, engineering, health care, arts, science, and in particular, *none of them* was specialized in computer security. 20% of the participants were female and 80% were male. They ranked their general computer background as: Poor (20%), Average (40%), and Excellent (40%), and their general computer security skills as Poor (40%), and Average (60%). Our participant samples are in line with those reported in prior password manager studies [24], [34], [39], [40].

In our pre-study questionnaire, we posed participants with several questions about their technical background, habits of using password managers and choice of password. Most participants stated that they visit web-pages that require authentication several times a day. 25% of them said they do not login to their Gmail account on daily basis, 50% log in "once a day" and the rest of them login "several times a day". Only one participant said he/she uses a password manager, around 30% said they do not use any password manager, and the rest rely on the browser or the cookies of the visited web-site to remember their password for "fast and easy login process". Among the password choices that we questioned (easy to remember, difficult for others to guess, strong randomized password, similar over multiple accounts), 50% of the participants said they choose passwords that are easy to remember, all participants said they pick password that are difficult for others to guess, and 67% said they reuse the same password over multiple accounts, and none of the users had confidence that their passwords are random or strong. These numbers serve to re-confirm the need for a tool like SPHINX that can strengthen users' passwords.

B. Main Study Analysis

Answer to the two questions at the end of each task (P1-P5 and R1-R4, as described in Section IV-D) polls for easiness of each task and satisfaction of the participants with the system. Figure 4 presents the Likert scores corresponding to each task for the first question (easiness) and the second question (satisfaction). Unprotected Login, which is the baseline for the rest of the study, shows how easy users found SPHINX-free login to Gmail. Based on the results, and as expected, participants found login to Gmail (with the test username and password that was provided to them) very easy. No error occurred in performing this task by the participants.

The next task was Activation, in which the participant had to update his/her passwords to activate SPHINXThe results show that SPHINX performed well in this task, and users found this task very easy and were very satisfied with SPHINX. The Password Update task is, by nature, very similar to activation (the user needs to update the password). Therefore, we averaged the error rates in performing these two similar tasks and found the error rate to be around thirty percent. The main issue of user failures during activation was forgetting to enter the predefined prefix, which leads to failure in calling SPHINX from the extension side. Other issues were entering an incorrect password and forgetting to accept the alert (the consent message) shown on the phone. We believe this result is mainly because of the unfamiliarity of the users with the new system. These results *do not* degrade the security of the system in any way, since the attacker still needs to have access to the device to login even if pwd is exposed. In a practical setting it is sensible to assume that users would remember to enter the prefix as part of their password (with first two publicly known characters). The need for entering the prefix can be avoided by enabling SPHINX on services by default.

Password update in SPHINX, just like other hash-based

managers, is essential to update the currently used, possibly weak, password to a randomized one. However, we note that users do not frequently update their passwords so lower score could still be acceptable.

The averaged users' ranking over multiple "Protected Logins" (i.e., first, second and third protected logins, as described in Section IV-D) is 4.70. The high average score of 4.70 shows that users found login with SPHINX easy. This score is comparable to the unprotected login score (4.55) in which the user logged in to Gmail using a username and a memorable password. We further compared the score of unprotected login and the average score of protected login using *Wilcoxon Signed Rank Test* but the result was not statistically significant at $p < 0.05$. The average error rate committed by the participants in performing this task was eleven percent and the main cause of failure was forgetting to enter the predefined prefix especially in the first attempts, which is perhaps due to unfamiliarity with the system. It can be observed in Figure 4 (P3, P5, and R3 – protected login) that, as the users were repeating the task and becoming more familiar with the system, they scores slightly improved and they had less error. However, the result of the *Friedman Test* did not show any statistical significance when we compared the scores.

The next task was to log in directly with the strong SPHINX generated password. This scenario might happen when the user cannot run the protocol (e.g., due to a missing browser plugin). A similar situation with a password only system happens when the user picks a strong randomized password. As anticipated, the users did not find this task very easy (score of 2.75) and were not as satisfied as they were with in other tasks. Average error rate in performing this task was also understandably high, around 45%. In this task, in addition to Likert score answer options, we included an extra answer option for those who did not remember the password or did not take or have a note of it. 40% of users picked this option in the answers, and said in real-life it would be very probable that they would not remember a strong randomized password. This answer in fact shows an advantage of SPHINX: in general users avoid strong passwords as was shown in the pre-study questions, therefore a system like SPHINX that can generate a strong password based on a memorable password would be beneficial to many users.

The final task was the installation of the SPHINX plugin. Based on the study it seems the installation task was extremely easy and users were extremely satisfied with the system in performing it.

C. Post-Study Analysis

Here, we summarize the results from the post study questionnaires. The score of "Transparency" was 3.55 (std dev = 0.87), which is in line with the main study and shows that most of the users found their experience working with SPHINX similar to the password-only mechanism. The score of "Security and Trust" was 4.13 (std dev = 0.97), which shows that users perception of security of SPHINX was high. The users generally agreed that SPHINX helps them to secure their system, they could trust SPHINX to harden their password, and they do not have any issue with not knowing and not remembering their strong password. This insight suggests that users, not only found SPHINX to be easy to use but also perceived it to be secure. This is an important result for SPHINX that may push it towards real-world deployment.

The "Necessity" score was 3.32 (std dev = 1.12) which suggests that users are more in agreement (than disagreement) that they will need SPHINX to protect their account. Finally, the "Portability" score was 3.75 (std dev = 1.25) which shows that logging from a remote computer would be generally convenient. This result is also in line with our main study where users found login from a remote computer easy and did not commit errors setting up the system.

SUS Score: The average SUS score for SPHINX reported by our study was 79.4 (std dev = 8.9). Considering that industry averages for SUS scores tend to hover in the 60 to 70 range [36], results show that users found the systems to be much more usable than the average. Users scored SUS of a password-only authentication 77.5 (std dev = 11.9), which compered to SPHINX does not show statistical significance using Wilcoxon signed-rank test. This is a promising result demonstrating the overall attractive usability of SPHINX.

Efficiency Results: In our experiments, we timed the SPHINX login protocol while users were performing the protected login tasks. The start point was the time the password was entered and the end point was the time the randomized password was received by the browser extension (inclusive of the time for user explicit consent). The averaged execution time was 1968.00ms with an standard deviation of 579.76ms.

Qualitative Feedback: In answers to the open-ended questionnaire, almost all users correctly described SPHINX as a system that helps them to—more securely and more easily—authenticate to web services, and many of them said they would be willing to use SPHINX in daily life especially to protect their accounts with financial/banking sites and email providers. Some of the answers are quoted below:

1) From your understanding, what does SPHINX do? *"StoPS helps me to remember easier passwords while in reality it provides strong passwords to the web sites."*
2) Did you face any problem/issue/difficulty when using SPHINX?*"Not much, just a little getting-used-to."*
3) Do you have any suggestions for SPHINX that can make it more useful or easier to use? *"Perhaps some extra documentation or instructions present in the tool can be very handy."*
4) Do you think SPHINX is better than other password managers? *"It is easier to use the browser password storage, but SPHINX seems to be more secure"; "I do not have any experience with other password managers but it seems to be great!"*
5) Will you be willing to use SPHINX in your day-to-day use? *"absolutely if it is free"; "Most definitely"*
6) Which types of sites you will be interested in using SPHINX? *"Banking, emails, ..."*, *"Any and all", "Every place that includes a password."*

VI. Discussion, Limitation & Future Directions

Online SPHINX Service: Our current implementation of SPHINX is geared for smartphones. However, it is by no

means limited to that. Since SPHINX does not require a secure channel between D and C and since the device is oblivious to the user's password (pwd or rwd), one possibility is to outsource the device functionality to a remote online (third-party) service, to give rise to an online manager (Online SPHINX). For such an online setting, in contrast to other online password managers, SPHINX would provide optimal resistance in the event of service/manager compromise since only OPRF key is stored on the online service and is independent of user's password (in particular, this password is not needed for authenticating the user to the online SPHINX).

The online SPHINX service can act as a full replacement of the smartphone as an independent password manager. Online SPHINX can also serve as the password manager to allow for login from the smartphone itself, which in our current setting (smartphone instantiation) is not supported. Finally, the online service can also be a complement to the smartphone instantiation for backup purposes (see next discussion item).

Since SPHINX does not require a secure/authenticated channel between D and C, the online SPHINX can use any public (unprotected) channel, such as the Internet (e.g., even without TLS and PKI).

Online SPHINX can also benefit from a distributed service in which the OPRF keys for web accounts is distributed among several servers. This prevents learning specific keys upon the compromise of one server or a subset of servers (with a threshold scheme). It also provides better "availability" guarantees. OPRF computation in SPHINX can be thersholdized using polynomial secret sharing.

Further work is warranted to carefully design an online version of SPHINX, which can retain all the security and usability advantages of SPHINX while providing increased service availability. Our assumption is that such setting will provide the same security features as SPHINX in Zero Interaction mode, while it may provide better usability since the reliance on the smartphone would not be needed. In a real-world system, both online and smartphone implementations can be offered to the users as a holistic solution, similar to many currently-deployed commercial password management services.

Key Back-Up and Device Upgrade: One natural concern about any smartphone password manager, like the smartphone-based instantiation of SPHINX, is the permanent loss of the phone. To deal with such situations, SPHINX users should back-up the OPRF key on an external storage. When using a new device, this key can then be recovered from the back-up device. Similarly, when upgrading to a new device (e.g., when buying a new phone), the key from the old phone or the back-up device can be copied over to the new device. Such a key transfer should be performed over a secure channel (e.g., a wired connection between the devices). The current device-based password managers also recommend backing up the list of stored passwords on the device in the same way. Even the current two-factor authentication (2FA) systems recommend similar strategies. SPHINXonline service could be used to facilitate such a back-up (as discussed above).

Alternative Devices: Other types of smart wearable devices, such as smart watches and glasses, are additional viable platforms on which SPHINX can be instantiated (Wearable SPHINX). "Security keys", such as the FIDO Universal second factor USB devices, are yet another alternative (USB SPHINX). Login from smartphone with SPHINX will be possible if the wearable is used as the manager. Future SPHINX prototypes might be developed on such alternatives.

Client Compromise: Password managers are not intended, designed or capable of resisting against client side compromise. Indeed, none of the password managers that we studied are secure against client compromise. The reason is that the password is entered in the webpage manually by the user or automatically by the password manager. In either case, an attacker who resides on a client can theoretically intercept the password. Malicious code and key-loggers are always a threat to browsers in spite of security enhancements in the browsers. However, in our SPHINX system, because we use a "key-ed" password hardening scheme, an attacker who learns pwd using a key-logger can *not* succeed in logging into the web service. An attacker, who compromises the client machine and get it to execute a malicious code, can obtain rwd of an on-going session, and thereby succeed in logging to *only* that service, even if the user uses the same pwd for all services.

One potential solution to offer security under client compromise could be to employ a 2FA mechanism. Since 2FA mechanisms require a password as well as a one-time PIN code produced by the device (the second factor), they can offer better resistance in the event of client compromise (key-logging one PIN code from the client machine will not be sufficient for the attacker to log in over *new* sessions). Since 2FA and SPHINX both use a device during the authentication process, it seems natural to integrate the two schemes together so as to achieve all the security advantages provided by the latter and the resilience to client compromise offered by the former. Such integration may work transparently with current web services that already use 2FA as a means to login users. However, a thorough future investigation is necessary to formalize and realize such SPHINX-enhanced 2FA mechanisms.

Limitation, and Further User Studies: We note that while our reported usability study of the smartphone-based SPHINX system serve to demonstrate its promising feasibility (as is customary in usable security research), further studies may need to be conducted with larger and more diverse set of participant samples, possibly in field settings. We are pursuing this line of research in our ongoing work.

VII. SPHINX vs. Other Managers

We analyze SPHINX with respect to several security, usability and deployability metrics, and compare it with many other managers. Our analysis is summarized in Table IV.

The listed metrics for comparison are built upon the evaluation framework of [21], which was designed to assess web-based authentication schemes. We base our comparison upon a similar set of metrics but refine and extend the list further to meet the characteristics of password managers. This refined list may be independently used as a specialized framework to evaluate password managers.

The list of studied schemes is comprehensive and includes three main categories of password managers: (1) *Hash-based*

TABLE IV: SPHINX vs. Other Password Managers. "ODA-Resistance" denotes resistance to offline dictionary attacks. Other metrics are drawn from [21].

	SECURITY								USABILITY				DEPLOY.	
	S1. Unique-Password-Enforcer	**S2. Resilient-to-Phishing**	**S3. ODA-Resistant-Server-Compromise**	**S4. Storeless**	**S5. ODA-Resistant-Device-Compromise**	**S6. Resilient-Upon-Theft**	**S7. Resistant-to-MITM-D-to-C**	**S8. Resilient-to-Physical-Observation**	**U1. Memorywise-Effortless**	**U2. Password-Update-Not-Necessary**	**U3. Scalable-for-Users**	**U4. Physically-Effortless**	**D1. Client-Compatible**	**D2. Nothing-to-Carry**
PM0. Password-Only	n	n	n	y	n	y	-	n	y	y	n	y	y	y
PM1. SPHINX														
a) Smartphone Explicit Consent	y±	y	y	y	y	y	y	y	y	n	y	n	n	n
b) Smartphone Zero Interaction	y±	y	y	y	y	y	y	n	y	n	y	y	n	n
c) Online	y±	y	y	y	y	y	y	n	y	n	y	y	n	y
PM2. PwdHash	y	n⊙	n◇	y	n	y	-	n	y	n	y	y	n	y
PM3. Password Multiplier	y	n⊙	n◇	y	n	y	-	n	y	n	y	y	n	y
PM4. Passpet	y	n⊙	n◇	y	n	y	-	n	y	n	y	y	n	y
PM5. Firefox	n	n	n∗	n	n	n	-	n	y	y	y	y	y	y
PM6.1. Client-based Managers	n	n	n∗	n	n	n	-	n	y	y	y	n	n	y
PM6.2. Token-based Managers	n	n	n∗	n	n	n	y†	n	y	y	y	n	n	n
PM6.3. Online Managers	n	n	n∗	n	n	n	y‡	n	y	y	y	n	y	y
PM7. Tapas	n	y	n∗	n	n	n	y●	y	y	y	y	y	n	n

Notes:
± Enhanced by the use of two uniqueness parameters, domain name and OPRF key.
⊙ They provide higher resistance compared to password-only, but still an ODA is possible after the phishing attack.
◇ Unless the user chooses a randomized master password.
∗ Unless the user chooses a randomized password for each account.
† Establishment of confidential channel necessary.
‡ Establishment of confidential and authenticated channel necessary.
● Establishment of confidential and authenticated channel necessary.

password managers, in which the passwords are generated by applying a cryptographic function to the master password and a tag (e.g., the visited website's domain name). In addition to SPHINX, we included, in the comparison, three well-recognized academic works, namely, PwdHash [42], Password-Multiplier [30], and Passpet [48]; (2) *Traditional password managers*, that are usually available as browser extensions, desktop applications, token-based application (e.g., USB key or a smartphone apps), and online services. We included highly ranked or widely used commercial password managers in the list, including Firefox [11], LastPass [12], 1Password [1], Dashlane [5], and RoboForm [13]. We also included an instance of a smartphone password manager, Tapas [39], that is similar in functionality to traditional password managers, but unlike others, runs a protocol between the device and the terminal to encrypt, store and retrieve the password. In this study, we exclude approaches that require service-side changes (e.g., Phool Proof [35], MP-Auth [38], and Pico [43]).

SPHINX is different from traditional password managers in many ways. Traditional password managers, such as [1], [12], store the passwords encrypted with a single master password. In particular, with existing device-based managers, the user types in her master password on the application that unlocks the respective password and displays it on the screen. The user then copies the password over to the login-page to authenticate to the service. These managers require a confidential channel (e.g., resistant to shoulder-surfing) between the password manager app and the client, and are also open to dictionary attacks upon application compromise. In addition, an attacker could guess the master password in an online guessing attack against the device. Similarly a potentially malicious code running on the device could learn the user's master password as it is entered into the device and learn all stored passwords. Such password managers normally do not provide resistance against offline dictionary attack in the event of server compromise unless the user picks a random password for each site (i.e., the manager does not enforce a policy). Resistance against phishing is not provided by these managers.

In contrast, SPHINX does not require a secure channel, and *nothing* (in an information-theoretic sense) is learned about the password by the device (or a malicious software running on the device) or over the device-client channel, even without any protection of this channel. Finally, by randomizing and hardening the passwords, SPHINX imposes resistance against server-side offline dictionary attack.

SPHINX is also different from "hash-based password managers" that compute a cryptographically hash of a memorable password on the fly. An example of such password manager solutions is PwdHash [42]. PwdHash maps a low entropy password to a randomized one by hashing a (password, domain-name) pair and registering it as the password with the server.

PwdHash deterministically transforms a user's password into a more complex password but, unlike our scheme, this transformation does not help against offline dictionary attacks at a compromised server. Moreover, if a user uses the same master password pwd with PwdHash for different services, the compromise of a single server leads to the discovery of pwd via an offline attack and then to the (deterministic) calculation of all rwd's derived from pwd.

The authors of [42] mentioned the possibility of using a "short secret salt" or a "global password" that acts as a form of a PRF key, but this would be a weak key (either short key, or a password) which will be disclosed to the client machines In contrast, in our case, the device's key is never exposed outside of the device. Similar to PwdHash, in SPHINX resistance to phishing attacks is achieved by concatenating the domain name of the website to pwd, in computation of rwd; However, in our case the security implications are stronger: In PwdHash the attacker who obtains the randomized password through phishing can mount a dictionary attack to find the user's password, while in our case this is not feasible.

The security against online guessing attacks, provided by smartphone-based SPHINX is equivalent to rwd, whereas, other smartphone managers may or may not provide the same level of security depending on whether the user chooses a randomized password or not. Security level against online attack for online SPHINX is equivalent to pwd, same as other online managers.

In summary, SPHINX provides *many unique security features* that are not offered by any other password managers. These include: compulsory resistance against offline dictionary attack through non-deterministic (secret-keyed) password randomization, security against password reuse by generating a unique randomized password for each service, security against manager (device) compromise (physical or remote), security against MITM attack and eavesdropping against client-device channel with no security requirement for this channel. At the same time, SPHINX maintains many security and usability features that can be found in many other password managers. These include: resistance against online guessing attack, resistance against phishing attacks, compatibility with different services and clients, and being memory-wise effortless.

VIII. Conclusions

Passwords are a "necessary evil". In this paper, we attempted to respond to the growing security and usability problems with passwords by proposing SPHINX, a cryptographic password manager that can address most security and usability problems with passwords from the client/user side alone (i.e., transparent to most existing web authentication services). SPHINX is a password management approach, built atop an existing oblivious PRF (OPRF) scheme, that transforms a human-memorable password into a random password with the aid of a device without the need to store the passwords on the device. SPHINX offers several key security guarantees, namely, resistance to: (1) online guessing attacks, (2) offline dictionary attacks under server compromise, (3) offline dictionary attacks under device compromise, (4) phishing attacks, and (5) eavesdropping and man-in-the-middle attacks on the device-client channel. SPHINX also boasts to provide almost the same level of user experience as that of authentication using an easy to memorize password. Unlike other password managers, SPHINX perfectly hides passwords and the master password from itself, and thus remains secure under the realistic threat of the compromise of password managers. Also, unlike other password managers, SPHINX does not require a secure device-client channel. At the same time and like many other password managers, SPHINX can resist online guessing, offline dictionary under web service compromise and phishing attacks. We designed and implemented a smartphone-based instantiation of SPHINX. Our performance and usability evaluation of this instantiation shows that it is efficient, relatively simple to use and perceived to be secure and trustworthy by the users.

References

[1] 1Password: Simple, Convenient Security. https://1password.com/.

[2] Anonymous hackers claim to leak 28,000 PayPal passwords on global protest day. Available at: http://goo.gl/oPv2h.

[3] Blizzard servers hacked; emails, hashed passwords stolen. Available at: http://goo.gl/OTNWJC.

[4] Current Android Malware . Available at: http://goo.gl/0sWbXz.

[5] Dashlane Password Manager. https://www.dashlane.com/.

[6] Fine the source of your leaks. Available at: https://www.leakedsource.com/.

[7] Hackers compromised nearly 5M Gmail passwords. Available at: http://goo.gl/IRu07u.

[8] LinkedIn Confirms Account Passwords Hacked. Available at: http://goo.gl/AWB5KC.

[9] Nanohttpd java app. https://nanohttpd.com.

[10] One Year Of Android Malware. Available at: http://goo.gl/2UkUJS.

[11] Password Manager - Remember, delete, change and import saved passwords in Firefox. https://goo.gl/Qve4l7.

[12] Password Manager, Auto Form Filler, Random Password Generator & Secure Digital Wallet App. https://lastpass.com/.

[13] RoboForm: World's Best Password Manager. https://www.roboform.com/.

[14] RSA breach leaks data for hacking securid tokens. Available at: http://goo.gl/tcEoS.

[15] RSA SecurID software token cloning: a new how-to. Available at: http://goo.gl/qkSFY.

[16] Russian Hackers Amass Over a Billion Internet Passwords. Available at: http://goo.gl/aXzqj8.

[17] A. Adams and M. A. Sasse. Users are not the enemy. *Commun. ACM*, 42(12), 1999.

[18] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology – Eurocrypt*, 2000.

[19] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. 2013.

[20] J. Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012.

[21] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Security and Privacy*, 2012.

[22] J. Brooke. SUS: a "quick and dirty" usability scale. In P. W. Jordan, B. Thomas, B. A. Weerdmeester, and A. L. McClelland, editors, *Usability Evaluation in Industry*. Taylor and Francis, London, 1996.

[23] R. Chatterjee, J. Bonneau, A. Juels, and T. Ristenpart. Cracking-resistant password vaults using natural language encoders. In *2015 IEEE Symposium on Security and Privacy*, pages 481–498. IEEE, 2015.

[24] S. Chiasson, P. C. van Oorschot, and R. Biddle. A usability study and critique of two password managers. In *Usenix Security*, 2006.

[25] I. Dacosta, M. Ahamad, and P. Traynor. Trust no one else: Detecting MITM attacks against SSL/TLS without third-parties. In *European Symposium on Research in Computer Security*, 2012.

[26] W. Ford and B. S. Kaliski Jr. Server-assisted generation of a strong secret from a password. In *Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2000.

[27] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography*. 2005.

[28] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: Validating ssl certificates in non-browser software. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2012.

[29] M. Gollam, B. Beuscher, and M. Drmuth. On the security of cracking-resistant password vaults. In *ACM Conference on Computer and Communications Security*, to appear 2016.

[30] J. A. Halderman, B. Waters, and E. W. Felten. A convenient method for securely managing passwords. In *Proceedings of the 14th international conference on World Wide Web*. ACM, 2005.

[31] S. Jarecki, A. Kiayias, and H. Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In *Advances in Cryptology–ASIACRYPT*. 2014.

[32] S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena. Device-Enhanced Password Protocols with Optimal Online-Offline Protection. Technical report available at: http://eprint.iacr.org/2015/1099. To appear at ASIACCS 2016.

[33] A. Karole, N. Saxena, and N. Christin. A Comparative Usability Evaluation of Traditional Password Managers. In *International Conference on Information Security and Cryptology (ICISC)*, December 2010.

[34] A. Karole, N. Saxena, and N. Christin. A comparative usability evaluation of traditional password managers. In *Information Security and Cryptology-ICISC*. 2011.

[35] M. R. Karthiga and M. K. Aravindhan. Enhancing performance of user authentication protocol with resist to password reuse attacks. *International Journal Of Computational Engineering Research*, 2(8), 2012.

[36] J. R. Lewis. Ibm computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *Int. J. Hum.-Comput. Interact.*, 7(1), 1995.

[37] Z. Li, W. He, D. Akhawe, and D. Song. The emperor's new password manager: Security analysis of web-based password managers. In *USENIX Security Symposium*, 2014.

[38] M. Mannan and P. C. van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. In *Financial Cryptography and Data Security*. Springer, 2007.

[39] D. McCarney, D. Barrera, J. Clark, S. Chiasson, and P. C. van Oorschot. Tapas: design, implementation, and usability evaluation of a password manager. In *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 2012.

[40] S. E. McGregor, P. Charters, T. Holliday, and F. Roesner. Investigating the computer security practices and needs of journalists. In *24th USENIX Security Symposium (USENIX Security 15)*, 2015.

[41] R. Morris and K. Thompson. Password security: a case history. *Commun. ACM*, 22(11), 1979.

[42] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. In *USENIX Security Symposium*, 2005.

[43] F. Stajano. Pico: No more passwords! In *Security Protocols XIX*, pages 49–81. Springer, 2011.

[44] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying wolf: An empirical study of ssl warning effectiveness. In *USENIX Security Symposium*, 2009.

[45] E. Uzun, K. Karvonen, and N. Asokan. Usability analysis of secure pairing methods. In *Financial Cryptography and Data Security*. 2007.

[46] L. Whitney. LastPass CEO reveals details on security breach. CNET: http://www.cnet.com/news/lastpass-ceo-reveals-details-on-security-breach/, 2011.

[47] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password memorability and security: Empirical results. *IEEE Security and Privacy*, 2(5), 2004.

[48] K.-P. Yee and K. Sitaker. Passpet: convenient password management and phishing protection. In *Symposium on Usable privacy and security*, 2006.

[49] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *Security and Privacy (SP)*, 2012.

APPENDIX

A. Comparison of SPHINX with Other Password Managers

1) Analyzed Schemes: We refer to the existing text-based password authentication web services as **password-only** (**PM0**) ("without password manager (PM)"). This is the traditional model of authentication where the user logs in with a human-memorable password. All the other schemes listed below are compared with this model. Any scheme that can provide similar level of usability as password-only mechanism while offering additional security features is preferable.

Hash Based Password Managers

PM1. SPHINX: This refers to our proposed scheme. In our analysis, we consider the primary design of SPHINX in the explicit consent mode, and we will adjust any changes in each of the metrics in case of the zero-interaction mode.

PM2. PwdHash: This refers to a mechanism in which the browser extension computes a deterministic (i.e., unkeyed) hash of input pwd and the domain name of the visited page to construct a randomized password. The domain name is used for resistance against phishing attacks. Although the user can assign different tags to each service (rather than the domain name), in real-life, users will typically choose a weak predictable tag.

PM3. Password Multiplier: Similar to PwdHash, Password Multiplier is a browser extension that applies an unkeyed cryptographic hash function of the master username, master password and the target website domain to generate a randomized password for a given user account. It offers resistance to phishing attacks in the same way as SPHINX and PwdHash.

PM4. Passpet: Similar to PwdHash and Password Multiplier, Passpet computes an unkeyed hash of a password to construct a randomized password that will be registered with the service. To provide phishing resistance, Passpet incorporates domain name of the target website in hash function computation (the same way as SPHINX and PwdHash).

Note that of all the above methods, only SPHINX uses a secret keyed function to compute the randomized password.

Traditional Password Managers

PM5. Firefox Password Manager: Firefox Password Manager stores the usernames and the passwords in the terminal storage, and automatically fills the login form when the users visits a website. Firefox Password Manager can work with or without a master password (i.e., with or without encrypting

the stored passwords). In our analysis, we consider the version with master password (since this is the fair setting to compare with SPHINX and other password managers). Although we pick Firefox as an instance of desktop password managers, other browsers also offer similar built-in password managers and our comparative analysis applies to them exactly the same way.

PM6. Commercial Password Managers: This category of password managers can further be divided into: *PM6.1. Client-based* (e.g., browser plugin), *PM6.2. Token-based* (e.g., a smartphone app), and *PM6.3. Web-based* (e.g., cloudbased service) commercial password managers, that are often different in the offered security or usability properties. Lastpass, 1Password, Dashlane, and Roboform are instances of commercial cross-browser, cross-platform password managers that store the passwords on some computational device (client/token or web server) and lock them using a master password. They offer several other features such as account syncing, password generation, and form auto fill (elements that are orthogonal to the features of SPHINX and can be combined with it).

PM7. Tapas: Tapas is a device-based password manager that suggests a *dual-possession authentication*, leveraging a desktop computer and a smartphone, which runs a protocol between the browser extension (Manager) and the smartphone (Wallet) over a "secure channel" to store the encrypted password on the Wallet and retrieve it when the user visits the webpage. Tapas maintains security of the managed passwords by encrypting and storing the passwords on a smartphone, and keeping the decryption key inside the browser on the paired computer. The dual-possession feature of Tapas helps to protect the user-chosen password in case of offline attacks against the device (unlike other traditional token-based managers). SPHINX achieves this property without the need to share a key, or establish a secure channel, between client and device.

2) Metrics and Comparative Analysis: ***Security Metrics***

S1. Unique-Password-Enforcer (captured in S6, Resilient-to-Leaks-from-Other-Verifiers, in [21]): SPHINX, PwdHash, Password Multiplier, and Passpet uniquely select a randomized password for each website. All these mechanisms incorporate domain name of the target website (which is assumed to be unique) in their cryptographic calculation. SPHINX being a *keyed mechanism* assigns different keys to each service, and, even without incorporating the domain, generates a different/unique secure password. In the other password managers, it is up to the user to select different passwords for each website. Many of the users prefer to reuse their password over different services (a known user attribute that was confirmed via our study), which is a major security issue in password authentication services.

S2. Resilient-to-Phishing (S7 in [21]): As mentioned earlier, SPHINX, PwdHash, Password Multiplier, and Passpet, incorporate domain name of the webpage in their calculation (precisely in a hash function), and therefore are resistant to phishing attacks. However, unlike SPHINX, other schemes are susceptible to offline dictionary attacks as the phisher can recover the master password, given the hashed password corresponding to the phisher's domain. Tapas provides phishing protection by ensuring that passwords are submitted to the exact site (determined by the site's URL and SSL/TLS certificate) they were registered with. Other password managers that we are studying do not provide resistance against phishing.

S3. Offline-Dict-Attack-Resistant-Server-Compromise (captured in S5, Resilient-to-Internal-Observation, in [21]): This feature is *unique* to SPHINX and no other password manager offers security against offline dictionary attacks upon service compromise (unless the user deliberately chooses a random password to register with the service). SPHINX cryptographically randomizes the memorable password, but unlike other hash-based password managers, the cryptographic function is not a deterministic hash function. The cryptographic function is an Oblivious-PRF function in which the browser extension inputs the password (and the domain name for phishing protection) and the device inputs a secret key to reconstruct the password. Therefore, the password that is registered with the service is cryptographically random and attacker cannot perform a dictionary attack against the server. Although, PwdHash, Password Multiplier, and Passpet generate a randomized password, the password is derived from a deterministic hash function on input of a master password (and a predictable domain name), that is susceptible to an offline attack against the master password, upon learning the (hash of) password hash stored on the server.

S4. Storeless: We refer to mechanisms that do not store the password encrypted or unencrypted as storeless. Except for the hash-based password managers in our list (i.e. SPHINX, PwdHash, Password Multiplier, and Passpet) that compute the password on the fly, all other mechanism store the password encrypted or unencrypted (on the token, on the client machine, or on an online server). Passpet is the only hash-based password manager that stores domain names, which might expose the privacy of the user, but this does not affect its security.

S5. Offline-Dict-Resistant-Device-Compromise (captured in S5, Resilient-to-Internal-Observation, in [21]): This property implies that an attacker cannot intercept the user's input from inside the user's device (e.g., by a malware).

Tapas as well as token-based (and web-based) password managers that store the password encrypted might mistakenly appear to be secure in case of the device (or the password manager server, in case of online password managers) get compromised. However, note that once the device gets compromised, the security falls back to the security of the master/nominal password. In that case, commercial password managers that require one single master password to protect all accounts, offer lower level of security (the attacker can compromise "all" the accounts that are served by the password manager). In contrast in SPHINX no information about the pwd and rwd is leaked from the device, since pwd is not entered into the device and rwd is not stored on the device.

S6. Resilient-Upon-Theft (S8 in [21]): This property applies to token/device-based managers and implies that an attacker who can get physical access to the device, temporarily or permanently cannot obtain the passwords. The security arguments under this property are exactly the same as the previous item.

S7. Resistance-to-MITM over device-client channel (captured in S5, Resilient-to-Internal-Observation, in [21]): This

metric applies to all token-based and web-based password managers. SPHINX is the only password manager does not require any requirement on device-client channel (i.e, secure and authenticated channel between the device and the client is not required) and therefore is secure (in the strongest information-theoretic sense) against MITM attack and eavesdropping. Unlike SPHINX, Tapas creates an authenticated channel between the Wallet (device) and the Manager (browser) (via pairing, for example), and then only it can provide the required security against MITM. Traditional web-based password managers protect against MITM attack only through the use of SSL channels (which might be compromised) between the web-based password manager and the client. In traditional token-based password managers the users are required to manually read and enter the password from the device, the manual-human based channel is typically considered secure against the MITM attack, however, physical observation over this channel is possible and is captured in the next item (S9).

S8. Resilient-to-Physical-Observation (S1 in [21]): This property implies that the attacker cannot impersonate a user after observing him/her during the authentication process earlier. These attacks include shoulder surfing, filming the keyboard, recording keystroke sounds, or thermal imaging of keypad. All of the listed commercial password managers require the user to enter the master password and therefore are not resilient to physical observation. Although Passpet, PwdHash and Password Multiplier fill in the password field with the randomized password, they still require the user to enter the master secret. SPHINX, however, is secure to physical observation, since our primary design requires the user to approve a dialog on the device before reconstructing password and filling the password field. To improve the usability of the SPHINX system, the user can optionally disable the need for approval (zero interaction), however, in that case SPHINX would not be Resilient-to-Physical-Observation.

Usability Metrics

U1. Memorywise-Effortless (U1 in [21]): We call a system Memorywise-Effortless if the user is not required to remember any secrets except for a password or a master password per service. With our definition, all the studied managers are Memorywise-Effortless.

U2. No-Initial-Password-Update: SPHINX and all the hash-based managers in our list (PwdHash, Password Multiplier, and Passpet) compute a cryptographic function of the password that the user enters to the web-sites. Hence, they require the user to update their original password when moving over to one of these schemes from a password-only scheme. Although, in SPHINX usability study users reported that this step is easy to perform, all other studied password managers work without changing the master password.

U3. Scalable-for-Users (U2 in [21]): The schemes that do not require extra effort from the users' side when the number of accounts are increased are considered as scalable. Scalability is defined from the users point of view and not from the system deployment perspective. With this definition, all the studied password manager are scalable, since users' effort does not increase with increase in the number of accounts.

U4. Physically-Effortless (U4 in [21]): The schemes that do not require user to do anything beyond typing one password, and perhaps clicking a button to activate or run the password manager service are "Physically-Effortless". SPHINX requires the user to tap the approve button on the device, which is a design choice for extra security. As we discussed in the real-life study, SPHINX can work as a service on the phone and would not necessarily require this physical approval from the user-side (at the loss of some security properties). Other hash-based password managers are Physically-Effortless. However, commercial token-based, client-based, and web-based password managers are not effortless since they require the user to transfer the password from the device to the client in addition to typing the master password (unless they offer an additional feature of form auto filling).

Deployability Metrics

D1. Client-Compatible (D4 in [21]): The schemes that do not require any changes at the client-side are called Client-Compatible. This includes schemes that do not require any software or browser extension. Except for the Firefox password manager and web-based commercial password manager, none of the other studied scheme are Client-Compatible.

D2. Nothing-to-Carry (U3 in [21]): The schemes that do not necessarily require additional hardware to operate are in this category. Device-aided schemes would not satisfy this property. However, device-aided schemes increase the security of the system, since an attacker who knows the master password, requires the second factor to authenticate to the service. SPHINX is a device-aided approach. The current implementation of SPHINX uses a smartphone as the secondary device. Tapas is also a device-aided password manager. Token-based password manager also require extra hardware or device.

Q1. Demographic Information

The demographic information of participants in our study is summarized in Section V-A.

Q2. Computer Skills and Technical Background

1) How do you rank your general computer skills? (Poor, Average, Excellent)
2) How do you rank your general computer security skills? (Poor, Average, Excellent)
3) How comfortable are you with Chrome browser? (Extremely, Very, Moderately, Slightly, Not at all)
4) How familiar are you with browser extensions, plugins or add-ons? (Extremely, Very, Moderately, Slightly, Not at all)
5) How often do you visit websites that require password from your computer(e.g. Gmail, Facebook, Twitter,...)? (Not on a daily basis, Once a day, Twice a day, Several times a day)
6) How often do you login to Gmail? (Not on a daily basis, Once a day, Twice a day, Several times a day)
7) How do you usually choose a password? (Easy to remember, Difficult for others to guess, Strong randomized password, Similar over multiple accounts)
8) Have you used any password manager before (e.g. storing password on the browser or on the mobile

phone, or applications such as LastPass or PwdHash, ...)? (Always, Most of the time, About half the time, Once in a while, Never)

9) Which of the following password managers have you used before?(Basic browsers password managers, LastPass, PwdHash, 1Password, Roboform2Go, KeePass)
10) What is your primary reason for using a password managers? (I do not use them, Fast and easy logins, Increasing security, Personal choice)

Q3. Task Easiness: How easy it was to execute this task?

Q4. Task Satisfaction:. How satisfied are you with SPHINX at performing this task?

Q5. System Usability Score:

Considering the recent experience logging in to Gmail using SPHINX the participants rated the 10 SUS questions [22].

Q6. SPHINX Specific Questions: Compared to a password-only authentication mechanism that does not require the device, how much do you agree with the following statements: (Answer Options: Strongly disagree, Somewhat disagree, Neither agree nor disagree, Somewhat agree, Strongly agree)

- *Transparency*: My experiment logging in with SPHINX is similar to logging in with password only.
- *Security and Trust*: (1) I feel my passwords are more secure using SPHINX; (2) I trust SPHINX to protect my password; (3) I am uncomfortable with not knowing my actual passwords for a web site; (4) Passwords are safer when users do not know their actual password; (5) am comfortable with letting SPHINX decide a strong password for me.
- *Necessity*: (1) My passwords are safe even without using SPHINX; (2) I need to use SPHINX on my computer to protect my passwords.
- Portability: I am confident that logging in from remote computers will be convenient.

Q6. Open Ended Questions:

Following is the list of open ended questions.

1) From your understanding, what does SPHINX do?
2) Did you face any problem/issue/difficulty when using SPHINX?
3) Do you have any suggestions for SPHINX that can make it more useful or easier to use?
4) Do you think SPHINX is better than other password managers?
5) Will you be willing to use SPHINX in your day-to-day use?
6) Which types of sites you will be interested in using SPHINX?