End-to-End Passwords

Scott Ruoti
MIT Lincoln Laboratory
scott@ruoti.org

Kent Seamons
Brigham Young University
seamons@cs.byu.edu

ABSTRACT

Passwords continue to be an important means for users to authenticate themselves to applications, websites, and backend services. However, password theft continues to be a significant issue, due in large part to the significant attack surface for passwords, including the operating system (e.g., key loggers), application (e.g., phishing websites in browsers), during transmission (e.g., TLS man-in-the-middle proxies), and at password verification services (e.g., theft of passwords stored at a server). Relatedly, even though there is a large body of research on improving passwords, the massive number of application verification services that use passwords stymie the diffusion of improvements—i.e., it does not scale for each improvement to require an update to every application and verification service.

To address these problems, we propose a new end-to-end password paradigm that transfers password functionality to two endpoints, the operating system (entry, management, storage, and verification) and the password verification service (verification, and verification token storage). In this paradigm, passwords are *never* shared with applications or transmitted over the network, but are instead verified using zero-knowledge protocols. There are five key benefits of this approach that are not possible with the current password paradigm: (a) a minimal attack surface, (b) protection from password phishing, (c) protection from malware, (d) consistent password policies, and (e) the ability to more rapidly diffuse improvements from password research.

KEYWORDS

End-to-end, passwords, password-based authentication, safe password entry, strong password protocols

ACM Reference Format:
Scott Ruoti and Kent Seamons. 2017. End-to-End Passwords. In *NSPW 2017: 2017 New Security Paradigms Workshop, October 1–4, 2017, Santa Cruz, CA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3171533.3171542

1 INTRODUCTION

Even with years of research into new authentication technologies, passwords still dominate the authentication landscape. This is due primarily to a combination of security, deployability, and usability that has been difficult to match [12].

Despite the persistence and popularity of passwords, there are serious threats that plague current password-based authentication at every level. First, at the operating system level, passwords can be stolen by malware that logs users' keystrokes. Second, at the application level, users are easily tricked into entering their credentials into phishing applications (e.g., phishing website) [45]. Third, at the transmission level, passwords can be stolen when in flight; this is especially true considering that users struggle to identify when a secure session will be used to transmit their password [28]. Finally, at the password verification service level, passwords are frequently stolen from storage. This type of attack is especially worrisome since users have no information, control, or assurance regarding how the server stores their passwords. Most of the recent large password database leaks did not properly salt and hash the passwords [37].

These attacks occurring at all levels of password processing demonstrate that the attack surface for passwords is extremely large. There have been many attempts in the research literature to improve password-based authentication—e.g., strong password protocols [10, 57, 119], password creation policies [65, 106, 114], phishing-resistant interfaces [29], advice to administrators [37]. Unfortunately, most of this research has failed to have a tangible impact on password-based authentication. One key reason for this lack of diffusion is that there are too many systems to update. It is not feasible to update every password-based authentication system every time there is a new suggestion from the research literature.

In this paper, we argue that to make substantial progress to securing passwords we need to adopt a new paradigm for password-based authentication. As such, we propose a new end-to-end password paradigm, with the operating system and the password verification service as the endpoints that are responsible for nearly all password handling. The application and any communication channels act as untrusted components that only serve to relay messages in the zero-knowledge proof between the operating system and password verification service.

Functionality is not split evenly between the operating system and the password verification service. Instead, the OS assumes nearly all functionality associated with passwords—including creation, storage (optional), management (i.e., policy), entry, and verification. The password verification service is only responsible for verifying passwords. To further secure users' passwords, this paradigm requires the use of strong password protocols [119]—i.e., passwords **never** leave the operating system's control, and verification of the password is accomplished using a zero-knowledge proof.

In addition to the benefits of using a strong password protocol—e.g., the verification service cannot store passwords in plaintext—there are several benefits that are uniquely tied to our new end-to-end password paradigm:

(1) **Reduced attack surface.** The use of strong password protocols obviates the need to trust the communication channel, rendering any communication-level attacks ineffectual.

NSPW 2017, October 1–4, 2017, Santa Cruz, CA, USA

ACM ISBN 978-1-4503-6384-6/17/10…$15.00
https://doi.org/10.1145/3171533.3171542

Relatedly, by reducing the application to act as only a communication channel, we similarly limit any attacks at the application level. While the verification service must still store a verifier token, this token is guaranteed to have properly protected the underlying password data (e.g., salting and hashing). Moreover, most other functionality has been removed from the verification service, substantially limiting viable attacks. Finally, the functionality that is in the operating system can be hardened using operating system primitives [25, 110], leaving a smaller attack surface than when the equivalent functionality was in an application.

(2) **Protection from password phishing.** By centralizing all password entry into the operating system, there will be a single interface for all password entry. By definition, any other password-entry interface will become a phishing interface. This makes it feasible to automatically detect phishing attacks, sidestepping the problem of requiring the user to distinguish between a legitimate and malicious password entry interface [29].

(3) **Protection for malware.** To our knowledge, the threat model for all existing password-based authentication research includes the host environment as part of the trusted computing base. For example, key logging malware can trivially steal passwords as they are entered. By moving password entry into the operating system, it will be possible to run password entry in a protected mode that restricts any program other than the operating system from observing user input (e.g., key logging, microphone, camera) [110]. Furthermore, password functionality could be part of the signed portion of the operating system kernel [25], protecting this functionality from advanced malware such as rootkits.

(4) **Consistent password policies.** As the operating system will be responsible for the creation and entry of all passwords, it can help users create passwords in a consistent manner. No longer will users need to remember a different password policy for every application they use. Additionally, the operating system can monitor the health of users' passwords, and give them advice regarding their password posture—such as notifying a user that they are sharing a password between a common website and a high-security website like a banking institution.[1]

(5) **Rapid diffusion of improvements to password-based authentication.** Centralizing most password functionality in the operating system has the benefit that there are fewer implementations that need to be updated when improvements or fixes need to be made. For example, if a new password entry interface is shown to be more usable than the existing interface, only the implementation in each OS will need to be updated, and not every single application that has a password entry interface. Similarly, if a class of passwords is shown to be insecure, the operating system can detect insecure passwords when they are entered and notify the user of this new result.

[1]Some password managers are starting to provide this type of advice, but it will be able to be much more comprehensive when a single entity handles all passwords.

We recognize that it would be trivial to create an unusable instantiation of this end-to-end paradigm or associated architecture. To this end, we stress that any attempt to prototype our paradigm or architecture must be done based on the principles of user-centered design [5], usable security [41], and scientific, empirical evaluations.

Roadmap. Section 2 gives an overview of related work. Section 3 describes the end-to-end password paradigm in greater depth, and Section 4 gives a possible architecture for this new paradigm. Section 5 discusses a research agenda for end-to-end passwords, and Section 6 concludes the paper.

2 RELATED WORK

Passwords are the de-facto standard for authentication. There have been multiple attempts at either improving them or replacing them: proxy [36, 83], federated [34, 49, 67, 86, 107], graphical [19, 104], cognitive [54, 60, 98, 113], paper tokens [47, 69, 116], visual crypto [84], hardware tokens [32, 56, 89, 99, 121], phone-based [43, 77, 82, 109], biometric [6, 24, 87], and recovery-based [14, 59, 94]. Many of these offer important security benefits, but none of them has been able to replace current password-based authentication due to perceived limitations in usability and deployability [12]. Chiasson et al. found that even if a non-password-based system is usable, users are still likely to prefer passwords [18]. Thus, it is imperative to recognize that for the foreseeable future passwords are not going away, and research must be done to strengthen password-based authentication [52].

2.1 Phishing

Dhamija and Tygar [29] proposed dynamic security skins as a solution to address phishing. In their approach, the browser creates a trusted window for entering passwords. The browser and the server each create an image (or skin) that surrounds the password entry window, and the user should check that they match before trusting the password prompt enough to enter their password. Unfortunately, dynamic security skins was never evaluated empirically, and later research suggests that users are unlikely to notice if security indicators are missing from a malicious interface [95].

2.2 Password Policies

There has been significant work regarding password policies. Research by Inglesant and Sasse shows that password policies often place undue burdens on users as they try to cope with managing too many passwords [55]. Password creation policies is an area that has been explored to identify how users create secure passwords and how mechanisms like password strength meters can impact the process [65, 106, 114], Another vein of research examines the longstanding practice of forcing users to change their passwords on a regular basis, finding that it was ineffective [122].

2.3 Password Storage

Best practices for password storage at the server dictate that passwords should be salted and hashed before being stored. Florencio et al. [37] showed that the majority of prominent cases involving leaked password databases did not follow this practice, making it easier for the attacker to recover all of the leaked passwords.

Almeshekah et al. [7] introduced a novel approach for server-side password storage that prevents an offline attack. The design of the password file includes support for dummy passwords that do not correspond to a valid user. Instead, an attacker is led to believe they have cracked a legitimate password, and any attempt to log in to the server provides reliable evidence of a password file leak and sets off an alarm.

In our work, we put the client in charge of creating the stored, cryptographically-protected password. This protects the user from trusting a server that fails to follow best practices for storing passwords.

2.4 Password Managers

Cloud-based password managers have become popular, and ease the burdens of password management on users that are willing to trust a third-party to store their passwords. Two recent security evaluations of password managers by Li et al, [75] and Silver et al. [96] revealed security flaws in popular password managers that can result in the compromise of user's passwords. This work illustrates the challenges of securing web-based password managers.

Password managers are designed to ease the burden of password management. Recent studies have explored password managers' usability. Chiasson et al. [20] conducted a 26-person user study comparing two password managers: PwdHash and Password Multiplier. Karole et al. [63] conducted a usability study of three password managers. The users in their study preferred portable password managers to a web-based manager. They credited the finding to the reluctance of users to trust an online password manager.

2.5 Strong Password Protocols

Password authenticated key exchange (PAKE) is a widely-recognized way to secure password-based authentication. PAKE protocols allow a user and a server to establish a session key using a shared secret derived from the user's password. The protocols do not require that a secure channel (e.g., TLS) be established first. PAKE protocols also prevent an eavesdropper from gaining any information that can be used in an offline attack against the user's password. Without the password, an attacker is also unable to determine the session key. Thus, the session key can be used for mutual authentication. There are a variety of PAKE protocols: EKE [10], PAK and PPK [13], SPEKE [57], J-PAKE [50], and EC-SAKA [4].

One problem with the original PAKE protocols is that a shared secret stolen from the server can immediately be used to impersonate the user. To address this problem, augmented PAKE protocols were created: AMP [71], Augmented-EKE [11], B_SPEKE [58], PAK-Z [76], AugPAKE [66], and SRP [118, 119]. These protocols are verifier-based, meaning that even if the shared secret is stolen, an attacker must still perform an off-line attack on the user's password before they can impersonate the user.

In 2005 Abdalla et al. proposed the first provably secure three-party PAKE (3PAKE) protocol [2]. 3PAKE extends the model of PAKE by including a trusted third-party (IDP). The user (U) and server (RP) both share a secret with IDP, but not with each other. IDP will help them establish a shared key and mutually authenticate each other. This reduces the number of parties that must share a secret with any given user. Recent research has continued to create improved 3PAKE protocols [17, 22, 70, 115].

While 3PAKE requires RP and IDP to share a secret, this is not always desirable. Abdalla et al. also proposed Gateway-oriented PAKE (GPAKE), a variant of 3PAKE that does not require a shared secret between RP and IDP [1]. This original protocol was susceptible to an undetectable online guessing attack [16, 120], but several later protocols have addressed this attack [3, 16, 112].

2.6 Safe Password Entry

Strong password protocols provide no protection if users enter their passwords into phishing applications. As such, it is important to have a trusted path [85] to the legitimate interfaces, providing safe password entry [90]. It should be noted that safe password entry without strong password protocols is still vulnerable to attack; the core idea of the proposed password paradigm is to demonstrate how safe password entry and strong password protocols can be used together to significantly strengthen password-based authentication.

The first step in using a trusted path is to have a secure mechanism for instantiating the trusted path. The most widely used mechanism is the secure attention key [42] (SAK): a special key or a sequence of regular keys that is detected by the operating system kernel, after which the operating system creates a secure channel for the user to interact with the intended interface. SAK has been used in operating systems (e.g., control-alt-delete in Microsoft Windows) and has been proposed for use in browsers (e.g., PwdHash [88], WebWallet [117]).

The second step is to ensure that users know when they have instantiated the trusted path, and with which interface they are currently interacting with. This can be done using hardware—for example, having an indicator on the hardware that shows when the user is interacting directly with a secure interface in the operating system. It can also be done using on-screen elements—for example, reserving a portion of the operating systems UI for secure interfaces [33] or graphically styling applications in a way that cannot be replicated by an attacker [29].

2.7 Single Sign-on

The problems of easy-to-guess passwords and password reuse hinder current password-based authentication. The number of parties that seek to establish passwords with users aggravates both problems. Single sign-on (SSO) addresses this problem by requiring that users only establish passwords with an identity provider (IDP). IDP assists other websites (RP) in authentication of the user (U). While U might have several IDPs, the number of IDPs is far less than the number of RPs. This helps users to avoid password reuse and encourages them to choose stronger passwords for use at their IDPs.

The two most well-known SSO systems on the web are OpenID [38, 39, 86] and OAuth [48]. OAuth has become the dominant SSO system in recent years, with Facebook Connect being the most prominent OAuth implementation [34]. These SSO systems have weaknesses. First, most SSO systems (including OpenID and OAuth) are built on browser redirects. This approach is both confusing to users [27, 103] and also provide an attack surface for phishers [15, 74, 78, 97, 100]. Second, current SSO systems largely ignore the

business implications of SSO on RPs and IDPs, which further limits their adoption [101]. Facebook Connect is the most widely adopted system precisely because it gives an incentive to RPs to adopt it (i.e., sharing information about users). There have been attempts to address these two problems in OpenID, but these systems have not seen adoption [102, 103]. We believe that strong password protocols have the potential to solve the susceptibility to phishing that most SSO systems experience.

Kerberos [79] has a long history as an SSO solution for enterprise and large organizations. Kerberos thwarts eavesdropping and phishing attacks by not transmitting the password to the server. Instead, the password is used to encrypt pre-authentication data sent to the server, and to decrypt a ticket returned to the client. Unlike PAKE protocols, Kerberos does not prevent an eavesdropper from using the encrypted data to conduct an offline attack to recover the password.

2.8 Multi-Factor Authentication

Passwords are a means of authenticating based on something you know. Other forms of authentication rely on something you have (e.g., YubiKey) or on something you are (e.g., fingerprint). While these other approaches have failed to supplant passwords, they can be used on top of passwords to provide additional security—i.e., multi-factor authentication. In practice, two-factor authentication based on the user's password and a device they own is the most common form of multi-factor authentication.

Authenticating users based on something they own is most commonly done by proving possession of a cellular phone. This can be accomplished by sending users a one-time password (OTP) sent over SMS [8], having an application which generates one-time passwords, scanning QR codes with the phone [30], demonstrating locality using ambient noise [61], etc. Alternatively, users can purchase specialty hardware that contains cryptographic keying material that uniquely identifies the users (e.g., YubiKey).

There is also a significant body of research that explores authenticating users based on who they are [111]. Examples include facial recognition [46], iris scanning [93], fingerprint scanning [21], and voice recognition. Weaker biometrics are also used for continuous authentication [105] (e.g., gesture [35] and gait [40] recognition).

The end-to-end password paradigm supports the use of multi-factor authentication. The non-password factors could be handled in the applications, or directly in the end-to-end password interface. The latter is more secure and could provide a consistent user experience that might lower the bar for the adoption of multi-factor authentication.

2.9 Pluggable Authentication Module

Pluggable Authentication Module (PAM) [92] is a mechanism that adds a layer of abstraction between application programs and the underlying authentication methods that they employ. An application developer programs to a standard API, and the operating system can be configured with alternative authentication mechanisms. One advantage of PAM is that it eases the cost to deploy new solutions. It may be possible to implement the end-to-end passwords paradigm using PAM.

3 END-TO-END PASSWORDS

In this section, we describe our threat model and the current paradigm for passwords and its problems. We then detail why existing efforts to secure passwords are insufficient to solve these problems. Next, we describe our new paradigm for passwords—called **end-to-end passwords**. Finally, we discuss the benefits unique to this new paradigm and also list its limitations.

3.1 Threat Model

Our threat model includes common attacks at all the levels of password-based authentication mentioned earlier. First, any local attacks involving malware on the user's device or phishing attempts that try to trick the user into disclosing their password. Second, any active or passive network attacker seeking to interfere with the authentication process in order to obtain the user's password. Third, an attacker that breaks into the password verification service in order to steal passwords or steal material that can be used in on offline attack. We do not address shoulder surfing or the risk of audio/video recordings near the user that attempt to discover the user's password. We also don't include hardware keyloggers, but focus on software attacks only.

3.2 Current Password Paradigm

At a high level, the current password paradigm requires users to enter passwords into an application, those passwords are then transmitted over some channel to a password verification service, which then verifies the authenticity of the password, and reports back to the application whether authentication succeeded. In practice, though, a wide range of responsibilities are spread across all parties—the user, applications, the communication channel, and the password verification services. This diffusion of responsibilities also creates a large attack surface. See Figure 1.

Users are responsible for creating passwords, remembering passwords, managing their password strategy, and entering passwords when prompted. Password phishing is a significant problem that research has been unable to effectively stop. The risk of stolen credentials is made more severe by poor password hygiene—password reuse and weak passwords. Furthermore, malicious applications can trick users into authenticating to a legitimate service, then use this authenticated session to carry out malicious activities under a user's persona.

Applications are responsible for the widest range of responsibilities. They require users to create the password and may also manage the password that a user may choose (e.g., enforcing a password policy). Applications also require users to enter their passwords before transmission to a password verification service in order to authenticate the user. Finally, in some cases, an application may also store a user's passwords (e.g., password manager).

Applications can be directly attacked in order to compromise passwords as they are entered. Alternatively, if the application stores the password an attacker can attempt to steal it. Finally, applications may enforce counter-productive password policies that weaken a user's password hygiene, such as requiring frequent password resets.

The communication channel is only responsible for transmitting a user's password, yet it is a relatively easy target for attack. In

	User	Operating System	Applications (e.g., websites)	Communication Channel (e.g., Internet)	Password Verification Service
Responsibility for passwords	• Create • Remember • Manage • Enter	–	• Create • Manage • Enter • Transmit • Authenticate • Store	• Transmit	• Receive • Verify • Manage • Store
Attack Surface	• Phishing attacks • Poor password hygiene • Authenticate malicious application	• Malware (e.g., key loggers)	• Compromise application • Theft of stored passwords • Poor password policy	• Plaintext channel • Downgrade attack (e.g., TLS MitM)	• Compromise service • Theft of stored passwords • Poor password policy

Figure 1: Current password model

most systems, the plaintext password is transmitted to the password verification service, even if the network connection is encrypted. If the security of the channel can be compromised then an attacker gains the password. In practice, there are still some communication channels that are unencrypted, rendering the user's password available to an eavesdropper. Additionally, attacks against the communication channel's security (e.g., TLS man-in-the-middle [81]) can also divulge the user's password to an attacker.

Lastly, the password verification service is responsible for receiving the user's password and then verifying that it matches the stored password. Additionally, the password verification services often manage the password that a user may choose (e.g., enforcing a password policy). Attackers can either directly compromise the verification service to steal passwords during processing, but more often an attacker will focus on stealing the password database. Often these passwords are stored in plaintext or with poor security [37]. Similar to the application, the verification service may also enforce counter-productive password policies that weaken a user's password hygiene.

Interestingly, even though the operating system is the interface between the user and the application, it has no responsibilities in the current password paradigm.[2] Regardless, it is still a part of the attack surface, as malware such as key loggers can be used to steal passwords as they are entered.

3.3 Inadequate Solutions

The threat space for passwords is well known, and many individual threats have proposed solutions. For example, password storage at the verification service can be significantly strengthened by requiring that passwords are hashed with a keyed algorithm executed on a cryptocard [37]. Alternatively, strong password protocols (e.g., SRP [119]) can obviate the need to transmit the password over the communication channel to the password verification system.

While it might be tempting to think that these piecemeal solutions can sufficiently strengthen passwords, their lack of adoption indicates that something more is needed. More specifically, adopting piecemeal solutions to the current password ecosystem is inadequate for the following reasons:

(1) **There are too many password systems.** Between all relevant applications and password verification services there are tens-of-thousands if not hundred-of-thousands of systems that rely on passwords. As such, updating **each** of these applications or verification services **every time** there is an improvement to password-based authentication is not scalable. Additionally, the large number of implementations is directly related to the large attack surface for the current password paradigm.

(2) **It is difficult to verify that best practices are followed.** Many applications and verification services claim to have strong security for passwords, but in practice this often fails to be true. While it would be ideal to analyze each application and verification service to verify whether they use best-practice, this is infeasible considering the large number of varied implementations. Moreover, many password verification services are hosted remotely and are completely unavailable for black-box analysis. As such, with the current paradigm it is unlikely that the security of password verification services can ever be fully trusted.

(3) **Phishing attacks are nearly-impossible to block.** No matter how much password-based authentication is strengthened, if users can be fooled into entering their password into a malicious application, that password will still be compromised. While unphishible interfaces have been proposed [26], they are untested and later research has shown that it is difficult for users to distinguish between legitimate and malicious interfaces [95].

[2]While passwords are used to authenticate to the operating system, in this case the operating system is acting as yet-another application.

(4) **No resilience against a compromised host environment.** The current password ecosystem is predicated on the assumption of a clean host environment. If there is malware—such as a key logger—on the system, then the user's password will be stolen whenever it is entered. While this assumption is necessary for the existing paradigm, it does not match reality—many users are operating in a host environment that is partially compromised (e.g., keylogger installed, though no OS rootkit installed).

None of this is to say that the research done to strengthen password-based systems is wasted. Instead, it means that a new password paradigm is needed in order to allow existing research to achieve its full potential.

3.4 End-to-end Passwords

To address these limitations, we propose a novel end-to-end password paradigm that combines strong password protocols with a trusted path for password creation, management, and entry. In this paradigm, the operating system and password verification service, as the end points of password entry and verification, respectively, are responsible for handling nearly all password-related functionality. In contrast, the application and communication channel only serving to facilitate communication between the operating system and password verification service (see Figure 2).

Critically, passwords **never** leave the operating system. This is accomplished through the use of strong password protocols—instead of transmitting the user's password, the operating system executes a zero-knowledge proof demonstrating possession of the password to the verification service. This zero-knowledge proof can be executed over an unsecured channel and is impervious to both eavesdroppers and active attackers.

Because the password verification service is not under the user's control or ability to audit, we limit its responsibilities to verifying the zero-knowledge proofs generated by the operating system, and storing a verifier tokens needed to validate the zero-knowledge proof. While these verifier tokens can still be stolen and used to brute-force the user's password, they still have one key advantage—the operating system constructs the verifier token and can ensure that it is properly salted and hashed.

Finally, the user's operating system acts as a trusted pathway [85] for the creation, managing, and entry of user credentials, providing *safe password entry*. Safe password entry provides two benefits—first, it ensures that strong password protocols are used during authentication and second, it identifies and blocks non-approved password interfaces. Without safe password entry, strong password protocols are ineffectual against phishing, as the attacker will simply refuse to use a strong password protocol.

In the end-to-end password paradigm, the operating system's safe password entry interface handles all stages of the password life-cycle—for example, verifier token creation and password entry—becoming the singular location where users interact with passwords. It is also responsible for detecting other password entry interfaces and blocking them, protecting users from entering their passwords into a phishing application.

While the singular password entry interface in the operating system may evoke thoughts of single sign-on and password managers, end-to-end passwords is orthogonal to both of those approaches. End-to-end passwords can be implemented without single sign-on or password manager functionality, though it can also include them as well (see Section 5).

3.5 Benefits

The most important benefit of this new paradigm is a significant reduction of the attack surface for passwords (see Figure 2). Importantly, applications and the communication channel can be treated as fully untrusted entities, removing them from the attack surface. Similarly, the attack surface of the password verification service has been significantly reduced, with theft of the stored verifier being the only residual attack vector. While the operating system potentially has a larger attack surface, it can be protected through operating system hardening, requiring an attacker to compromise the security of the operating system. Finally, as discussed later in the paper, the new paradigm helps prevent phishing attacks and can also be used to improve user's password hygiene.

This new paradigm also addresses each of the four limitations previously identified for the current password paradigm:

(1) **There are too many password systems.** End-to-end passwords reduces the number of systems that play a significant role in password-based authentication. Specifically, most responsibilities for passwords are centralized in the operating system. In most cases only a handful of operating system implementations will need to be updated when improvements to password-based authentication are discovered. While changes to the strong password protocol will require modifications to the application, communication channel, and password verification service, these represent a small fraction of possible improvements to password-based authentication.
(2) **It is difficult to verify that best practices are followed.** As the operating system is local, it is possible for it to undergo block-box analysis to determine whether a given implementation follows best practices. Additionally, the use of a strong password protocol—which provides mutual authentication—allows the operating system to verify that the password verification service is correctly verifying the user's password. Furthermore, the operating system controls how the passwords are salted and hashed, ensuring that this process is done correctly for all applications. Finally, even though the verification service's storage of the verification token cannot be audited, the loss of the verifier database is no worse than the theft of a password database currently is, and is potentially much less impactful.
(3) **Phishing attacks are nearly-impossible to block.** As the operating system hosts the only legitimate password entry interface, by definition all other password entry interfaces displayed to the user are malicious. This fact makes it simple for the operating system to detect phishing attacks, obviating the need for the user to be ever vigilant against password phishing. Specifically, the operating system can detect any other application that is displaying a password entry interface and terminate the application.

	User	Operating System	Applications (e.g., websites)	Communication Channel (e.g., Internet)	Password Verification Service
Responsibility for passwords	• Create • Remember • Manage • Enter	• Create • Manage • Enter • Authenticate • Store	• Transmit zero-knowledge proof	• Transmit zero-knowledge proof	• Verify • Store a verifier token (i.e., not the password)
Attack Surface	• Authenticate malicious application	• Compromise hardened OS primitives	—	—	• Theft of stored verifier

Figure 2: New end-to-end password model

(4) **No resilience against a compromised host environment.** Using operating system hardening techniques—e.g., TPM, signed kernels [25], isolated execution contexts [110]—it is possible to protect password functionality hosted in the operating system from attackers. For example, the password entry interface could force the system into an isolated execution context where traps for user input are ignored, preventing a key logger from scraping the user's password. Similarly, using a signed kernel, we can prevent rootkits from compromising the operating system's control of passwords. While these protections are not infallible, they do make it more difficult for attackers to compromise the system.

3.6 Limitations

While end-to-end passwords can significantly strengthen password-based authentication, it is not a panacea to all authentication-related problems. First, it does not remove the need for the verification service to store a verification token in order to verify the user's identity. While this verification token has several important benefits over existing password storage, it can still be used to perform a brute-force attack against the user's password. Second, the end-to-end password paradigm does not address what an application does after authentication takes place. If a user authenticates to a malicious application, then that application can take malicious action using the authenticated account.

Third, this paradigm requires that when a vulnerability is discovered, applications will need to wait for the operating system to deploy a mitigation—i.e., the application is unable to independently deploy a mitigation. While in most cases, operating system patches are more reliable than application updates, this is not true if patches are unavailable (e.g., old Android devices) or if users fail to install those patches. While the latter can be addressed by forcing security patches and updates to be deployed, the former is more difficult. Later in the paper we describe potential solutions to this problem, but these solutions are untested and for now this problem remains a limitation.

4 EXAMPLE ARCHITECTURE

In this section, we give an example architecture that adopts the end-to-end password paradigm. It is only an example architecture, as several portions of the architecture are malleable and could be changed while still staying within the bounds of the end-to-end password paradigm.

In our architecture, the strong password protocol is instantiated using the Secure Remote Password protocol [119] (SRP). The workflow for establishing a new account is given in Figure 3. While that diagram demonstrates account establishment, the process is the same as authentication, except that steps four and five are omitted. The detailed steps in this workflow are:

(1) The user indicates to the operating system that they wish to create an account or authenticate for a given application.
(2) The user is shown a password entry interface by the operating system. While this interface is displayed, no other applications are shown, and user input is not available to any entity other than the operating system.
(3) The user enters their password to enter their credentials into the operating system.
(4) (Account Establishment Only) The operating system generates a salt and verification token for the entered password. For password hashing in the SRP protocol, we recommend the use of PBKDF2.
(5) (Account Establishment Only) Using TLS, the username, verification token, and salt are sent to the password verification service. Our architecture uses TLS for simplicity and interoperability with existing technologies.
(6) The operating system and the password verification service each transmit a single message allowing for the zero-knowledge proof of password knowledge.
(7) In the next set of messages, the operating system proves the user's knowledge of the password, and the password verification service proves its knowledge of the verification token.
(8) The user is notified by the operating system that account creation or authentication was successful.

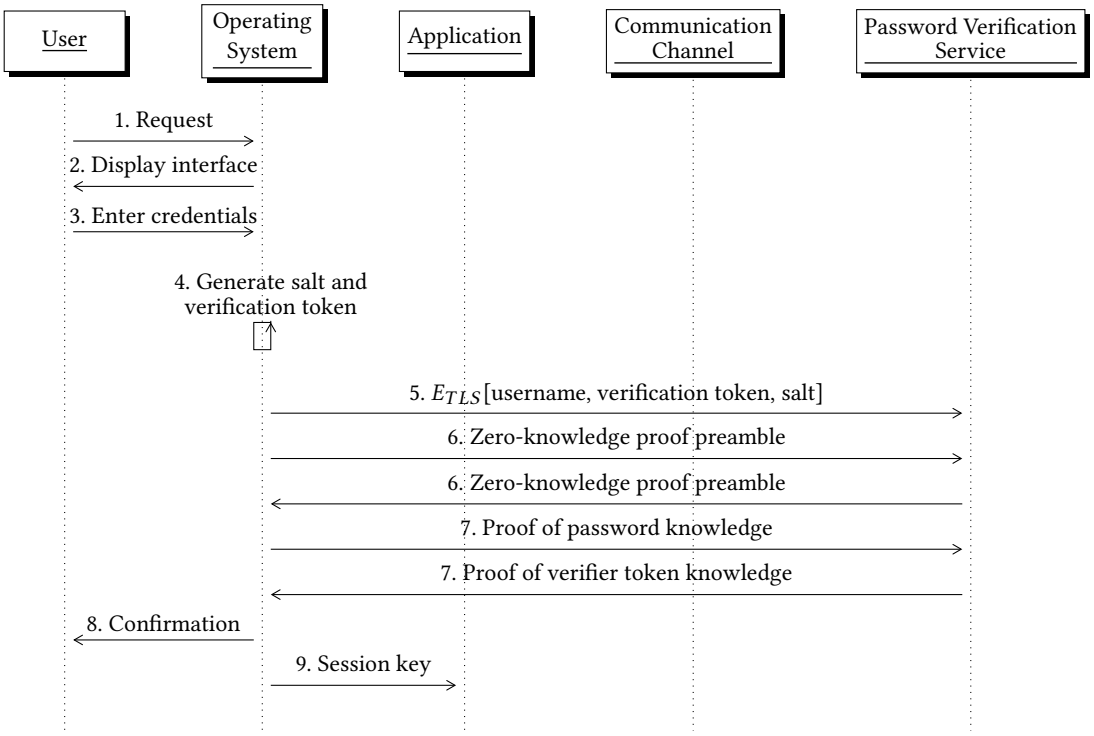

Figure 3: Account Establishment Flow Diagram

(9) The application is given a session key derived from the zero-knowledge proof. This session key can be used to authenticate operations created by the application.

In the remainder of this section, we describe the functionality of each component in greater depth.

4.1 Operating System

The operating system has the lion's share of the responsibility for end-to-end passwords. This imbalance is ideal, as consolidation of features in the operating system instead of in each application has the potential to speed up the diffusion of improvements for password-based authentication. Additionally, the operating system is the most secure part of the user's operating environment, providing the best chance of protecting passwords from malware. As such, we identify several important protections for the operating system to provide.

First, the operating system should ensure that password functionality runs in an isolated, secure context. For example, the system could restrict access to user input (e.g., keystrokes, microphone) while the user is entering their passwords. Similarly, the operating system should put as much of its functionality in the signed portion of the kernel as a safeguard against privileged malware, including rootkits.

Second, the operating system should actively detect and block attempts to phish the user's password. An advantage of the end-to-end password paradigm is that it makes it feasible to automatically detect password phishing. If the system detects a password entry interface that it does not host, it is a password phishing attack. The operating system can take defensive actions such as prevent keyboard input into that application while the interface is present, remove the interface, kill the application or a specific screen in the application, etc. Detection of password entry fields can use traditional methods (e.g., LastPass auto-fill [72]). Alternatively, because the detection mechanism is located in the operating system, it is possible to use image detection on the final image displayed to the user to detect phishing interfaces. This allows detection of phishing interfaces that are not currently detectable—e.g., applications that draw their interface instead of using UI widgets, and interfaces composed from multiple applications that appear to be a single application interface to the end user.

Third, the operating system should provide a conditioned-safe ceremony for the user to instigate authentication [62]. This ceremony is especially important during the transition to end-to-end passwords when there are password entry interfaces that have not yet migrated to the operating system. A conditioned-safe ceremony will help ensure that users are typing their passwords into an interface controlled by the operating system.

Fourth, because the operating system can observe all of the user's passwords, it could enforce a coherent password policy. For example, passwords for all high-value applications must be strong enough to resist an offline attack.[3] For other sites, the operating system could require passwords that resist only online attacks, not offline attacks. Furthermore, the operating system could correlate

[3]While determining whether an application is high-value objectively is difficult, there are possible workarounds. For example, the operating system could maintain a curated list of well-known, high-value applications (e.g., banking). Alternatively, users could self-identify which applications they consider high-value.

the passwords used across multiple applications and notify users of re-used or similar passwords that could compromise their security.

Fifth, the operating system should leverage the mutual authentication provided by SRP (and strong password protocols in general). The mutual authentication information allows the operating system to know that authentication completed successfully, indicating that the user had previously established an account with the password verification service. Presenting this information to the user in a cogent fashion could help protect users from disclosing sensitive information (e.g., credit card information) to sites that can't authenticate themselves to the user.

4.2 Application

Since the application is not an endpoint, it has limited impact on end-to-end passwords. Its primary function is to relay messages between the operating system and the password verification service. After authentication is complete, it also receives a session key that can be used to sign operations, linking those operations to the user's account.

All other functions—for example, showing the state of mutual authentication—is left to the operating system. If the operating system detects that the application is trying to assume the operating system's responsibilities, it will block the application. This is essential to ensure that functionality stays at the endpoints, and is not simulated/moved to the application.

4.3 Communication Channel

Similar to the application, the communication channel is only used to transmit messages between the operating system and password verification service (by way of the application). Due to the nature of strong password protocols, the communication channel does not need to be encrypted.

4.4 Password Verification Service

The password verification service is the other end-point in this architecture, and is the only entity with significant responsibilities other than the operating system. The key responsibility of the password verification service is to store the verification token and use it to authenticate the user. While the verification token is guaranteed to be properly salted and hashed, ideally the verification service will store it securely. In the best case, the application will store the verification tokens in a database encrypted by a cryptocard [37].

It is important to note that in contrast to the current paradigm, the password verification service has no control over the passwords selected by the user. While this prevents the verification service from promoting poor password policies, it also prevents verification services that are used as part of high-security systems from enforcing stronger password policies. To address this, during account establishment, the password verification service is allowed to inform the operating system the desired level of password security. While the operating system ultimately decides the password policies that it will enforce, it can use this information to guide its decision.

5 RESEARCH AGENDA

In conjunction with the end-to-end password paradigm and our example architecture, we also describe a research agenda. This agenda has two purposes: first, it identifies research that must be done to guide the implementation and adoption of end-to-end passwords, and second, it describes new research questions that exist because of the new paradigm. This agenda can be split into three topic areas—usable security research, systems research, and research supporting the transition from the current password paradigm—though, there is substantial overlap between all three areas. For each of these areas, we list several topics of particular note but do not claim that this list is exhaustive.

5.1 Usable Security

Passwords are a heavily user-centric technology, and any attempt to change the way users interact with passwords must avoid introducing usability hurdles, or users will reject it. To this end, it is important that principles of usable security are applied to system and interface design. Also, design and implementation must be driven by a systematic application of the scientific method [53] in order to establish which design elements are essential to the use and adoption of end-to-end passwords.

User acceptance and mental models. At the outset of this research agenda, it is necessary to conduct studies to evaluate users' attitudes towards end-to-end passwords, and their willingness to adopt this system. Specifically, this research would identify what features end-to-end passwords needs to provide in order to motivate adoption, and what pain-points must be avoided. Additionally, studies should explore users' mental models for authentication, single sign-on, password managers, and trust in the operating system. This information would help inform later efforts, leading to a design and implementation of end-to-end passwords that would meet users needs and be suitable for adoption by the masses.

Account selection and password entry interface. Users spend a non-negligible amount of time authenticating each day [51]. Optimizing the efficiency of password entry and account selection has the potential to save users a significant amount of time and effort. One simple approach to streamlining authentication is to allow the operating system to remember the user's password and automatically re-authenticate them within a set amount of time (e.g., for 30 days). Still, care must be taken to ensure that efficient password entry does not compromise the account selection experience, as users frequently share computers (e.g., between spouses) and have multiple accounts with a single application.

Accessible authentication. For a variety of reasons, many users struggle to authenticate using existing password entry interfaces [31]. Even though there have been efforts made to make authentication more accessible [9, 68], adoption by applications is nearly non-existent. In the end-to-end password paradigm, there is only a single authentication interface for all applications, meaning that accessibility improvements to this interface would benefit all applications. This provides a much more rapid means of testing and deploying more accessible authentication. Alternatively, the end-to-end password entry interface could be made modular, allowing different user groups (e.g., visually impaired, elderly) to use interfaces specialized to support their particular abilities.

User-initiated authentication. In our architecture, we require that the user—not the application—starts the authentication process. As such, the user needs some method for indicating which application they want to authenticate to and under which context (e.g., the domain that is being authenticated against). Identifying the appropriate mechanism for this action—e.g., key combination, interface element to click—is a key research problem that needs to be identified in depth. Additionally, work needs to be done to help the user ensure that they are authenticating to the application they believe they are, and not a malicious application that has stolen focus from the legitimate application (e.g., an invisible application displayed over the legitimate application).

Safe ceremonies for password entry. Safe ceremonies help ensure that users are in a safe context when executing a sensitive activity [62]. For example, Microsoft Windows lets users press control-alt-delete to ensure that they are interacting with the operating system. Some type of safe ceremony should also be employed to allow users to initiate password entry. This could be a keyboard action like in Microsoft Windows or clicking an unspoofable user interface element. Research should establish which mechanisms are most usable and help the users accurately indicate which application needs to be authenticated. Additionally, if a conditioned-safe ceremony for password entry could be discovered [62], that would be ideal.

Monitor and enforce password policies. Because the operating system has the ability to observe all of the user's passwords, it could be used to enforce a cogent password policy. For example, for high-value applications, it could enforce a policy requiring passwords that are sufficiently strong to prevent offline attacks. For low-value applications, the operating system could require that passwords are only strong enough to resist online attack, not offline attack [37]. Furthermore, the operating system could correlate the passwords used across multiple application and help users identify re-used or similar passwords that could compromise their security.

Research will need to be done to identify the correct password policies to enforce for which classes of applications, otherwise this advice could be ignored, or prove detrimental to users' overall security [37, 122]. Similarly, research should guide the presentation of policy to help users create appropriate strength passwords [65, 106, 114].

Mutual authentication. Mutual authentication allows the password verification service to know that the user has the correct password, and it allows the user to know that they previously established an account with that password verification service. This helps prevent a malicious application from pretending to be a legitimate application, as the malicious entity will not have the password verifier stored by the legitimate password verification service. This then helps the user, who will not enter sensitive information (e.g., credit card, address, social security number) into the application until they have confirmed that they really have a pre-existing account with the password verification service.

How to inform the users of this mutual authentication is an open research question. While a naïve solution would be to inform the user immediately after authentication, it is possible that the user would struggle to track which applications had been authenticated and which had not. On the other hand, designing unspoofable security indicators is a hard problem [95]. Still, the benefit of mutual authentication justifies additional research to see if there is a solution that is usable and secure.

Choosing fewer, but stronger passwords. If users had fewer passwords to manage it is possible that they would choose stronger passwords. Other than reusing passwords, which has known problems, there are two approaches to reducing the number of passwords a user needs to remember—password managers (e.g., LastPass [72] and single sign-on (e.g., OAuth [48] or FacebookConnect [34]).

Password Manager. For end-to-end passwords, the operating system would become the password manager. When a user creates a new account, the password manager will generate a password. It would store these generated passwords in an encrypted password vault that is protected with a master password. The user would be required to remember only this master password. Once the master password is entered, the operating system would select the correct password from storage and use it for authentication.

The strength of this approach is that the user needs to remember only one password, yet every password verification service will have a verifier token for a different password. Moreover, even if the verifier token is stolen, the user's account is safe because the randomly generated passwords will be sufficiently strong to survive offline brute force attacks. In this way, a password manager is a weak form of two-factor authentication. The user needs to have both the device that stores the set of randomly generated passwords and knowledge of the master password that unlocks them.

The disadvantage is that a user can no longer move to a new device and immediately authenticate. Instead, the user will be required to synchronize the new device with the existing password vault. This can be done device-to-device, but this limits the user to only using devices which can be brought together and which have previously been synchronized. This sacrifices a key benefit of passwords, that they can be used anywhere and at any time. Alternatively, the password vault can be stored and shared using the cloud. This allows the user to access their passwords from any Internet-connected device, but creates a single, Internet-accessible, point of failure.

Single Sign-on. Single sign-on (SSO) reduces the number of passwords a user needs to create by allowing many applications to authenticate to a single password verification service. Simply, SSO can be seen as a more secure form of password reuse—even though the user authenticates to many applications with a single password, neither these applications or their backend services have access to the password verification service.

The key benefit of SSO in comparison to password managers is that users can still authenticate from any device. The disadvantage is that if the SSO verifier token is stolen, it can then be used to compromise a wide range of applications. Still, this disadvantage is tempered by the fact that SSO password verification services (e.g., Google, Facebook) are likely to employ much stronger security for their storage of verifier tokens (e.g., encryption using a cryptocard).

5.2 Systems Research

The separation between systems research and usable security research for password-based authentication is somewhat vague. Still,

we have identified several research topics that are more closely related to traditional systems research.

Safe storage of password verification tokens. Significant research has examined the storage of passwords at the password verification service [37]. A similarly concentrated research effort needs to be made for password verification tokens. In each case, proposed methods should be evaluated to determine their feasibility both for large corporations and smaller companies. For example, while cryptocards are likely available in the corporate setting, it is highly unlikely that they will be used by smaller companies. By addressing the latter need, it is more likely that these better password storage behaviors can be widely adopted.

Detecting password interfaces. In order to prevent password phishing, it is necessary to detect other password entry interfaces. After detection, these malicious interfaces can be blocked—for example, the hosting application could be terminated, or the operating system could refuse to render the interface to the user's screen and block user input to the application until the interface is removed. For applications that host sub-applications (e.g., browsers), the application could be notified of the malicious interface so that that the hosting application can remove the malicious sub-application (e.g., website) without requiring the operating system to terminate the entire application.

Research should be conducted to increase the efficiency of detecting password entry interfaces. This includes the use of traditional methods such as parsing the application's document object model. Additionally, the operating systems vantage point allows other novel methods for detecting phishing interfaces. For example, the operating system could employ image detection algorithms to identify password interfaces. This would allow the detection of interfaces drawn directly to the screen (i.e., not using UI widgets) or interfaces that visually appear to be a single interface but are actually cobbled together from multiple malicious applications. Alternatively, similar to Google's Password Alert browser extension [44] the operating system could monitor all keystrokes and detect if the user ever types their password into a non-operating system interface. Upon detection, the operating system could attempt to terminate the application receiving the keystrokes or alert the user of the potential compromise.

Pluggable operating system component. Instead of requiring that the operating system provider implements end-to-end passwords, the provider could instead provide a pluggable interface allowing for third-party end-to-end password implementations. There are two key benefits to this approach. First, it would allow the end-to-end password system to be updated separately from the operating system. This could address the limitation of running end-to-end passwords on systems where operating systems updates are unavailable (e.g., old Android devices). Second, it would allow users to choose the implementation that they most prefer—e.g., if they dislike the Microsoft implementation, they could use the Google implementation. Research needs to be conducted to explore whether this approach is viable and how it should be designed.

Alternative strong password protocol approaches. We selected SRP as the strong password protocol in our example architecture as it is the most efficient strong password protocol, both computationally and in the number of rounds of communication required. Due to the use of the SRP, we required that the application be handed the session key generated during authentication. An alternate approach would be to use a gateway-based strong password protocol [3, 112], treating the application as the gateway, allowing the gateway to authenticate the user without requiring the operating system to divulge the session secret it derived during the authentication process. Similar research could explore alternative constructions of strong password protocols that might address other use cases.

Passwords for encryption. In addition to authentication, passwords are sometimes used to generate symmetric keys to encrypt a file. For example, SSH keys are password encrypted when stored on the user's device. While the focus of our paradigm was password-based authentication, research needs to examine how the operating system can also be responsible for allowing users to enter passwords to decrypt files. This research will fully centralize the use of passwords into the operating system and away from applications.

Continuous authentication. Instead of requiring the user to frequently re-enter their password, the operating system could instead leverage principles of continuous authentication [80, 105] and session resumption. Research should examine how the operating system can make the user aware that continuous authentication is being used, and help users track which applications are still authenticated. Profoundly, with the end-to-end password paradigm, new authentication functionality, such as continuous authentication, can be added to any application without needing to modify the application.

Application supplied context. Currently, the application's functionality is limited to act as an intermediary for the operating system and the password verification service to execute the strong password protocol. Potentially, the application could also be allowed to provide additional context to the operating system regarding authentication. For example, the application could indicate which domain the application wishes to authenticate or which account the user needs to authenticate. Research should identify both what context would be helpful for the application to provide as well how to ensure that this provided context cannot be used to trick users into taking unintended actions.

Application initiated authentication. In our architecture, we require that the user—not the application—starts the authentication process. While this was done to best exemplify the end-to-end password paradigm, we also recognize that it might be possible and advantageous to allow the application to initiate authentication. Still, research would need to be done to understand what effect giving this additional responsibility to the application would have on the attack surface. Similarly, research could also examine a hybrid approach where both applications and the user can initiate authentication.

Multi-factor authentication. Multi-factor authentication (MFA) is entirely compatible with end-to-end passwords. The naïve approach would be to allow applications to handle the non-password factors, but it might also be possible to integrate the additional factors directly into the end-to-end encryption operating system component. For example, instead of storing salts at the password verification service, they could be derived using an MFA protocol. As the salt value is needed to execute the strong password protocol,

authentication would be blocked until MFA-authentication successfully completed. This approach could also reduce the information stored at the password verification service. Usability studies would be needed to identify which approach is best.

Hardware tokens. Hardware tokens could be used to further increase the security of end-to-end passwords. For example, a password vault could be stored in a hardware token, only being unlocked after the end-to-end password system verified the master password, providing simple two-factor authentication. The operating system could prevent any other application from detecting the presence of the second factor, reducing the risk of leaving the device attached to the computer. Alternatively, hardware tokens can also be used to address shoulder surfing and hardware-based key loggers [12].

Trustworthy Password Verification Service. While the end-to-end password paradigm relies on a secure local environment—the operating system—future research could attempt to further secure remote environment—the password verification service. For example, combining a hardware security module (HSM), a trusted platform module (TPM), and Intel's Software Guard Extensions [23] (SGX) could allow the user to verify that the password verification service was properly storing users' verifier tokens. Doing so would mean that passwords are fully protected during their entire lifecycle, maximizing the security gauntnesses available to password-based authentication by itself.

5.3 Transition

Currently, password entry is tightly integrated with applications, plaintext passwords submitted over TLS, and service providers failing to follow best practices for storing encrypted passwords. We envision a new password paradigm characterized by password entry that is tightly integrated with and under the control of the operating system, zero-knowledge password proofs where plaintext passwords are never sent to a service provider, and best practices for encrypted password storage being under the control of the user. The transition from this current state to a new password paradigm will not be immediate. Still, there are research opportunities that could accelerate this process and help incrementally increase the security of passwords.

Discover incentives for adoption. Research should be done to identify the incentives that will convince applications and password verification services to begin adopting the end-to-end password paradigm. While this might be a simple problem of network effects [64], it could be that other factors could also speed adoption. For example, adoption of OAuth is partially driven by the fact that it allows applications to gather basic information about the user (e.g., name, age, friends).

Early adoption pathways. Mobile phones provide a compelling first platform for deploying end-to-end passwords. Already, mobile operating systems have begun assisting applications with authenticating users—for example, using Touch ID to unlock applications on iOS. End-to-end passwords could be added to these mobile operating systems to provide a universal, standardized, and accessible method for users to enter their passwords, regardless of the application that authentication is taking place in. By also integrating password manager-like functionality, mobile operating systems could obviate the need for users to enter passwords, instead leveraging more appropriate authentication techniques (e.g., biometrics). Finally, because there would be a single password entry interface, mobile interfaces could more easily detect the password phishing interfaces that have recently begun to plague mobile platforms.[4]

Additionally, providing an end-to-end passwords Amazon Web Services (AWS) password verification service would allow developers to easily build new applications and modify existing application to support strong password protocols.[5] Such a service would have three primary benefits to developers. First, it would remove the need for developers to individually implement and maintain the password verification service, reducing the time it takes them to get their application to market and allowing more time to be focused on the application's core functionality. Second, it would automatically be updated to fix errors and better integrate with advances in authentication interfaces (i.e., end-to-end passwords). Third, it would allow the developer to announce that they were providing strong security for user's passwords. All of these reasons would be strong incentives for developers to adopt the AWS password verification service, accelerating the adoption of strong password protocols compatible with the end-to-end password paradigm.

Replace existing interfaces. The operating system could immediately begin to detect existing password interfaces. After detection, the operating system could overlay the existing interface [73, 91, 108] and use this overlay to require the user to enter their password using the operating system's interface. The entered password would then be transferred to the overlayed application and authentication would continue as normal. This would help habituate user to the correct process for entering their passwords even before applications have begun to adopt the end-to-end password paradigm.

Research should identify how to best detect and overlay existing interfaces. Additionally, care should be taken to gather sufficient contextual cues to ensure that the user knows which application they are authenticating too, helping them avoid phishing attacks.

Gradual password guidance. As the number of passwords that are entered through the operating system increases, the operating system will be able to give increasingly helpful guidance to the user. For example, the operating system could monitor user keystrokes and warn users when they enter a password out of the operating system that they had previously entered in the operating system. This could help users recognize when they accidentally divulged a sensitive password that should have been entered in the operating system, similar to what is done by Google's Password Alert [44]. Alternatively, the operating system could begin advising the user of weaknesses in their password selection strategies and also identify instances of password reuse, similar to what is beginning to be done by LastPass [72].

User training. Users are very habituated to the current password paradigm. During the transition to end-to-end passwords, it will be essential to properly train users on the new paradigm. Specifically, users need to be taught how to ensure that the transitional operating system interface (described above) is used to enter

[4] https://motherboard.vice.com/en_us/article/ne7gxz/ios-iphone-password-phishing-app-popups

[5] Such a service would not need to be a single sign-on service, but could properly store credentials for each application local to that application.

all passwords; this is needed to protect users from phishing sites that claim they do not yet support end-to-end passwords. Similarly, users need to be assisted in building correct mental models for end-to-end passwords. Importantly, research will need to be done to explore how this training can happen in-line with users' tasks, and without significantly disrupting their workflow.

6 SUMMARY

In this paper, we describe a new paradigm for password-based authentication—end-to-end passwords. In this paradigm, password responsibilities are pushed to the endpoints of password-based authentication—the operating system and the password verification service. Of these two endpoints, most responsibilities are concentrated in the operating system.

The benefits of this paradigm are five-fold. First, it significantly reduces the attack surface for password-based authentication. Second, it has the potential to finally make a significant impact on password phishing. Third, it offers protection from a compromised host, which is outside the threat model of other approaches. Fourth, it supports the consistent enforcement of password policies across all of a user's accounts. Finally, concentrating password responsibilities in the operating system makes it easier to diffuse improvements rapidly.

This paper identifies open research questions related to end-to-end passwords. Specifically, we note the importance that the development of this paradigm be guided by systematic application of user-centered design principles and empirical usability analysis. As such, there are a number of usability issues to be addressed in this new paradigm. There are also a number of systems research questions to be addressed to increase the capabilities and strengthen the security of this new paradigm. Lastly, we briefly describe the research needed to enable the transition from the current password paradigm to the end-to-end password paradigm.

7 ACKNOWLEDGMENT

The authors thank Trevor Smith and Ken Reese for helpful feedback on an early draft of the paper. The authors also thank the anonymous reviewers and the workshop attendees for their helpful feedback that strengthened the paper.

REFERENCES

[1] M. Abdalla, O. Chevassut, P.A. Fouque, and D. Pointcheval. 2005. A Simple Threshold Authenticated Key Exchange from Short Secrets. *Proceedings of the Twenty-Fourth International Conference on the Theory and Application of Cryptographic Techniques* (2005), 566–584.

[2] M. Abdalla, P.A. Fouque, and D. Pointcheval. 2005. Password-based Authenticated Key Exchange in the Three-Party Setting. *Proceedings of the Seventh International Workshop on Theory and Practice in Public Key Cryptography* (2005), 65–84.

[3] M. Abdalla, M. Izabachène, and D. Pointcheval. 2008. Anonymous and Transparent Gateway-Based Password-Authenticated Key Exchange. *Proceedings of the Sixth International Conference on Applied Cryptography and Network Security* (2008), 133–148.

[4] P.E. Abi-Char, A. Mhamed, and B. El-Hassan. 2007. A Fast and Secure Elliptic Curve Based Authenticated Key Agreement Protocol for Low Power Mobile Communications. In *Proceedings of the The 2007 International Conference on Next Generation Mobile Applications, Services and Technologies*. IEEE, 235–240.

[5] Chadia Abras, Diane Maloney-Krichmar, and Jenny Preece. 2004. User-Centered Design. *Bainbridge W. Encyclopedia of Human-Computer Interaction* 37, 4 (2004), 445–456.

[6] P.S. Aleksic and A.K. Katsaggelos. 2006. Audio-Visual Biometrics. (2006), 2025–2044 pages.

[7] Mohammed H Almeshekah, Christopher N Gutierrez, Mikhail J Atallah, and Eugene H Spafford. 2015. Ersatzpasswords: Ending Password Cracking and Detecting Password Leakage. In *Proceedings of the Thirty-First Annual Computer Security Applications Conference*. ACM, 311–320.

[8] Fadi Aloul, Syed Zahidi, and Wassim El-Hajj. 2009. Two Factor Authentication Using Mobile Phones. In *Proceedings of the Seventh IEEE/ACS International Conference on Computer Systems and Applications*. IEEE, 641–644.

[9] Shiri Azenkot, Kyle Rector, Richard Ladner, and Jacob Wobbrock. 2012. Passchords: Secure Multi-Touch Authentication for Blind People. In *Proceedings of the Fourteenth International ACM Conference on Computers and Accessibility*. ACM, 159–166.

[10] S.M. Bellovin and M. Merritt. 1992. Encrypted Key Exchange: Password-Based Protocols Secure against Dictionary Attacks. In *Proceedings of the Thirteenth IEEE Symposium on Security and Privacy*. IEEE, 72–84.

[11] S.M. Bellovin and M. Merritt. 1993. Augmented Encrypted Key Exchange: A Password-Based Protocol Secure against Dictionary Attacks and Password File Compromise. In *Proceedings of the First ACM Conference on Computer and Communications Security*. ACM, 244–250.

[12] J. Bonneau, C. Herley, P.C. van Oorschot, and F. Stajano. 2012. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *Proceedings of the Thirty-Third IEEE Symposium on Security and Privacy*. IEEE, 553–567.

[13] Victor Boyko, Philip MacKenzie, and Sarvar Patel. 2000. Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. In *Proceedings of the Nineteenth International Conference on the Theory and Application of Cryptographic Techniques*. Springer, 156–171.

[14] J. Brainard, A. Juels, R.L. Rivest, M. Szydlo, and M. Yung. 2006. Fourth-Factor Authentication: Somebody You Know. In *Proceedings of the Thirteenth ACM Conference on Computer and Communications Security*, Vol. 30. 168–178.

[15] S. Brands. 2007. The Identity Corner—The Problem(s) with OpenId. https://idcorner.org/2007/08/22/the-problems-with-openid/. (2007). Accessed 2017/04/14.

[16] J.W. Byun, D.H. Lee, and J.I. Lim. 2006. Security Analysis and Improvement of a Gateway-Oriented Password-Based Authenticated Key Exchange Protocol. *IEEE Communications Letters* 10, 9 (2006), 683–685.

[17] T.Y. Chang, M.S. Hwang, and W.P. Yang. 2011. A Communication-Efficient Three-Party Password Authenticated Key Exchange Protocol. *Information Sciences* 181, 1 (2011), 217–226.

[18] S. Chiasson, R. Biddle, and P.C. van Oorschot. 2007. A Second Look at the Usability of Click-Based Graphical Passwords. In *Proceedings of the Third Symposium on Usable Privacy and Security*. ACM, 1–12.

[19] S. Chiasson, E. Stobert, A. Forget, R. Biddle, and P.C. Van Oorschot. 2012. Persuasive Cued Click-Points: Design, Implementation, and Evaluation of a Knowledge-Based Authentication Mechanism. *IEEE Transactions on Dependable and Secure Computing* 9, 2 (2012), 222–235.

[20] S. Chiasson, P.C. van Oorschot, and R. Biddle. 2006. A Usability Study and Critique of Two Password Managers. In *Proceedings of the Fifteenth USENIX Security Symposium*. USENIX, 1–16.

[21] T Charles Clancy, Negar Kiyavash, and Dennis J Lin. 2003. Secure Smartcard-based Fingerprint Authentication. In *Proceedings of the 2003 ACM Workshop on Biometrics Methods and Applications*. ACM, 45–52.

[22] Y. Cliff, Y. Tin, and C. Boyd. 2006. Password Based Server Aided Key Exchange. In *Proceedings of the Fourth International Conference on Applied Cryptography and Network Security*. Springer, 146–161.

[23] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016 (2016), 86.

[24] J. Daugman. 2004. How Iris Recognition Works. *IEEE Transactions on Circuits and Systems for Video Technology* 14, 1 (2004), 21–30.

[25] Derek L Davis. 1999. Secure Boot. (Aug. 1999). US Patent 5,937,063.

[26] R. Dhamija. 2007. Security Usability Studies: Risk, Roles and Ethics. In *Proceedings of 2017Workshop on Security User Studies*.

[27] R. Dhamija and L. Dusseault. 2008. The Seven Flaws of Identity Management: Usability and Security Challenges. *IEEE Security and Privacy* 6, 2 (2008), 24–29.

[28] R. Dhamija, J.D. Tygar, and M. Hearst. 2006. Why Phishing Works. In *Proceedings of the Eighteenth ACM Conference on Human Factors in Computing Systems*. ACM, 581–590.

[29] Rachna Dhamija and J Doug Tygar. 2005. The Battle Against Phishing: Dynamic Security Skins. In *Proceedings of the First Symposium on Usable Privacy and Security*. ACM, 77–88.

[30] Ben Dodson, Debangsu Sengupta, Dan Boneh, and Monica S Lam. 2012. Secure, Consumer-Friendly Web Authentication and Payments with a Phone. *Mobile Computing, Applications, and Services* (2012), 17–38.

[31] Bryan Dosono, Jordan Hayes, and Yang Wang. 2015. "I'm Stuck!": A Contextual Inquiry of People with Visual Impairments in Authentication. In *Proceedings of the Eleventh Symposium on Usable Privacy and Security*. 151–168.

[32] S. Drimer, S. Murdoch, and R. Anderson. 2009. Optimised to Fail: Card Readers for Online Banking. *Proceedings of the Thirteenth International Conference on Financial Cryptography and Data Security* (2009), 184–200.

[33] Jeremy Epstein, John McHugh, Hilarie Orman, Rita Pascale, Ann Marmor-Squires, Bonnie Danner, Charles R Martin, Martha Branstad, Glenn Benson, and Doug Rothnie. 1993. A High Assurance Window System Prototype. *Journal of Computer Security* 2, 2-3 (1993), 159–190.

[34] Facebook. 2017. Facebook Connect. https://developers.facebook.com/blog/post/2008/05/09/announcing-facebook-connect/. (2017). Accessed 2017/04/14.

[35] Tao Feng, Ziyi Liu, Kyeong-An Kwon, Weidong Shi, Bogdan Carbunar, Yifei Jiang, and Nhung Nguyen. 2012. Continuous Mobile Authentication Using Touchscreen Gestures. In *Proceedings of the 2012 IEEE Conference on Technologies for Homeland Security*. IEEE, 451–456.

[36] D. Florêncio and C. Herley. 2008. One-time Password Access to Any Server without Changing the Server. *Proceedings of the Eleventh International Conference on Information Security* (2008), 401–420.

[37] Dinei Florêncio, Cormac Herley, and Paul C Van Oorschot. 2014. An Administrator's Guide to Internet Password Research. In *Proceedings of the Twenty-Eighth Large Installation System Administration Conference*. 35–52.

[38] OpenID Foundation. 2017. OpenId. http://openid.net/. (2017). Accessed 2017/04/14.

[39] OpenID Foundation. 2017. OpenId Foundation. http://openid.net/foundation/. (2017). Accessed 2017/04/14.

[40] Davrondzhon Gafurov. 2007. A Survey of Biometric Gait Recognition: Approaches, Security and Challenges. In *Annual Norwegian Computer Science Conference*. 19–21.

[41] Simson Garfinkel and Heather Richter Lipford. 2014. Usable Security: History, Themes, and Challenges. *Synthesis Lectures on Information Security, Privacy, and Trust* 5, 2 (2014), 1–124.

[42] Virgil D. Gligor, C. Sekar Chandersekaran, Robert S. Chapman, Leslie J. Dotterer, MS Hetch, Wen-Der Jiang, Abhai Johri, Gary L. Luckenbaugh, and N Vasudevan. 1987. Design and Implementation of Secure Xenix. *IEEE Transactions on Software Engineering* 2 (1987), 208–221.

[43] Google. 2017. Google 2-Step Verification. https://www.google.com/landing/2step/. (2017). Accessed 2017/04/14.

[44] Google. 2017. Password Alert. https://github.com/google/password-alert. (2017). Accessed 2017/04/14.

[45] Anti-Phishing Working Group. 2008. Phishing Activity Trends: Report for the Month of January, 2008. (2008).

[46] Guodong Guo, Stan Z Li, and Kapluk Chan. 2000. Face Recognition by Support Vector Machines. In *Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition*. IEEE, 196–201.

[47] N. Haller, C. Metz, P. Nesser, and M. Straw. 1996. A One-Time Password System. http://www.ietf.org/rfc/rfc1938.txt. (1996).

[48] E. Hammer-Lahav, D. Recordon, and D. Hardt. 2011. The OAuth 2.0 Authorization Protocol. (2011).

[49] M. Hanson, D. Mills, and B. Adida. 2011. Federated Browser-Based Identity Using Email Addresses. In *Proceedings of the 2011 W3C Workshop on Identity in the Browser*.

[50] F. Hao and P. Ryan. 2011. Password Authenticated Key Exchange by Juggling. *Proceedings of the Nineteenth International Workshop on Security Protocols* (2011), 159–171.

[51] C. Herley. 2009. So Long, and No Thanks for the Externalities: The Rational Rejection of Security Advice by Users. In *Proceedings of the 2009 New Security Paradigms Workshop*. ACM, 133–144.

[52] C. Herley and P. Van Oorschot. 2012. A Research Agenda Acknowledging the Persistence of Passwords. *Proceedings of the Thirty-Third IEEE Symposium on Security and Privacy* 10, 1 (2012), 28–36.

[53] C. Herley and P. Van Oorschot. 2017. SoK: Science, Security, and the Elusive Goal of Security As a Scientific Pursuit. *Proceedings of the Thirty-Eighth IEEE Symposium on Security and Privacy* (2017).

[54] N. Hopper and M. Blum. 2001. Secure Human Identification Protocols. *Proceedings of the Twentieth International Conference on the Theory and Application of Cryptographic Techniques* (2001), 52–66.

[55] Philip G Inglesant and M Angela Sasse. 2010. The True Cost of Unusable Password Policies: Password Use in the Wild. In *Proceedings of the Twenty-Second ACM Conference on Human Factors in Computing Systems*. ACM, 383–392.

[56] IronKey. 2017. IronKey. http://www.ironkey.com/. (2017). Accessed 2017/04/14.

[57] D.P. Jablon. 1996. Strong Password-Only Authenticated Key Exchange. *ACM SIGCOMM Computer Communication Review* 26, 5 (1996), 5–26.

[58] D.P. Jablon. 1997. Extended Password Key Exchange Protocols Immune to Dictionary Attack. In *Proceedings of the Sixth IEEE workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. IEEE, 248–255.

[59] M. Jakobsson, L. Yang, and S. Wetzel. 2008. Quantifying the Security of Preference-Based Authentication. In *Proceedings of the Fourth ACM Workshop on Digital Identity Management*. ACM, 61–70.

[60] R. Jhawar, P. Inglesant, N. Courtois, and M.A. Sasse. 2011. Make Mine a Quadruple: Strengthening the Security of Graphical One-Time PIN Authentication. In *Proceedings of the Fifth IEEE International Conference on Network and System Security*. IEEE, 81–88.

[61] Nikolaos Karapanos, Claudio Marforio, Claudio Soriente, and Srdjan Capkun. 2015. Sound-Proof: Usable Two-Factor Authentication Based on Ambient Sound. In *Proceedings of the Twenty-Fourth USENIX Security Symposium*. 483–498.

[62] C. Karlof, JD Tygar, and D. Wagner. 2009. Conditioned-Safe Ceremonies and a User Study of an Application to Web Authentication. In *Proceedings of the Fifteenth Network and Distributed System Security Symposium*, Vol. 9.

[63] Ambarish Karole, Nitesh Saxena, and Nicolas Christin. 2010. A Comparative Usability Evaluation of Traditional Password Managers. In *Proceedings of the Thirteenth International Conference on Information Security and Cryptology*. Springer, 233–251.

[64] Michael L Katz and Carl Shapiro. 1994. Systems Competition and Network Effects. *The Journal of Economic Perspectives* 8, 2 (1994), 93–115.

[65] Patrick Gage Kelley, Saranga Komanduri, Michelle L Mazurek, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Julio Lopez. 2012. Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-cracking Algorithms. In *Proceedings of the Thirty-Third IEEE Symposium on Security and Privacy*. IEEE, 523–537.

[66] K. Kobara and S.H. Shin. 2012. Efficient Augmented Password-Only Authentication and Key Exchange for IKEv2. (2012).

[67] D.P. Kormann and A.D. Rubin. 2000. Risks of the Passport Single Signon Protocol. *Computer Networks* 33, 1 (2000), 51–58.

[68] Ravi Kuber and Shiva Sharma. 2010. Toward Tactile Authentication for Blind Users. In *Proceedings of the Twelfth International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, 289–290.

[69] M. Kuhn. 1998. OTPW—A One-time Password Login Package. (1998).

[70] J.O. Kwon, I.R. Jeong, K. Sakurai, and D.H. Lee. 2007. Efficient Verifier-Based Password-Authenticated Key Exchange in the Three-Party Setting. *Computer Standards and Interfaces* 29, 5 (2007), 513–520.

[71] T. Kwon. 2001. Authentication and Key Agreement Via Memorable Password. In *Proceedings of the Seventh Network and Distributed System Security Symposium*, Vol. 20. Internet Society, 31–33.

[72] LastPass. 2017. LastPass. https://lastpass.com/. (2017). Accessed 2017/04/14.

[73] Billy Lau, Simon P Chung, Chengyu Song, Yeongjin Jang, Wenke Lee, and Alexandra Boldyreva. 2014. Mimesis Aegis: A Mimicry Privacy Shield-a System's Approach to Data Privacy on Public Cloud. In *Proceedings of the Twenty-Third USENIX Security Symposium*. USENIX, 33–48.

[74] B. Laurie. 2007. OpenId Phishing Heaven. http://www.links.org/?p=187. (2007). Accessed 2017/04/14.

[75] Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Song. 2014. The Emperor's New Password Manager: Security Analysis of Web-Based Password Managers.. In *Proceedings of the Twenty-Third USENIX Security Symposium*. 465–479.

[76] P.D. MacKenzie. 2002. The PAK Suite: Protocols for Password-Authenticated Key Exchange. (2002), 2 pages.

[77] M. Mannan and PC van Oorschot. 2011. Leveraging Personal Devices for Stronger Password Authentication from Untrusted Computers. *Journal of Computer Security* 19, 4 (2011), 703–750.

[78] C. Messina. 2009. OpenId Phishing Brainstorm. (2009).

[79] B Clifford Neuman and Theodore Ts'o. 1994. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine* 32, 9 (1994), 33–38.

[80] Koichiro Niinuma, Unsang Park, and Anil K Jain. 2010. Soft Biometric Traits for Continuous User Authentication. *IEEE Transactions on information forensics and security* 5, 4 (2010), 771–780.

[81] Mark O'Neill, Scott Ruoti, Kent Seamons, and Daniel Zappala. 2016. TLS Proxies: Friend or Foe?. In *Proceedings of the 2016 ACM on Internet Measurement Conference*. ACM, 551–557.

[82] B. Parno, C. Kuo, and A. Perrig. 2006. Phoolproof Phishing Prevention. *Proceedings of the Fourteenth International Workshop on Security Protocols* (2006), 1–19.

[83] A. Pashalidis and C.J. Mitchell. 2004. Impostor: A Single Sign-On System for Use from Untrusted Devices. In *Proceedings of the Fifth IEEE Global Telecommunications Conference*, Vol. 4. IEEE, 2191–2195.

[84] PassWindow. 2017. PassWindow. http://www.passwindow.com/. (2017). Accessed 2017/04/14.

[85] Lili Qiu, Yin Zhang, Feng Wang, Mi Kyung, and Han Ratul Mahajan. 1985. Trusted Computer System Evaluation Criteria. In *National Computer Security Center*. Citeseer.

[86] D. Recordon and D. Reed. 2006. OpenId 2.0: A Platform for User-Centric Identity Management. In *Proceedings of the Second ACM Workshop on Digital Identity Management*. ACM, 11–16.

[87] A. Ross, J. Shah, and A.K. Jain. 2007. From Template to Image: Reconstructing Fingerprints from Minutiae Points. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29, 4 (2007), 544–560.

[88] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J.C. Mitchell. 2005. Stronger Password Authentication Using Browser Extensions. In *Proceedings of the Fourteenth USENIX Security Symposium*, Vol. 5. USENIX.

[89] RSA. 2017. RSA SecureID. (2017). Accessed 2017/04/14.

[90] Scott Ruoti, Jeff Andersen, and Kent Seamons. 2016. Strengthening Password-Based Authentication. In *Proceedings of the Second Who Are You?! Adventures in Authentication Workshop at the Symposium on Usable Privacy and Security*.

[91] Scott Ruoti, Nathan Kim, Ben Burgon, Timothy Van Der Horst, and Kent Seamons. 2013. Confused Johnny: When Automatic Encryption Leads to Confusion and Mistakes. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*. ACM, Newcastle, United Kingdom.

[92] V. Samar and R. Schemers. 1995. RFC 86.0: Unified Login with Pluggable Authentication Modules (PAM). http://www.kernel.org/pub/linux/libs/pam/pre/doc/rfc86.0.txt.gz. (1995).

[93] S Sanderson and JH Erbetta. 2000. Authentication for Secure Environments Based on Iris Scanning Technology. (2000).

[94] S. Schechter, A.J.B. Brush, and S. Egelman. 2009. It's No Secret—Measuring the Security and Reliability of Authentication Via "Secret" Questions. In *Proceedings of the Thirtieth IEEE Symposium on Security and Privacy*. IEEE, 375–390.

[95] Stuart E Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. 2007. The Emperor's New Security Indicators. In *Proceedings of the Twenty-Eighth IEEE Symposium on Security and Privacy*. IEEE, 51–65.

[96] David Silver, Suman Jana, Dan Boneh, Eric Yawei Chen, and Collin Jackson. 2014. Password Managers: Attacks and Defenses. In *Proceedings of the Twenty-Third USENIX Security Symposium*. 449–464.

[97] M. Slot. 2008. Beginner's Guide to OpenId Phishing. https://blog.rootshell.be/2008/11/05/beginners-guide-to-openid-phishing/. (2008). Accessed 2017/04/14.

[98] S.L. Smith. 1987. Authenticating Users by Word Association. *Computers and Security* 6, 6 (1987), 464–470.

[99] F. Stajano. 2011. PICO: No More Passwords! *Proceedings of the Nineteenth International Workshop on Security Protocols* (2011), 49–81.

[100] S.T. Sun. 2012. Simple but Not Secure: An Empirical Security Analysis of OAuth 2.0-Based Single Sign-On Systems. (2012).

[101] S.T. Sun, Y. Boshmaf, K. Hawkey, and K. Beznosov. 2010. A Billion Keys, but Few Locks: The Crisis of Web Single Sign-On. In *Proceedings of the 2010 New Security Paradigms Workshop*. ACM, 61–72.

[102] S.T. Sun, K. Hawkey, and K. Beznosov. 2010. OpenId-Enabled Browser: Towards Fixing the Broken Web Single Sign-On Triangle. In *Proceedings of the Sixth ACM Workshop on Digital Identity Management*. ACM, 49–58.

[103] S.T. Sun, E. Pospisil, I. Muslukhov, N. Dindar, K. Hawkey, and K. Beznosov. 2011. What Makes Users Refuse Web Single Sign-On?: An Empirical Investigation of OpenId. In *Proceedings of the Seventh Symposium on Usable Privacy and Security*. ACM, 4.

[104] H. Tao. 2006. *Pass-Go—A New Graphical Password Scheme*. Ph.D. Dissertation. University of Ottawa.

[105] Issa Traore. 2011. *Continuous Authentication Using Biometrics: Data, Models, and Metrics: Data, Models, and Metrics*. IGI Global.

[106] Blase Ur, Patrick Gage Kelley, Saranga Komanduri, Joel Lee, Michael Maass, Michelle L Mazurek, Timothy Passaro, Richard Shay, Timothy Vidas, Lujo Bauer, and others. 2012. How Does Your Password Measure Up? the Effect of Strength Meters on Password Creation.. In *Proceedings of the Twenty-First USENIX Security Symposium*. 65–80.

[107] T.W. van der Horst and K.E. Seamons. 2007. Simple Authentication for the Web. In *Proceedings of the Third International IEEE Conference on Security and Privacy in Communications Networks and the Workshops*. IEEE, 473–482.

[108] Timothy W. van der Horst and Kent Eldon Seamons. 2009. Encrypted Email Based upon Trusted Overlays. (March 2009). US Patent 8,521,821.

[109] Vasco. 2017. Cronto. http://www.cronto.com/. (2017). Accessed 2017/04/14.

[110] Amit Vasudevan, Bryan Parno, Ning Qu, Virgil D Gligor, and Adrian Perrig. 2009. *Lockdown: A Safe and Practical Environment for Security Applications*. Technical Report. Carnegie Mellon University.

[111] James Wayman, Anil Jain, Davide Maltoni, and Dario Maio. 2005. An Introduction to Biometric Authentication Systems. *Biometric Systems* (2005), 1–20.

[112] F. Wei, Z. Zhang, and C. Ma. 2011. Gateway-Oriented Password-Authenticated Key Exchange Protocol in the Standard Model. *Journal of Systems and Software* (2011).

[113] D. Weinshall. 2006. Cognitive Authentication Schemes Safe against Spyware. In *Proceedings of the Twenty-Seventh IEEE Symposium on Security and Privacy*. IEEE, 6–pp.

[114] Matt Weir, Sudhir Aggarwal, Michael Collins, and Henry Stern. 2010. Testing Metrics for Password Creation Policies by Attacking Large Sets of Revealed Passwords. In *Proceedings of the Seventeenth ACM Conference on Computer and Communications Security*. ACM, 162–175.

[115] H.A. Wen, T.F. Lee, and T. Hwang. 2005. Provably Secure Three-Party Password-Based Authenticated Key Exchange Protocol Using Weil Pairing. (2005), 138–143 pages.

[116] Alexander Wiesmaier, Marcus Fischer, Evangelos G. Karatsiolis, and Marcus Lippert. 2004. Outflanking and securely using the PIN/TAN-System. (2004). http://arxiv.org/abs/cs.CR/0410025

[117] Min Wu, Robert C Miller, and Greg Little. 2006. Web Wallet: Preventing Phishing Attacks by Revealing User Intentions. In *Proceedings of the Second Symposium on Usable Privacy and Security*. ACM, 102–113.

[118] Thomas Wu. 1998. The Secure Remote Password Protocol. In *Proceedings of the Fourth Network and Distributed System Security Symposium*, Vol. 98. Internet Society, 97–111.

[119] T. Wu. 2002. SRP-6: Improvements and Refinements to the Secure Remote Password Protocol. (2002).

[120] T.C. Wu and H.Y. Chien. 2009. Comments on Gateway-Oriented Password-based Authenticated Key Exchange Protocol. In *Proceedings of the Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. IEEE, 262–265.

[121] YubiCo. 2017. YubiKey. http://www.yubico.com/products/yubikey-hardware/yubikey/. (2017). Accessed 2017/04/14.

[122] Yinqian Zhang, Fabian Monrose, and Michael K Reiter. 2010. The Security of Modern Password Expiration: An Algorithmic Framework and Empirical Analysis. In *Proceedings of the Seventeenth ACM Conference on Computer and Communications Security*. ACM, 176–186.