Secure Passwords Through Enhanced Hashing

Benjamin Strahs   Chuan Yue   Haining Wang
Department of Computer Science
The College of William and Mary
Williamsburg, VA 23187, USA
{*bgstra,cyue,hnw*}@*cs.wm.edu*

Abstract

Passwords play a critical role in online authentication. Unfortunately, passwords suffer from two seemingly intractable problems: password cracking and password theft. In this paper, we propose *PasswordAgent*, a new password hashing mechanism that utilizes both a salt repository and a browser plug-in to secure web logins with strong passwords. Password hashing is a technique that allows users to remember simple low-entropy passwords and have them hashed to create high-entropy secure passwords. PasswordAgent generates strong passwords by enhancing the hash function with a large random salt. With the support of a salt repository, it gains a much stronger security guarantee than existing mechanisms. PasswordAgent is less vulnerable to offline attacks, and it provides stronger protection against password theft. Moreover, PasswordAgent offers some usability advantages over existing hash-based mechanisms, while maintaining users' familiar password entry paradigm. We build a prototype of PasswordAgent and conduct usability experiments.

1 Introduction

Passwords remain the most common security method to authenticate or verify a user's online identity [25]. They provide a powerful guard against unauthorized access to systems and data, and are ubiquitously used in various online activities such as shopping, banking, communication, and learning. User authentication via password relies on the *something you know* authentication factor, i.e., you know some secret that no one else does. Although two other authentication factors *something you have* (e.g., hardware token) and *something you are* (e.g., fingerprint) have also been recognized and used in practice, they have not gained a wide acceptance on the Internet, primarily because of their high cost, limited flexibility, and restricted portability. On the contrary, passwords are simple, inexpensive, easy to implement, and convenient to use. Consequently, they occupy the dominant position in online user authentication, and this situation will not change in the foreseeable future.

Despite their prevalence and importance in online authentication, passwords do have two well-known and long-standing problems: weak passwords are easy to crack, and passwords are vulnerable to theft. Password security depends on creating strong passwords and protecting them from being stolen. A strong password should be sufficiently long, random, and hard to discover by crackers. In contrast, a weak password is usually short, common, and easy to guess. Examples of strong passwords include "t3wahSetyeT4" and "Tpftcits4Utg!"; and examples of weak passwords include "susan123" and "password" [6]. Weak passwords suffer from vulnerability to brute-force and dictionary attacks [29]. The dilemma in a password system is that a user will often choose guessable passwords simply because they are easy to remember [11, 19, 29]. Moreover, no matter how strong they are, passwords are also vulnerable to theft. One of the most significant threats to online users is *phishing* attack [18, 39], which uses social engineering techniques to steal users' personal identity data on spoofed websites [1]. In recent years, this type of identity theft has risen sharply and has cost billion-dollar losses [2]. Other attacks like *shoulder surfing* [33] can also steal user passwords, especially in public places such as cybercafes, airports, and libraries.

As more online services are password-protected, users have to create and memorize an increasing number of passwords. This, combined with the inherent limitation of human memory, forces users to revert back to insecure habits such as choosing simpler passwords, reusing passwords across different websites, or even writing down their passwords [37]. A recent large-scale study of web password usage shows that on average, a user has approximately 6.5 passwords shared across 25 different websites, and the majority of users choose weak passwords that contain only lower case letters [22].

To address these problems and enhance online password security, a number of techniques have been proposed. For example, *password managers* generate strong passwords and automatically store them in a local database [8, 5]. *Single sign-on* systems allow users to log into many websites using one account, which reduces the number of passwords a user must remember [9]. *Graphical passwords* enable users to click on images to authenticate themselves [26, 17]. However, these solutions all have their own limitations. *Password managers* store passwords on a fixed computer and thus lack mobility; *single sign-on* systems place too much trust on a centralized system and thus are vulnerable to single-point failure [27]; and *graphical passwords*, although proposed as an alternative to traditional text-based passwords, are still hampered by security and usability concerns [16, 34].

A promising approach to obtaining secure online passwords is *password hashing*, in which hashed passwords are sent to remote websites instead of plain-text passwords. Password hashing is very attractive for a few reasons: it is lightweight and convenient to use, increases password strength, and can defend against phishing attacks. This approach has been taken in projects such as the Lucent Personal Web Assistant (LPWA) [23], PwdHash [31], Password Multiplier [24], and Passpet [38]. However, these systems still have security limitations which will be discussed in Section 2. Moreover, password hashing systems, if not carefully designed and implemented, suffer from usability problems that may directly lead to security exposures [14].

In this paper, we present *PasswordAgent*, an automatic password management system with enhanced hashing, which consists of a Salt Repository server and a browser plug-in Agent for securing online passwords. The Salt Repository stores a list of salts for each registered user while the Agent provides the user interface, salt retrieval, and hashing functionality. When a plain-text password needs to be protected for a specific website, the user simply activates the Agent and enters the plain-text password. The Agent automatically concatenates the plain-text password and the website specific salt to deterministically generate the site password via a hash function.

The contribution of PasswordAgent to online password management lies in the following aspects. First, it automatically provides a stronger security guarantee by using randomly generated and securely stored salts. Second, it improves phishing protection by giving users accurate warnings if they attempt to enter protected account information on an unprotected website. Moreover, even if phishers obtain the plain-text passwords by using subtle techniques such as JavaScript attacks or "spoofed password field in Flash" [31], they still cannot access users' accounts because they do not have the salts. Third, as long as the password to the Salt Repository server is not observed by an attacker, PasswordAgent also reduces the risks of shoulder surfing attacks. Finally, a few usability suggestions made in [14] are incorporated into PasswordAgent, providing some usability advantages over existing solutions.

The remainder of this paper is structured as follows. We describe existing password hashing solutions in Section 2. We present the design of PasswordAgent in Section 3 and analyze its security and usability in Section 4. We detail the implementation and usability evaluation of PasswordAgent in Section 5. We discuss the limitations of PasswordAgent in Section 6, and finally we conclude in Section 7.

2 Related Work

In this section, we highlight the contributions of the Lucent Personal Web Assistant (LPWA) [23] and three recent systems: PwdHash [31], Password Multiplier [24], and Passpet [38]. These existing systems exemplify the concept and value of password hashing in online user authentication, and they are most related to our proposed PasswordAgent.

LPWA is an HTTP proxy providing data anonymity services to users. To a user, LPWA generates secure, consistent, and pseudonymous usernames, passwords, and email aliases for different websites based on three inputs: a UserID, a universal password to the proxy, and a destination website address. Using LPWA, users can protect their real identities and weed out junk email based on the recipient email address. LPWA was successful before, but now it has serious limitations. LPWA does not support HTTPS, but the identities that need to be protected the most are those that are transmitted via HTTPS. LPWA also requires users to fully trust the proxy server, which knows all the login credentials to the destination servers, resulting in security and privacy concerns.

PwdHash is a browser extension that transparently creates a different password for each site, improving web password security and defending against phishing attacks. PwdHash addresses a few challenges of implementing password hashing as a secure and transparent extension to web browsers. In particular, PwdHash uses the destination domain name as a salt and sends a hashed password to the remote site. However, PwdHash, as acknowledged by the authors, is vulnerable to two major kinds of attacks. One is a dictionary attack on the hashed passwords. This vulnerability is due to three factors: a phishing site can obtain hashed passwords, PwdHash uses MD5 [30], a very fast hashing algorithm, and the salt is publicly known. The second vulnerability of Pwd-

Hash is its susceptibility to advanced phishing attacks, such as using Flash objects or focus stealing. Flash objects and focus stealing are a form of phishing that allows keyboard strokes to be intercepted before other browser plugins have a chance to handle them.

As a browser extension, Password Multiplier can generate strengthened passwords for an arbitrary number of accounts while requiring the user to memorize only a single short password. It uses the same three inputs as LPWA: a UserID, a master password, and a destination domain name. The key contribution of Password Multiplier is using a strengthened hash function to deterministically generate high-entropy passwords. However, the main problem with Password Multiplier is that all the derived passwords will be known to attackers if the master password is stolen. At present, it is possible for an attacker to steal a master password through a keylogger or other spyware. Moreover, changing a password for a specific site is complicated because Password Multiplier requires users to remember additional information. Changing the master password also becomes tedious because the password on every site needs to be updated.

Built upon Password Multiplier and Petname Tool [7], Passpet turns a single master password into distinct passwords for different websites and uses petnames to help users recognize phishing attempts. In order to generate correct passwords, Passpet relies on a remote server to store site label files. However, Passpet has the same drawback as Password Multiplier in terms of master password vulnerability. Changing the master password is still tedious because a user needs to migrate passwords for every site. In addition, its remote storage server is vulnerable to various malicious attacks, which is acknowledged by the authors. We compare PasswordAgent with these existing systems and detail the advantages of PasswordAgent in Section 4.

3 Design of PasswordAgent

3.1 Overview

PasswordAgent consists of two major components: the Salt Repository and the Agent. The Repository stores salt lists enabling PasswordAgent to function transparently across either enterprise networks or the Internet. The Agent is used to retrieve the salts from the Repository, provide visual security indicators, and generate site specific passwords. In our design, each enterprise network maintains a Salt Repository providing salt storage services for its users. To achieve high reliability and scalability, it is possible that multiple servers function as the Salt Repository within one enterprise network. Usually, each user has a primary Salt Repository, but it is possible that one user has salt lists stored in multiple repositories.

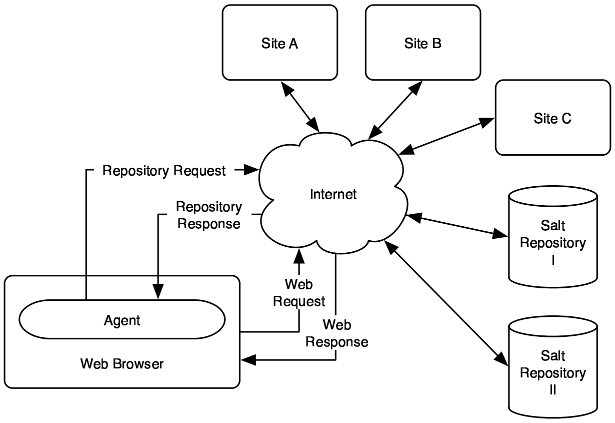


Figure 1: The architecture of PasswordAgent.

In contrast, the Agent is associated with each individual web browser as a browser extension. The basic architecture of the PasswordAgent is shown in Figure 1. Before continuing, it is important to have a grasp of the five different types of passwords discussed in this paper. Table 1 describes these passwords in detail.

Term	Description	Example
Plain-text Password	The user's password.	"secret"
Protected Password	The plain-text password but with additional data (either the activation hotkey or activation prefix) added to notify PasswordAgent that a site password needs to be generated.	"@@secret" OR [F2]"secret"
Site Password	The unique password generated for a site based on the site salt and plain-text password.	"2T7fYe10"
Agent Password	A password chosen by the user to protect their salts. It is only entered at the beginning of the browser session.	"likk@#0"
Repository Password	A password automatically generated from the Agent Password and used to authenticate to the Salt Repository.	"LT8@!dbn9"

Table 1: Password terminology.

To facilitate the deployment of the Salt Repository inside an enterprise network, the Repository can be integrated with any accessible web service that implements the Repository Interface. The web server can be publicly accessed via the Internet so that users can retrieve their salt lists from any location. The interface is a simple XML protocol that allows a user to register an account, save a domain and its associated salt, and retrieve a list of domains and salts. All the information in the salt list is encrypted before being stored. This not only guards against a compromised Repository, but also alleviates privacy concerns by making the domains inaccessible to anyone but the user. An overview of the data stored by the Salt Repository is illustrated in Figure 2.

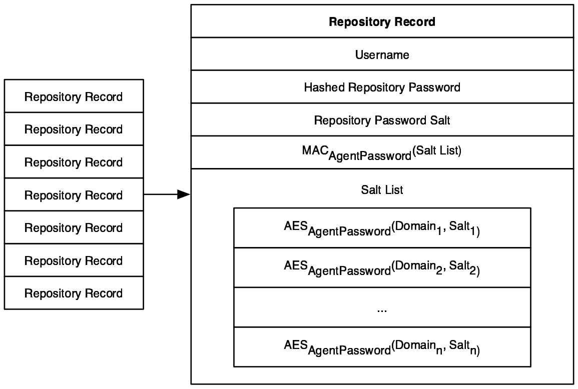


Figure 2: Data stored by the Salt Repository.

The Agent directly integrates into a web browser, allowing a user to generate passwords. The Agent is a variation of PwdHash for Firefox, but creating plug-ins for other major browsers should be a relatively simple task.

A PasswordAgent session begins like a normal browsing session with the user launching a web browser. The user must then log into the Agent with a username and Agent password. The Agent then transparently determines which Repository to use, and loads the user's salt list from it. The user continues to browse the web until a login form is encountered. Once the user enters a password field, the Agent indicates whether the current site is registered or not. If the user activates PasswordAgent, a unique site password will be generated by hashing the site's salt and the entered plain-text password. The login form is submitted with the site password, and the user logs into the site.

3.2 User Flow

Before using the PasswordAgent service, the user must register with a Repository and install the Agent. Registration consists of selecting a username and Agent password. Because the Agent is Repository agnostic, the username must provide enough information to determine which Repository to use. In consideration of this, all PasswordAgent usernames take the form of `username@domain` where `domain` is the domain name that the Repository belongs to. For example, a user who has the username `jsmith` and utilizes the institution of XYZ's PasswordAgent service would use the login `jsmith@xyz.edu`. The Agent can then locate the Repository at passwordagent.xyz.edu. This approach requires minimal memorization for a user, and allows for easy deployment and configuration of the Repository.

The Agent encrypts all the information that it stores in the Repository, so there is a requirement for both an encryption key and a password to authenticate to the Repository. To avoid having the user memorize two secrets, the Agent password is used as the encryption key and the Repository password is generated by hashing the full username (including the domain) and the Agent password together:

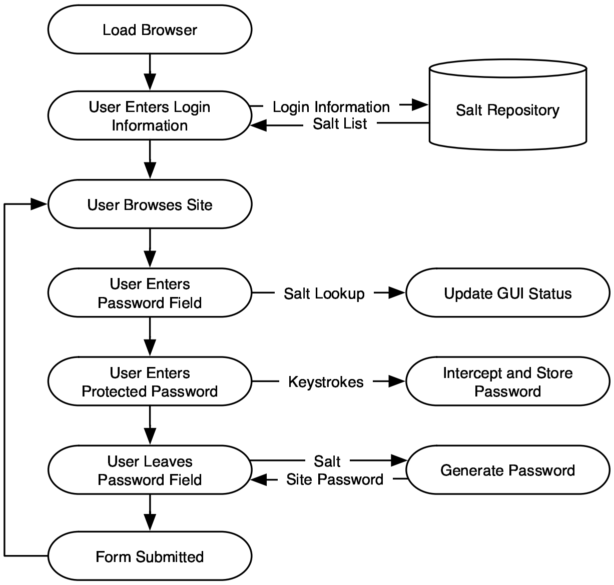


Figure 3: Basic user flow.

$$Password_{Repository} = SHA256(Password_{Agent}||Username) \quad (1)$$

$$EncryptedSalt = AES_{Password_{Agent}}(PlaintextSalt) \quad (2)$$

Since the Repository never knows the Agent password of a user, it cannot decrypt the stored information. In order to guarantee the integrity of the salt list, the Agent also stores a Message Authentication Code (MAC) [12, 13] calculated as:

$$MAC = HMAC - SHA256_{Password_{Agent}}(Domain_1||Salt_1 \\ ||Domain_2||Salt_2||...||Domain_n||Salt_n) \quad (3)$$

Registration can be accomplished either with automatic enrollment by a network administrator or via a web form provided by the Repository. Once registered for an account, a user would then need to install the browser plug-in. After installation, the user is ready to log in and begin a browsing session. An overview of a PasswordAgent session is given in Figure 3.

3.2.1 Login

When a web browser (e.g., Firefox) is first launched, the Agent lacks a salt list and is unable to protect any passwords. The user must authenticate to the Repository via

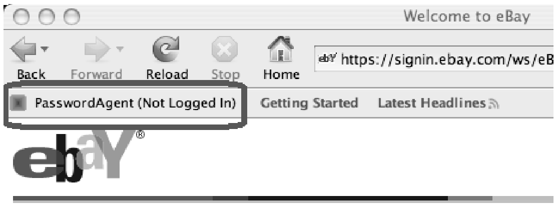


Figure 4: Toolbar displaying the status of PasswordAgent.

a login dialog. A toolbar button displayed by the Agent allows the user to enter the login dialog as shown in Figure 4.

Once the user enters a username and Agent password, the Agent determines the location of the Repository based on the domain portion of the username, generates the Repository password, and retrieves the salt list from the Repository. If successful, the salt list is decrypted using the user's Agent password and the MAC is verified. Should the MAC determine that the salt list has been tampered with, the user is warned via a dialog and the Agent remains logged out. Otherwise, the Agent updates the toolbar to reflect the new logged in status and retains the salt list in memory until the user logs out or exits the browser. To guard against fraudulent Repositories, all communication is performed over HTTPS. Because the Repository has to identify itself via an SSL certificate, the Agent is protected from being tricked into divulging the Repository password.

3.2.2 Browsing

If the user enters a password field during the process of web browsing, the Agent toolbar changes to inform the user whether or not the current site is registered. A site is considered registered if it has a salt associated with it, as shown in Figure 5(a), and unregistered if it does not, as shown in Figure 5(b). This allows the user to decide whether or not to enter a protected password.

3.2.3 Password Protection

The password input mechanism of the Agent is similar to that of PwdHash. In order to notify the Agent that a site password should be generated, the user enters a protected password. A protected password is created by either prefixing the plain-text password with @@ or by pressing a hotkey (F2). For example, a user who wishes to protect the plain-text password "secret" would type the protected password "@@secret" which would cause PasswordAgent to generate a site password. When a protected password is entered, PasswordAgent captures all keystrokes before they appear on the page, so JavaScript keyloggers cannot steal the plain-text password. Once the user leaves the password field, the protected password is analyzed to reveal the plain-text password (by removing the @@ prefix) and hashed together with the site's salt to create the site password. The site specific password is generated using the SHA256 hash function [3]:

$$Password_{Site} = SHA256(Password_{Plaint-text}||Salt_{Site}) \qquad (4)$$

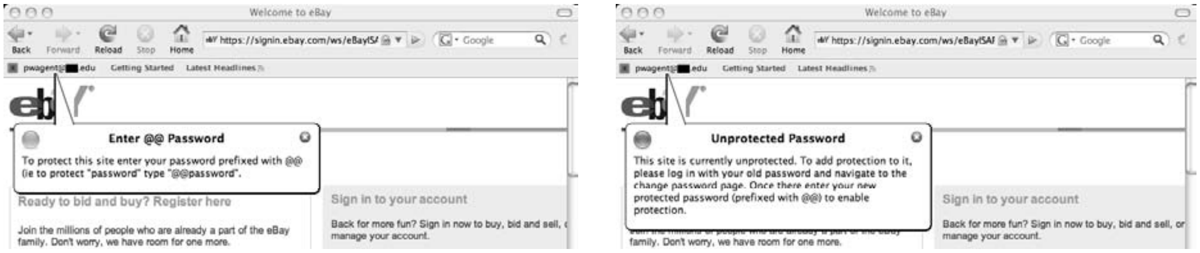
The main reason for using either the @@ prefix or the F2 hotkey is to let a user explicitly inform the hashing mechanism where to intercept the plain-text password. This guarantees that the data in other input fields will not be incorrectly hashed. The prefix @@ is chosen, because it is extremely unlikely that it will appear in a normal context. This enables PasswordAgent to scan the keystream and interpret @@ as an indicator to activate password protection. F2 is selected as a hotkey since it is currently not mapped to any functionality in Firefox [31].

3.2.4 Site Registration

To create a site password for a website, the site must first be registered in order to have a salt associated with it. If a user attempts to generate a site password via the prefix or hotkey mechanism on a site that has not yet been registered, an instructional dialog will appear. The dialog walks the user through registering the site with PasswordAgent. The dialog first confirms that the user wants to register the site with PasswordAgent, and hasn't accidentally triggered password protection. The user is then given a list of already registered sites as shown in Figure 6. The user is asked if the target site appears in this list. If it does, then the target site is actually a phishing attempt because it appears to be a registered site but in reality is from a different domain. The user is warned and prompted to navigate away. If the target site is not listed, the user is asked if he or she has an account with the target site or is creating a new account. Different instructions are displayed based on the user's response:

If the user has an existing account with the target site, that account must be migrated to use PasswordAgent. Migration is achieved by: (1) logging in with the plain-text password, (2) navigating to the change password page, and (3) entering the new protected password. A salt will then be generated, encrypted, and sent to the Salt Repository along with the updated MAC.

If the user is creating a new account on the unregistered site, then the user simply has to enter a protected password on the site registration form. The salt is generated, the new MAC for the salt list is calculated, and



Enter @@ Password

To protect this site enter your password prefixed with @@ (ie to protect "password" type "@@password".

Ready to bid and buy? Register here

Join the millions of people who are already a part of the eBay family. Don't worry, we have room for one more.

Sign in to your account

Back for more fun? Sign in now to buy, bid and sell, o manage your account.

(a)

Unprotected Password

This site is currently unprotected. To add protection to it, please log in with your old password and navigate to the change password page. Once there enter your new protected password (prefixed with @@) to enable protection.

family. Don't worry, we have room for one more.

Sign in to your account

Back for more fun? Sign in now to buy, bid and sell, or manage your account.

(b)

Figure 5: User focused password fields: (a) on a protected site, (b) on an unprotected site.

both are sent to the Repository for storage. The new salt is then used to generate the site password.

3.2.5 Multiple Accounts on One Site

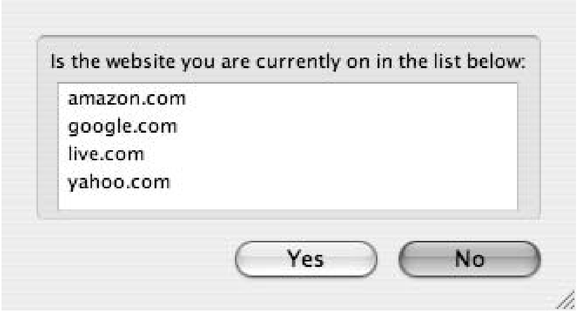
A user can have multiple accounts on a single site, for example, someone may have two Gmail accounts. PasswordAgent is compatible with this scenario, as it can use the site's salt to hash both passwords. In this case, password uniqueness cannot be guaranteed because if the user selects the same password for both accounts, the protected password will also be the same. This is a minor issue, given that it is a relatively rare scenario. This issue also exists in PwdHash and Password Multiplier. Password protection is provided in that a compromised password on the site with multiple accounts will only effect that particular site - all other sites are guaranteed to have unique passwords.

3.2.6 Changing Site Password

The site password can be changed by one of two mechanisms. The first is to change the plain-text password as one would do with a normal password (i.e., "@@password" to "@@newpassword"). The Salt Repository does not need to be notified in this case, since the salt remains unchanged. The new password is protected in the same manner as the old password. This has the advantage of not requiring the user to learn any new paradigms about changing passwords. The second is to keep the plain-text password intact but to change the site salt.

3.2.7 Password Format

Every site has different requirements for passwords. Some sites require at least one non-alphanumeric character, while others prohibit them entirely. To allow for these different formats, the user's plain-text password is examined for clues as to the makeup of a valid site password. If the user does not include a non-alphanumeric character in the plain-text password, the site password would not contain one and the site would notify the user of the incorrect composition of the password. Any changes in the plain-text password will be reflected in the site password, enabling PasswordAgent to generate valid passwords for all sites without any specific prior knowledge. Such a design was first presented in [31]. While this technique does leak information about the plain-text password, it is of little concern because no information about the salt is revealed. This technique avoids the need to constantly update a list of composition rules for common sites on the Internet. This also addresses an important usability issue of users being dissatisfied with site passwords. Users become concerned when sites, like Hotmail, offer a password strength meter and the site passwords are rated as medium instead of strong [14]. By inspecting plain-text passwords for clues, the indicated strength of a password is directly related to the strength of the plain-text password. It should be noted that the actual strength of the site password is greater, even if the password meter indicates they are the same. A user provided character string has less entropy than a salted and hashed version of that string.



Is the website you are currently on in the list below:

amazon.com
google.com
live.com
yahoo.com

Figure 6: Information dialog that assists users in recognizing phishing sites.

3.2.8 Roaming

Roaming can be achieved in one of two ways. A roaming user can either install the Agent as outlined before, or site passwords can be generated via a web interface provided by the Repository. The web interface allows the user to log in and generate passwords for a specific site, which can be copied and pasted into the login form. This enables users without the ability to install the Agent to still access their accounts. PwdHash implements a similar web based mechanism for roaming users, however it is located at a URL that is complex and difficult to memorize [14]. Because the Repository is located at *passwordagent.domain*, it is simple to provide the web interface at that address. Doing this reduces the memory burden as users already know the domain of their Repository as part of their logins, all they have to do is remember to prepend *passwordagent* to it.

4 Security and Usability Analysis

4.1 Security Analysis

The primary goal of PasswordAgent, like other password hashing schemes, is to improve user security. Here we compare the security of PasswordAgent with those of LPWA, PwdHash, Password Multiplier, and Passpet in ten different aspects. The comparison results are summarized in Table 2. A detailed discussion is as follows, outlining the major security concerns with the existing password hashing mechanisms.

Unique Passwords: Each password hashing solution generates a unique password for each site, even if the plain-text password is the same.

Offline Attacks: PasswordAgent is less vulnerable to offline attacks. Because the salt list is not stored locally, launching an offline attack to retrieve the salt list is difficult. Moreover, the Salt Repository can defend against online attacks by limiting the number of login attempts allowed per minute. Password Multiplier and Passpet are also resistant to offline attacks as long as the local machine remains uncompromised. However, if an attacker breaches the computer and retrieves the cached master password, a relatively inexpensive offline attack can be launched to expose every site password. With PasswordAgent, even if the Agent password is stolen, only the salt list is revealed. The attacker would still need to launch an online attack against the target site to determine the site password.

Compromised Plain-text Password: In the scenario where the plain-text password is compromised, only PasswordAgent still provides user protection. An attacker would be unable to use the compromised password, because the random site salt is not known. PwdHash does not have this advantage, as the salt is the site's domain name, allowing an attacker to utilize the compromised password to access the site. Even worse, Password Multiplier and Passpet both use one plain-text password as a master password to generate all of the site passwords. Should the master password be compromised, every password protected by Password Multiplier and Passpet will also be compromised.

Compromised Site Password: All password hashing schemes claim to protect users when a site password is compromised. However, because PwdHash uses MD5 and a known salt, the domain name, it is possible to launch a brute force attack on the compromised password. A phisher impersonating a single site could launch a time-space trade-off attack and feasibly retrieve the plain-text password. In contrast, PasswordAgent defends against offline attacks with a large random salt. Assuming that it takes 1ms to calculate a hash with a 256-bit salt, it would take roughly 10^{66} years on average to find the plain-text password. Furthermore, even if attackers are able to recover the plain-text password, they still have to launch an online brute force attack in order to discover the salt for any other site that uses the same plain-text password.

Basic Phishing Protection: The nature of hash-based password generation allows all schemes to provide a basic level of phishing protection. Because each site password is unique, using any of these password generation tools on a phishing site will not immediately expose the login of the target site. As previously noted though, the site password can be used in offline attacks to reveal the plain-text password. LPWA, PwdHash, Password Multiplier, and Passpet all suffer from this problem. However, PasswordAgent offers the additional security with random salts, so even a stolen plain-text password will not give an attacker access to a login.

Advanced Phishing Protection: PasswordAgent provides early warning against phishing sites. If a user attempts to enter a protected password on an unregistered site, an information dialog notifies the user. This dialog, as shown in Figure 6, warns the user that the current site is not registered and displays a list of registered sites. This allows users to check if they are on a phishing site. Displaying security information in the browser chrome, PasswordAgent prevents its user interface from being spoofed by web pages. Because web pages do not have access to the browser chrome, it is difficult to place a fake login button or security indicator.

Shoulder Surfing Protection: PasswordAgent makes shoulder surfing—watching a user type in a password—much more difficult to succeed, because it requires the

	Security	LPWA	PwdHash	Password Multiplier	Passpet	**PasswordAgent**
1	Unique Password for Each Site	yes	yes	yes	yes	yes
2	Resist Offline Attacks	-	-	no	no	yes
3	Protect Compromised Plain-text Password	no	no	no	no	yes
4	Protect Compromised Site Password	yes	yes	yes	yes	yes
5	Basic Phishing Protection	yes	yes	yes	yes	yes
6	Advanced Phishing Protection	no	no	no	yes	yes
7	Enhance Shoulder Surfing Protection	no	no	no	no	yes
8	Secured Remote Storage	-	-	-	no	yes
9	Adaptation to Faster Computers	no	no	yes	yes	yes
10	Provide Data Anonymity	yes	no	no	no	no
	Usability					
1	Allow Easy Site Password Update	yes	yes	no	yes	yes
2	Notify if Site is Protected	no	no	no	yes	yes
3	Support all Site Specific Password Requirements	no	yes	no	no	yes
4	Minimal Change to Browsing Paradigm	yes	yes	no	no	yes
5	Requires 3rd Party Server	no	no	no	yes	yes

Table 2: Comparison of PasswordAgent with four other tools.

observation of two separate events, the typing of the Agent password and the typing of the site password. Since the Agent password is entered only when the user begins a session, an attacker is forced to hover around the victim for longer periods of time, increasing the chance of detection. Other schemes, however, only require one password, making the attacker easier to succeed.

Secured Remote Storage: The Salt Repository of PasswordAgent is cryptographically secure, and does not leak any useful information to attackers. By contrast, Passpet leaks not only whether a username exists (through the $list$ command) but also how large k_1 is [38], where k_1 is the number of iterations of a hash function used for generating the site password. The smaller the k_1, the weaker the password. Armed with this knowledge, an attacker can target a user with a small k_1 value and launch a brute force attack on the weakest master password. Both PasswordAgent and Passpet store only encrypted data and guarantee the integrity of the data with a MAC. Even in a situation in which a Salt Repository is compromised, the leaked information would not be useful because the attacker would have to brute force the salt list and then launch an online attack against the site specific passwords. It is technically possible to launch a brute force against the salt list, however it would take a prohibitively long time. This in combination with the required online attack against individual sites mitigates the possibility of a malicious Salt Repository compromising the security of PasswordAgent.

Adaptation to Faster Computers: PasswordAgent can adapt to faster computers and the associated greater power of attackers in launching dictionary/brute force attacks, by increasing the salt size. This is a minor change to the Agent implementation. The user simply regenerates a longer salt while keeping the plain-text password intact. The newly-generated site password is stronger, and no extra memory burden is placed on the user. In contrast, it is not easy for PwdHash to adapt to adversaries with more computing power. Both Passpet and Password Multiplier can increase the number of iterations to make it harder for an attacker to compute the site password.

Data Anonymity: Only LPWA has data anonymity as its goal. The other solutions, including PasswordAgent, focus solely on password protection. LPWA enables a user to browse, hold accounts, and email without ever revealing personal identification information.

4.2 Additional Usability Benefits

Usability is a key factor in any software system. A simple usability flaw might render a cryptographically secure system useless. Care is taken in the development of PasswordAgent to address usability concerns that exist in previous password hashing solutions. The specific usability benefits of PasswordAgent are detailed as follows.

Ease of Site Password Updating: PasswordAgent allows users to change their site passwords exactly like they normally do, via the change password page of the website. By changing it to a new protected password, users maintain all the benefits of PasswordAgent without any complicated or confusing processes. PwdHash has the same functionality. In contrast, Password Multiplier forces users to append information to the domain name being hashed. Not only is this confusing, but it also forces users to remember what additional information they are using for their logins [14]. Passpet uses a similar mechanism, in which users can change the label of a site to change the password. Unlike Password Multiplier though, it remembers the change and does not require additional memorization.

Notification of Protected Sites: Only PasswordAgent and Passpet notify users when a site requires protected passwords. PasswordAgent displays a “notification bubble”, which informs the user of the status of the site and how to login, as shown in Figures 5(a) and 5(b). In addition to notification bubble, PasswordAgent allows the user to view a list of all the registered sites. Both PwdHash and Password Multiplier fail to indicate whether a site is expecting a protected or plain-text password. Users who enter an incorrect password will often proceed to enter many of their other passwords, including plain-text passwords [14]. This leads to multiple passwords being exposed, a situation that is even worse than if no password protection is used.

Changing Master Password: The user can change the master password for the Salt Repository at any point without changing the password on any individual site. By entering the old and new Agent Password, the salt list can be decrypted and then re-encrypted with the new password. Because the same salts are used to generate the password, the site password remains the same. This is more convenient than in Passpet and Password Multiplier, where a change to the master password requires the user to login into each individual site and manually change the password.

Site Specific Password Requirements: Many sites have different password requirements, including size and acceptable characters. Only PasswordAgent and PwdHash examine the user’s plain-text password for clues to the expected composition of a password. Any errors with the plain-text password are mirrored in the site password, so the user receives useful feedback.

Minimal Changes to Browsing Paradigm: Similar to PwdHash, PasswordAgent makes only minimal changes to the normal interaction between a user and a web browser. The only two changes include: (1) the user must log into the Agent when beginning a session, and (2) the protected password must start with @@ (or the user must activate PasswordAgent via the F2 key). These minimal changes should make the adoption of PasswordAgent easy. Password Multiplier and Passpet both require obvious deviations from the normal user login.

Ease of Switching Storage Servers: PasswordAgent is completely repository agnostic, and can easily transfer the salt list from one repository to another. In contrast, Passpet uses the storage server address as part of the master password generation, thus any change in the storage server address forces users to create a new master password and update all their site passwords.

```
<?xml version=’1.0’ encoding=’utf-8’?>
<response>
  <status>statusInfo</status>
  <message>messageBody</message>
  ......
  <data>dataSection</data>
</response>
```

Figure 7: XML Response Format.

5 Implementation and Evaluation

5.1 Implementation

We build a prototype of PasswordAgent, in which the Salt Repository is implemented as a Java servlet and the Agent is implemented as a Firefox extension.

5.1.1 Salt Repository Interface

The Salt Repository Interface is a simple XML-based REST-style protocol, which allows the creation of user accounts, the updating of site salts, and the retrieval of the salt list. These methods fulfill the minimum requirements to maintain a salt list. The Interface is designed for ease of use with JavaScript’s XMLHTTPRequest object. Because the XMLHTTPRequest object allows synchronous HTTPS requests and can translate an XML response into a DOM document, it takes minimal additional code for the Agent to communicate with the Repository. The Salt Repository is written as a Java servlet, which eases its deployment across different platforms. Any web server supporting HTTPS can serve as a Repository, as long as it implements the Salt Repository Interface and is located at *passwordagent.domain*.

Here the Salt Repository is maintained by a publicly accessible HTTPS server that implements the REST [20] methods as listed in Table 3. These methods allow users to maintain their salt lists.

The Interface is designed to be as simple as possible for implementation, and uses a simple XML response format that is easy to parse. The response format is illustrated in Figure 7. Each response contains at least a <status> element that is either “success” or “error” and a <message> element that includes a natural language description of the response. Some methods return

Method	Description	Parameters	Data Section Format
CreateUser	Creates a new user account.	user - the desired username. password - the desired repository password. hmac - the HMAC code to store for the current (empty) salt list	N/A
GetSites	Retrieves the salt list for a user.	user - the username of the user. password - the repository password of the user	<data hmac=“SaltListHMAC”> <site domain=“domain.com” salt=“salt” /> <site domain=“domain2.com” salt=“salt2” /> </data>
SetSite	Stores a salt for a specified domain.	user - the username. password - the repository password. site - the domain salt - the new salt hmac - the HMAC code to store for the current (including this updated entry) salt list	N/A
UpdateUser	Updates a user’s repository password.	user - the username. password - the repository password. newpassword - the desired new authentication password.	N/A

Table 3: Salt Repository methods.

a <data> section that includes more information, allowing further data to be passed to the caller.

5.1.2 Agent

The Agent is a Firefox extension written in JavaScript and XML User Interface Language (XUL) [10], without using native components. It is a modified version of the open source PwdHash. While the basic password protection activation code remains the same, additional functionality is provided in the form of a GUI, a more secure hash function, and a module to communicate with the Salt Repository. PwdHash has no visible GUI, PasswordAgent, by contrast, includes status indicators and warning dialogs to assist users in protecting their passwords. PwdHash uses the MD5 hash function, but recent collision attacks have rendered MD5 insecure [35]. PasswordAgent uses SHA256 for all hashing functions and AES [4, 15] for salt encryption. Although PasswordAgent uses a more complex hash algorithm and hashes larger values, it is still reasonably efficient as it takes only about 45ms to generate a password using SHA256, benchmarked on a 2.26Ghz Intel machine running SuSE Linux 10.2 with 512MB RAM.

5.2 Evaluation

We focus our evaluation on the usability of PasswordAgent, which is a key measure determining whether a password manager is really useful or even secure [14]. We choose user studies, i.e., laboratory user tests [32, 36], to assess the usability of PasswordAgent. We select PwdHash for a direct comparison with PasswordAgent. This is because both use the same activation method, and PwdHash scores higher than Password Multiplier on perceived security and usability [14]. In the design our usability tests, we follow a similar approach to the usability study on PwdHash and Password Multiplier [14].

Question	People Responding “Yes”
Do you sometimes reuse passwords on different sites?	96.4% (27)
Are you concerned about the security of passwords?	28.6% (8)
Criteria for choosing passwords:	
Easy to remember	75.0% (21)
Difficult for others to guess	42.9% (12)
Suggested by the system	0% (0)
Same as another password	57.1% (16)
Other	10.7% (3)
Participation in online activities requiring personal or financial details:	
Online purchases	75% (21)
Online banking	75% (21)
Online bill payments	28.6% (8)
Other activities	42.9% (12)
Do you use:	
A password manager?	3.6% (1)
A password generation tool?	0% (0)

Table 4: Participants’ initial attitude towards password security.

5.2.1 Participants

There are 28 individuals ranging in age from 17 to 63 years old participated in the user study. Only one of the participants is a computer science major. None of the participants has any particular experience with computer security. A pre-task questionnaire, similar to the one in [14], is used to survey participants’ initial attitude towards password security. The questions and responses are summarized in Table 4. We can see that only 42.9% of participants choose “difficult for others to guess” passwords, only 4% of participants do not reuse passwords across different websites, and only one participant has ever used software to manage passwords. A useful password generation tool would resolve the security issues caused by these poor password practices.

5.2.2 Tasks

Each participant is asked to complete a set of tasks using two password generation plugins: PasswordAgent and PwdHash. The tasks are completed on two personal computers, designated A and B. Both computers run SUSE Linux and Mozilla Firefox. Computer A serves as the participants' primary computer, while Computer B is used to let participants install and use plugins from a computer other than their primary machine. Five tasks are carefully selected to reflect the realistic daily usage of a password generation plugin:

- **Migrate Login** : From computer A, logging on to a website W (Yahoo) with an account that has not yet been protected, migrating the account, and getting the password protected by the plugin. This task simulates taking an existing account and protecting it with the plugin.
- **Log Into Site** : From computer A, logging on to a website W (Google) with an account that has already been protected by the plugin. This task simulates a user's regular login process using protected accounts.
- **Update Password** : From computer A, logging on to the website W (Hotmail) with a protected account, and changing its password. This task simulates the process of changing the password of a protected account.
- **Second Login** : From computer A, logging on to a website W (Hotmail) with the protected account whose password has just been updated in "Update Password". This task simulates the process of logging in with updated passwords.
- **Remote Login** : From computer B, logging on to the website W (Amazon) with a protected account. This task simulates when users log in from remote machines that do not have the plugin installed.

Each task is performed with PwdHash and PasswordAgent. Participants are given a simple instruction sheet, which instructs them on how to use PwdHash and Password Agent. They are allowed to refer to the instructions whenever necessary. Accounts are created for the purpose of the usability tests, instead of having the participants use their personal accounts.

5.2.3 Results

Results are collected through both observation and questionnaires. An experimenter observes the test session of each participant and records the results. The experimenter does not provide additional instructions to a participant during the test session. The observed outcome of each task is classified into one of five groups: *successful*, i.e., the participant completes the task without a problem; *dangerous success*, i.e., the participant completes the task after an attempt that may lead to a security exposure;*failed*, i.e., the participant cannot complete the task and gives up; *false completion*, i.e., the participant erroneously thinks that the task has been correctly completed, when it has not; and *failed due to previous*, i.e., the participant does not complete this task due to the failure of previous task(s). Table 5 lists the task completion results for PasswordAgent and PwdHash. We can see that PasswordAgent achieves an over 90% success ratio for four tasks, and meanwhile it outperforms PwdHash in all the five tasks.

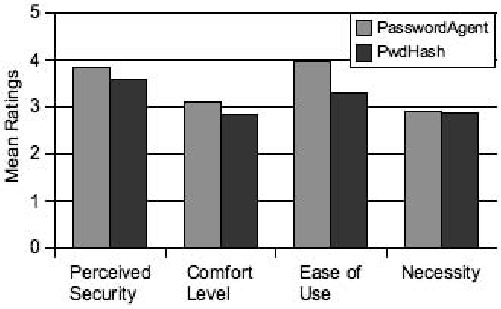


Figure 8: Mean questionnaire responses for each question group on scale of 1 to 5 (1 very negative, 3 neutral, 5 very positive).

After completing the tasks for a plugin, each user answers a questionnaire for that plugin. The questionnaire consists of eight Likert scale statements [28]. The participants are asked to indicate their degree of agreement with each statement after they finish the tasks. We use a five-point Likert scale: *strongly disagree*, *disagree*, *neutral*, *agree*, and *strongly agree*. Table 6 lists the questionnaire statements, which are very similar to the ones in [14]. A summary of the results are shown in Figure 8. The questionnaire focuses on four different categories: Perceived Security, Perceived Comfort, Perceived Ease of Use, and Perceived Necessity and Acceptance. While PasswordAgent scores higher than PwdHash in all four measurements, we further use t-test to determine the statistical significance of the differences in scores and observe that these differences do not have statistical significance.

6 Limitations

In this section, we discuss three limitations in PasswordAgent: vulnerability to keyloggers [21], the reliance on Salt Repository, and the usability limitations. PasswordAgent is designed to protect against web based attacks and cannot thwart compromises outside of the browser. Should a system have malicious software in-

Task	PasswordAgent					PwdHash				
	Success	Dangerous Success	Failures			Success	Dangerous Success	Failures		
			Failure	False Completion	Failed Previous			Failure	False Completion	Failed Previous
Migrate Login	92.9% (26)	0% (0)	7.1% (2)	0% (0)	0% (0)	75% (21)	14.3% (4)	10.7% (3)	0% (0)	0% (0)
Log Into Site	96.4% (27)	0% (0)	3.6% (1)	0% (0)	0% (0)	89.3% (25)	10.7% (3)	0% (0)	0% (0)	0% (0)
Update Password	96.4% (27)	0% (0)	3.6% (1)	0% (0)	0% (0)	67.8% (19)	14.3% (4)	17.9% (5)	0% (0)	0% (0)
Second Login	96.4% (27)	0% (0)	0% (0)	0% (0)	3.6% (1)	75% (21)	7.1% (2)	0% (0)	0% (0)	17.9% (5)
Remote Login	82.1% (23)	0% (0)	17.9% (5)	0% (0)	0% (0)	46.4% (13)	28.6% (8)	25% (7)	0% (0)	0% (0)

Table 5: Task completion results for PasswordAgent and PwdHash.

Perceived Security
My passwords are secure when using PasswordAgent.
I do not trust PasswordAgent to protect my passwords from cyber criminals.
Comfort Level with Giving Control of Passwords to a Program
I am uncomfortable with not knowing my actual passwords for a website.
Passwords are safer when users do not know their actual passwords.
Perceived Ease of Use
PasswordAgent is difficult to use.
I could easily log on to web sites and manage my passwords with PasswordAgent.
Perceived Necessity and Acceptance
I need to use PasswordAgent on my computer to protect my passwords.
My passwords are safe even without PasswordAgent.

Table 6: Post-task Questionnaire (for PasswordAgent, the questionnaire for PwdHash was identical other than the name of the software).

stalled such as spyware or a keylogger, both the Agent password and the individual site passwords can be compromised.

The Salt Repository is an important part of the PasswordAgent solution. Should it become unavailable (because of server issues, network problems, or DOS attacks), the user would be unable to log into any protected site. However, it is possible to use the Salt Repository as a backup, if the user's primary computer stores the salt list and then mirrors any changes to the Repository. This can achieve high reliability, but would come at a security cost. If the primary computer is compromised, the salt list has a higher chance of being exposed than before. A potential area for improvement would be the support of multiple synchronized repositories to prevent a single point of failure. Building such a mechanism is beyond the scope of this paper.

A user must activate the password protection by using @@ (the F2 key, or some other means). This is the main usability limitation that is common to PwdHash, Password Multiplier, and PasswordAgent. This extra activation step may make some users feel inconvenienced. Moreover, if a user forgets to invoke the protection, this limitation may lead to security exposures because the user's plain-text password might be sent to a phishing site [14]. Although the inconvenience still exists, the security risks caused by this limitation is eliminated in PasswordAgent. A phisher cannot obtain the correct site password since the salt is not accessible to the phisher.

Another usability limitation is that if a user forgets the Agent password, then there is no mechanism to retrieve the users salts. The user has to manually reset their passwords on each site, using a forgotten password feature. While inconvenient, most websites today provide a mechanism to reset forgotten passwords so serious harm is avoided.

7 Conclusion

We have developed PasswordAgent, an automatic password management system with enhanced hashing. PasswordAgent includes a Salt Repository and a browser plug-in Agent, and it provides a convenient and secure password protection service in an automatic manner. Without altering the normal interaction between a user and a login form, PasswordAgent automatically secures the user's plain-text password by rendering a unique site password for each website visited. Under the stronger security guarantee, a user's site password is robustly defended against password cracking and theft. We have implemented a prototype of PasswordAgent and conducted usability experiments. The evaluation results clearly indicate the usability benefits of PasswordAgent.

Acknowledgment

We would like to thank the anonymous reviewers and our shepherd Travis Campbell for their insightful comments. This work was partially supported by NSF grants CNS-0627339 and CNS-0627340.

References

[1] Anti-Phishing Working Group. http://www.antiphishing.org/index.html.

[2] Consumer sentinel network data book. Federal Trade Commission, February 2009.

[3] FIPS Publication 180-2. NIST, August 2002.

[4] FIPS Publication 197. NIST, November 2001.

[5] KeePass Password Safe. http://keepass.info/.

[6] Password strength. http://www.passwordmeter.com.

[7] Petname Tool. http://petname.mozdev.org/.

[8] RoboForm: Password Manager, Form Filler, Password Generator, Fill&Save Forms. http://www.roboform.com/.

[9] Windows Live ID. https://accountservices.passport.net/.

[10] XML User Interface Language (XUL) Project. http://www.mozilla.org/projects/xul/.

[11] ADAMS, A., AND SASSE, M. A. Users are not the enemy. *Commun. ACM 42*, 12 (1999), 40–46.

[12] BELLARE, M., CANETTI, R., AND KRAWCZYK, H. Keying hash functions for message authentication. In *Proceedings of Crypto'96* (1996), pp. 1–15.

[13] BELLARE, M., KILIAN, J., AND ROGAWAY, P. The security of the cipher block chaining message authentication code. In *Proceedings of Crypto'94* (1994), pp. 341–358.

[14] CHIASSON, S., VAN OORSCHOT, P., AND BIDDLE, R. A usability study and critique of two password managers. In *Proceedings of the 15th USENIX Security Symposium* (2006), pp. 1–16.

[15] DAEMEN, J., AND RIJMEN, V. The design of rijndael: Aes - the advanced encryption standard. *Springer-Verlag* (2002).

[16] DAVIS, D., MONROSE, F., AND REITER, M. K. On user choice in graphical password schemes. In *Proceedings of the 13th USENIX Security Symposium* (2004), pp. 151–164.

[17] DHAMIJA, R., AND PERRIG, A. Dejà vu: A user study using images for authentication. In *Proceedings of the 9th USENIX Security Symposium* (2000), pp. 45–58.

[18] DHAMIJA, R., TYGAR, J. D., AND HEARST, M. Why phishing works. In *Proceedings of the SIGCHI conference on Human Factors in computing systems* (2006), pp. 581–590.

[19] FELDMEIER, D. C., AND KARN, P. R. Unix password security - ten years later. In *Proceedings of Crypto'89* (1989), pp. 44–63.

[20] FIELDING, R. T., AND TAYLOR, R. N. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT) 2*, 2 (2002), 115–150.

[21] FLORENCIO, D., AND HERLEY, C. Klassp: Entering passwords on a spyware infected machine using a shared-secret proxy. In *Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC'06)* (2006), pp. 67–76.

[22] FLORÊNCIO, D. A. F., AND HERLEY, C. A large-scale study of web password habits. In *Proceedings of the 16th International Conference on World Wide Web* (2007), pp. 657–666.

[23] GABBER, E., GIBBONS, P. B., KRISTOL, D. M., MATIAS, Y., AND MAYER, A. Consistent, yet anonymous, Web access with LPWA. *Commun. ACM 42*, 2 (1999), 42–47.

[24] HALDERMAN, J. A., WATERS, B., AND FELTEN, E. W. A convenient method for securely managing passwords. In *Proceedings of the 14th international conference on World Wide Web* (2005), pp. 471–479.

[25] HERLEY, C., VAN OORSCHOT, P., AND PATRICK, A. S. Passwords: If we're so smart, why are we still using them? In *Proceedings of the Financial Cryptography and Data Security Conference* (2009).

[26] JERMYN, I., MAYER, A., MONROSE, F., REITER, M. K., AND RUBIN, A. D. The design and analysis of graphical passwords. In *Proceedings of the 8th USENIX Security Symposium* (1999), pp. 1–14.

[27] KORMANN, D. P., AND RUBIN, A. D. Risks of the passport single signon protocol. *Comput. Networks 33*, 1-6 (2000), 51–58.

[28] LIKERT, R. A technique for the measurement of attitudes. *Archives of Psychology 140* (1932), 1–55.

[29] MORRIS, R., AND THOMPSON, K. Password security: a case history. *Commun. ACM 22*, 11 (1979), 594–597.

[30] RIVEST, R. L. The md5 message-digest algorithm. In *RFC 1320* (April 1992).

[31] ROSS, B., JACKSON, C., MIYAKE, N., BONEH, D., AND MITCHELL, J. C. Stronger password authentication using browser extensions. In *Proceedings of the 14th USENIX Security Symposium* (2005), pp. 17–32.

[32] RUBIN, J., AND CHISNELL, D. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. John Wiley & Sons, Inc., 1994.

[33] TARI, F., OZOK, A. A., AND HOLDEN, S. H. A comparison of perceived and real shoulder-surfing risks between alphanumeric and graphical passwords. In *Proceedings of the second symposium on Usable privacy and security (SOUPS '06)* (2006), pp. 56–66.

[34] THORPE, J., AND VAN OORSCHOT, P. Human-seeded attacks and exploiting hot-spots in graphical passwords. In *Proceedings of the 16th USENIX Security Symposium* (2007), pp. 103–118.

[35] WANG, X., AND YU, H. How to break md5 and other hash functions. In *Proceedings of EUROCRYPT 2005* (2005), pp. 19–35.

[36] WHITTEN, A., AND TYGAR, J. D. Why johnny can't encrypt: a usability evaluation of pgp 5.0. In *Proceedings of the 8th USENIX Security Symposium* (1999), pp. 169–184.

[37] YAN, J., BLACKWELL, A., ANDERSON, R., AND GRANT, A. Password memorability and security: Empirical results. *IEEE Security and Privacy 2*, 5 (2004), 25–31.

[38] YEE, K.-P., AND SITAKER, K. Passpet: convenient password management and phishing protection. In *Proceedings of the second symposium on Usable privacy and security (SOUPS'06)* (2006), pp. 32–43.

[39] YUE, C., AND WANG, H. Anti-phishing in offense and defense. In *Proceedings of the 24th Annual Computer Security Applications Conference (ACSAC'08)* (2008), pp. 345–354.