# Passpet: Convenient Password Management and Phishing Protection

Ka-Ping Yee
University of California, Berkeley
ping@zesty.ca

Kragen Sitaker
kragen@pobox.com

## ABSTRACT

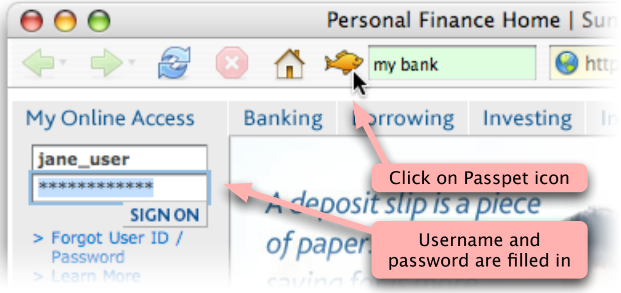
We describe Passpet, a tool that improves both the convenience and security of website logins through a combination of techniques. Password hashing helps users manage multiple accounts by turning a single memorized password into a different password for each account. User-assigned site labels (petnames) help users securely identify sites in the face of determined attempts at impersonation (phishing). Password-strengthening measures defend against dictionary attacks. Customizing the user interface defends against user-interface spoofing attacks. We discuss how these techniques are integrated into a single tool and compare Passpet to other solutions for managing passwords and preventing phishing.

## Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—Security and protection; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—Web-based services; H.4.3 [**Information Systems Applications**]: Communications Applications—Information browsers; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—Graphical user interfaces

## 1. INTRODUCTION

Passwords are the most commonly used type of authentication on the Web, but they have many usability problems and security weaknesses. Password security depends on choosing passwords that are unique and hard to guess, yet long passwords can be difficult to remember and retype correctly. The passwords that are easiest to choose and memorize tend to be vulnerable to *dictionary attacks*, in which an attacker tries to guess the password by constructing likely possibilities from lists of words and common passwords. Changing passwords frequently helps to resist attack, but makes the task of memorizing passwords even harder. Using the same password or related passwords at multiple sites compromises password secrecy, yet memorizing a different password for every site imposes an unrealistic burden on human users. Password login forms are also vulnerable to *phishing attacks*, in which the user is fooled into entering a password at an imitation site. Some of the more sophisticated phishing attacks also corrupt or mimic parts of the browser's user interface to mislead the user about a site's true identity.


Personal Finance Home | Su
my bank
http
My Online Access
Banking
orrowing
Investing
jane_user
************
SIGN ON
> Forgot User ID / Password
> Learn More
A deposit slip is a piece of paper saving f

Click on Passpet icon

Username and password are filled in

**Figure 1: One click on the Passpet button fills in a login form with a site-specific password.**

Passpet improves upon and combines several previously devised techniques — password hashing, petnames, password strengthening, and UI customization — to mitigate all of the problems just mentioned, helping users manage website logins both more conveniently and more securely. Whereas most other solutions improve password security at the cost of some extra effort to log in, Passpet improves password security while making the process of logging in more convenient. With Passpet, the user clicks a button to fill in a password, in login forms (Figure 1) and in other contexts.

As with any system that tries to address many competing requirements, Passpet incorporates some tradeoffs in its design. Passpet makes logging in more convenient at the cost of a more involved process to change the user's secret; it reduces the user's memory burden at the cost of introducing an external dependency; it supports non-SSL sites at the cost of some risk if non-SSL site passwords are intercepted. We have made these tradeoffs in a way that we believe improves security and usability for most users, keeping in mind that phishing attacks and dictionary attacks tend to be high-volume attacks.

In addition to describing Passpet as a specific point in the design space, this paper also contributes some ideas in secure interaction design that are separable and could be

*SOUPS* 2006, Carnegie Mellon University, Pittsburgh, PA

reapplied in other contexts:

- Using user-assigned site labels for password hashing.
- Continuously estimating the dictionary-attack time while the user enters a new password.
- Variable levels of password strengthening, configured by the user simply by waiting.
- Associating a security tool with a persona that differs from user to user.
- Customizing the button for activating a security tool in order to prevent spoofing.

Below, we will describe in more detail how these techniques are applied in Passpet, how Passpet generates passwords, how the usage of Passpet fits in with current practices, and how Passpet's security and usability compare to existing alternatives.

## 2. PROBLEM AND GOALS

In the following discussion of the problems surrounding passwords, we identify several goals for a password tool.

In the current password regime, the security goal of keeping passwords different from site to site competes with the limitations of human memory. Users tend to reuse passwords because memorizing many passwords is difficult and choosing many passwords is inconvenient. We would like the user to only have to memorize one secret instead of many, but still have a unique password at each site to reduce the user's vulnerability in the case of a break-in at one account or a malicious administrator at a site hosting an account.

Because any human-chosen secret is likely to have much less entropy than a truly random secret, we would like to provide a defense against dictionary attacks on such secrets. Since offline dictionary attacks depend on computational speed, we would also like this defense to remain effective in future years as faster computers become available.

The reason phishing works is that users are constantly asked to enter passwords without a convenient, reliable way to know whom they're giving their passwords to. The user interface they use to enter secrets is completely controlled by an unidentified party. Today's login forms *train* users to be excellent targets for phishing attacks. Any effective phishing solution must provide a way to identify the other party and must get users out of the habit of giving secrets to strangers. Some attacks try to fool users by imitating login forms, and some attacks imitate parts of the browser's user interface; we would like to address both of these kinds of attacks.

For robustness and flexibility, we would like our password scheme to avoid centralized external dependencies. We also prefer not to store the user's passwords, in order to reduce the user's exposure to risk in the event of theft or break-in of a system where passwords are stored.

We must also consider several compatibility and deployability constraints for a password tool. We assume that it is impractical to require websites to change their login functionality to accommodate a new tool, and further that it is impractical to expect users to migrate all their accounts to a new scheme at once. Currently, users can change their passwords at individual sites; we want users to still have this ability and to be able to use sites that require periodic password changes. Also, users can currently log in to websites from anywhere — such as from a computer at the office, from a friend's computer, or in an emergency while travelling. Retaining this ability is important.

Finally, we note that any password management tool will require some changes in user behaviour, however slight. Voluntary adoption of a tool will be much less likely if it offers no convenience benefit (or, worse, if it imposes additional inconvenience compared to current practice); therefore, we seek a design that improves convenience at least for the most common user task, which is logging in.

In summary, we aim to satisfy the following:

*Usability Goals*

1. Improve the convenience of logging in to websites.
2. Work with existing websites and login forms.
3. Allow site-by-site migration to using the tool.
4. Allow the user to change passwords for individual sites.
5. Allow the user to log in from more than one computer.
6. Let the user only have to memorize one secret.
7. Allow the user to change the master secret.

*Security Goals*

8. Use a unique password for each site.
9. Resist offline dictionary attacks on user-chosen secrets.
10. Adapt to the development of faster computers.
11. Avoid storing passwords in long-term storage.
12. Avoid introducing a centralized dependency.
13. Resist attacks based on fake website login forms.
14. Resist attacks based on imitating the browser UI.
15. Help the user reliably identify websites.
16. *Break the habit of entering passwords into webpages.*

## 3. BACKGROUND AND RELATED WORK

### 3.1 Password Management

*Password databases* are a component of most modern browsers, including Firefox, Safari, Internet Explorer, and Opera. Mac OS X also includes a password database at the operating system level. These databases maintain a list of the user's usernames and passwords, usually stored encrypted on the user's computer. Typically, the user is prompted to decide whether or not each password should be stored. When the browser detects that it has returned to a site for which it knows a stored password, it automatically fills in the login form with the stored username and password. One drawback to this scheme is that the stored passwords are only available on one computer (Goal 5). Introducing a remote service to provide access to one's passwords would pose the additional risk of remotely storing passwords (Goal 11). Finally, password databases do not address the problem of using the same password at multiple sites (Goal 8).

*Password hashing* refers to the practice of hashing a fixed, secret string (the "master secret") together with a variable, non-secret string to produce a password. Different values for the variable part yield different hash values, thus generating many different passwords, each with about as much entropy as the master secret. Using a one-way hash makes it difficult for an attacker who obtains one of the generated passwords to determine the master secret. Password hashing allows the user to only have to memorize one secret (Goal 6), but still use a unique password for each account (Goal 8). Since the login passwords are computed, they don't have to be stored (Goal 11). This scheme also leaves open the possibility of storing the non-secret strings elsewhere, allowing the user to move from computer to computer (Goal 5).

Among the many implementations of password hashing are the Lucent Personalized Web Assistant [10], HP Site Password [14], and PwdHash [20]. LPWA is an HTTP proxy that manages usernames, passwords, and e-mail addresses to anonymize and protect the user's website accounts. At the beginning of a session the user enters a master secret; in login forms, the user enters "\U" as the username and "\P" as the password, which the LPWA proxy replaces with computed values. HP Site Password is a standalone tool that accepts a master password and a site name and puts the computed site password on the system clipboard so the user can paste it into a password field. PwdHash is an implementation of password hashing in Firefox. Whenever the user enters a password beginning with "@@", PwdHash hashes the password together with the domain name of the website before submitting it.

## 3.2 Password Strength

Although the entire space of possible passwords may be very large, the attacker in a dictionary attack reduces the search space by making assumptions about how passwords are commonly chosen — for example, that they contain natural-language words, proper names, dates, and so on. If the user's password fits those assumptions, the attacker may hit upon the password in a tractable length of time. Increasing the amount of work required for an attacker to check each guess can improve resistance to dictionary attack (Goal 9).

Abadi, Lomas, and Needham [1] suggested that a user's chosen password can be strengthened by appending a random supplement of a fixed length. The random supplement is never stored; every login attempt requires searching for the value of the supplement.

Kelsey, Schneier, Hall, and Wagner [15] proposed a scheme called *key stretching* in which the password must be hashed repeatedly a fixed number of times before it can be used. Halderman, Waters, and Felten [12] refined this scheme in a tool called Password Multiplier, which caches an intermediate hashing result on the user's machine to make logins faster for valid users. Passpet uses a further refinement of this scheme in which the amount of hashing is variable so that it can keep up with increases in computing power (Goal 10).

## 3.3 Anti-Phishing Measures

*Blocked site lists* are a popular response to the phishing problem. In such schemes, a single central database maintains a list of fraudulent sites; browsers check this database before proceeding to a site. Earthlink [7] and Netcraft [17] both provide these services, implemented on the client by a browser toolbar. Microsoft's published plans for Internet Explorer 7 [9] include a coloured indicator in the address bar that shows the status of the current site in Microsoft's database. These approaches can prevent phishing attacks if fraudulent sites are discovered and listed quickly (Goals 13 and 14), but they demand universal trust in a single authority and centralize vulnerability to failure (Goal 12). A centralized blocking list also ignores the possibility of honest differences in opinion among users — to choose a recent example, there has been some controversy about whether Claria belongs on spyware lists [16]. Finally, there are significant privacy problems with any scheme in which many clients send browsing information to a central authority.

*Site information indicators* provide information about the site in the browser toolbar or status bar. The URL field is an indicator already present in all browsers; in theory, a user could check the domain name in the URL to avoid phishing attacks, but in practice, the URL bar provides little protection. Most of the participants in a recent study [5] were fooled by misleading domain names (such as `www.bankofthevvest.com`, which may be visually mistaken for `www.bankofthewest.com`, or `www.paypal-signin03.com`, which may seem to be associated with `www.paypal.com`).

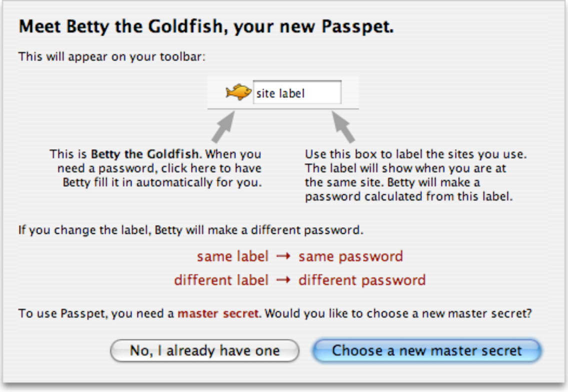
SpoofStick [3] is a browser extension that displays the current site's domain name in large letters in the toolbar to make the identification task a bit easier, but it still relies on the user's ability to distinguish legitimate and illegitimate domain names. Firefox displays the domain name of the SSL certificate in the status bar when a secure connection is present. However, even a perfectly competent expert user can be fooled by *homograph attacks* [11], in which the illegitimate domain name contains Unicode characters that are visually indistinguishable from their legitimate counterparts.

Instead of relying on the correctness of the domain name, some approaches rely on the SSL certificate authority. A browser extension called TrustBar [13] and the design for Internet Explorer 7 [9] both offer a toolbar indicator that displays the name of the certificate authority and the site's name in the SSL certificate. It is not clear whether the name of the CA will be meaningful to users, and the displayed site name remains susceptible to homograph attacks.

Some anti-phishing schemes employ *visual customization* of the login form, making the form look different for each user so that it will be hard to attackers to imitate accurately. The SiteKey feature [19] currently deployed by Bank of America displays a user-selected image and "image title" with the login form, in the hope that the user will notice when an attacker tries to imitate the login form but fails to display the correct image. Another scheme called Dynamic Security Skins [4] transparently overlays a user-selected image over the entire login form and relies on a new login protocol, SRP, for securing the authentication process.

A *petname system* is a system that lets a user assign *petnames* (local labels) to objects, and then translates global (or other) identifiers for objects into the user's local namespace of petnames [22]. Though the "petname" moniker is more recent, the concept is old and familiar. For example, when a mobile phone rings with an incoming call, and the caller's phone number is in the phone's contact list, most phones will translate the caller's number (the global identifier) into the caller's name (the user's local identifier). In instant messaging systems such as ICQ or AIM, each user has a unique number or username in a global namespace. Although users initially contact each other using this global identifier, most IM clients let the user assign "friendly names" to other users; thereafter, the IM client refers to the other user using the locally assigned name instead of the global identifier.

A Firefox extension called the Petname Tool [2] lets users



**Meet Betty the Goldfish, your new Passpet.**

This will appear on your toolbar:

This is **Betty the Goldfish**. When you need a password, click here to have Betty fill it in automatically for you.

Use this box to label the sites you use. The label will show when you are at the same site. Betty will make a password calculated from this label.

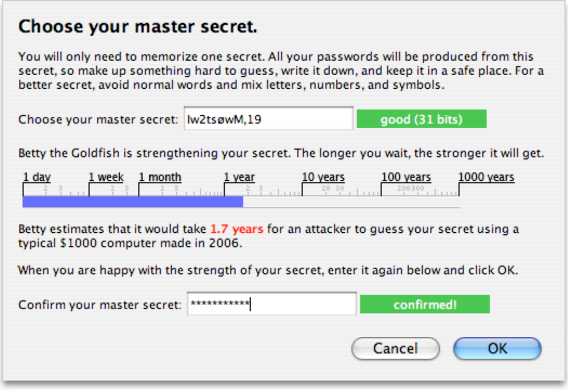
If you change the label, Betty will make a different password.

same label → same password

different label → different password

To use Passpet, you need a **master secret**. Would you like to choose a new master secret?

Figure 2: The user is introduced to a new Passpet.



**Choose your master secret.**

You will only need to memorize one secret. All your passwords will be produced from this secret, so make up something hard to guess, write it down, and keep it in a safe place. For a better secret, avoid normal words and mix letters, numbers, and symbols.

Choose your master secret: lw2tsøwM,19 good (31 bits)

Betty the Goldfish is strengthening your secret. The longer you wait, the stronger it will get.

Betty estimates that it would take 1.7 years for an attacker to guess your secret using a typical $1000 computer made in 2006.

When you are happy with the strength of your secret, enter it again below and click OK.

Confirm your master secret: *********** confirmed!

Figure 3: As the user waits, the attack time grows.

securely assign labels to websites. TrustBar [13] allows users to assign logo images as well as labels. When logging in, users can check the displayed site label to ensure that they are at the intended site. In addition to helping users detect imitation websites (Goal 13), petnames also enable users to identify sites and tell them apart (Goal 15).

# 4. DESIGN

We first describe how Passpet looks and works from the user's point of view; the next section will describe the underlying mechanisms. When we mention "Passpet," we are referring to the most recent version of Passpet, which is an extension to the Firefox browser. (There is also an earlier implementation of Passpet for Internet Explorer that provides only a subset of this functionality.)

## 4.1 Setup

After the Passpet Firefox extension is installed, the ensuing setup procedure has three steps:

1. The user is asked to enter a master address in the form *username*@*hostname*, where *hostname* identifies a Passpet server.

2. A random icon is automatically chosen from a set of animal icons and the animal is given a random name. The name and icon form Passpet's *persona* for interacting with the user (Figure 2).

3. The user chooses a master secret (Figure 3). A progress bar shows an estimate of how long it would take for an attacker to guess the secret. The attack time changes as the user types the secret and also increases while the user waits. When the user finds the attack time sufficiently large, the user confirms the secret by entering it again, then clicks "OK."

## 4.2 Everyday Use

Passpet appears as a toolbar button next to a text field in the browser toolbar. The image and text on the toolbar button are the icon and name of Passpet's persona.

1. When the browser starts, Passpet's persona is initially "sleeping" (Figure 4).

2. To "wake up" the persona, the user clicks on it and enters the master secret (Figure 5). Passpet identifies the persona in its request for the secret. The persona maintains the master secret in memory until the user puts it back to sleep.

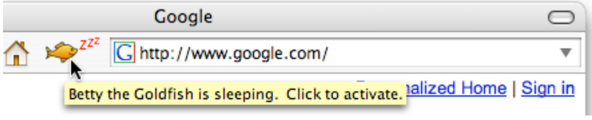
3. To fill in a password, the user clicks the Passpet button (Figure 1). If the cursor is in a password field, that password field is filled in; otherwise if there is a login form on the current page, it is filled in. If there is a username field paired with the password field and a username was previously entered there to log in, Passpet remembers and fills in the username as well.

If the user is using Passpet on a new machine to access previously established passwords, the user skips the secret-selection step (step 3 of setup). When the new persona is awakened for the first time, it will pause for about the same length of time that the user waited during setup. Thereafter, the persona awakens instantly.

## 4.3 Managing Relationships

The Passpet text field displays the user's *site label* for the current site, or "unknown" if the user has not assigned a label (Figure 6). For non-SSL sites, the text field has a pink background and the site label is displayed with a question mark to show that the site's identity is not verified. For SSL sites, the text field has a yellow background when the site is unlabelled, and a green background when the site is labelled. (Tooltips on the text field explain what these states mean.)

The user assigns a site label by clicking in the text field

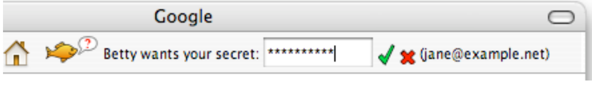


Google

http://www.google.com/

Betty the Goldfish is sleeping. Click to activate.

Figure 4: The Passpet persona is initially sleeping.



Google

Betty wants your secret: ********** (jane@example.net)

Figure 5: The user enters the master secret.

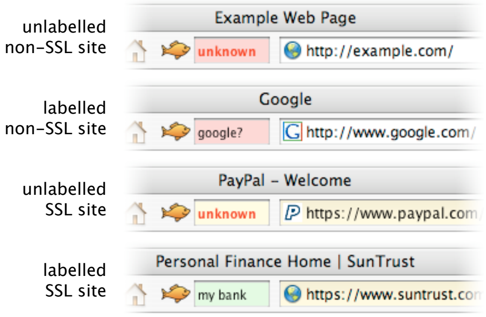
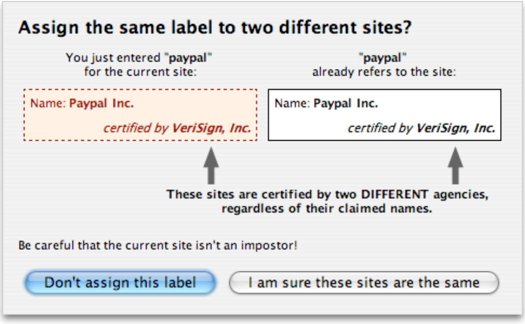


**Figure 6: The four states of the site label.**



**Assign the same label to two different sites?**

You just entered "**paypal**" for the current site:

"**paypal**" already refers to the site:

Name: **Paypal Inc.** *certified by **VeriSign, Inc.***

Name: **Paypal Inc.** *certified by **VeriSign, Inc.***

**These sites are certified by two DIFFERENT agencies, regardless of their claimed names.**

Be careful that the current site isn't an impostor!

**Figure 7: Passpet warns about a site label collision.**

and typing; the site label can also be edited in place there. If a site label is created or edited so that it duplicates an existing site label, Passpet notifies the user of the name collision (Figure 7) and displays information about the colliding sites. When both sites have SSL certificates, the label collision warning also indicates whether both certificates were issued on behalf of the same root CA (according to its public key), to indicate when an attacker might be trying to impersonate a CA (Figure 7).

- When setting up an account with a new site, the user enters a site label in the Passpet text field. On the account registration page, the user clicks on Passpet to fill in the new password.
- To start using Passpet with an existing account (Goal 3), the user enters a site label in the Passpet text field. On the site's "change password" page, the user enters the current password, then clicks on Passpet to fill in the new password.
- To change the password for a site (Goal 4), the user visits the site's "change password" page. The user clicks on Passpet to fill in the old password, changes the site label in the text field, then clicks on Passpet to fill in the new password.
- To change the master secret (Goal 7), the user asks for a new Passpet persona, yielding a second button and text field. The user can migrate a site to the new secret by going to the site's "change password" page, clicking on the old persona to fill in the old password and the new persona to fill in the new password. When the user is done with the old secret, the old persona can be discarded.

## 5. MECHANISM

### 5.1 Variable Appearance

To make the Passpet user interface hard to imitate, it has no standard icon. The Passpet button is the only way to activate Passpet, and the button icon and button text differ from user to user.

### 5.2 Site Identification

Passpet associates the site label entered by the user with a *site identifier* consisting of three parts: $(root\_key$, $field\_name$, $field\_value)$, following the scheme used by the Petname Tool [2] for SSL sites.

For SSL sites, $root\_key$ is the fingerprint of the root certificate authority's public key. If the site's SSL certificate has an Organization Name field, then $field\_name$ is "O" and $field\_value$ contains the Organization Name. Otherwise, $field\_name$ is "CN", and $field\_value$ contains the certificate's Common Name instead. If the user assigns the label "my bank" to a bank's SSL site, another site can only cause the site label "my bank" to appear if it can obtain a certificate with a certificate chain ultimately signed by the same root authority and which yields the same site identifier.

For non-SSL sites, the $root\_key$ is empty, the $field\_name$ is "D" (for domain), and the site identifier is the last $n + 1$ levels of the domain name when the domain name ends in a $n$-level top-level domain (TLD). For example, `.com` is a single-level TLD, so the site identifier for `www.subdomain.example.com` is `example.com`. However, `.mb.ca` is a two-level TLD, so the site identifier for `www.gov.mb.ca` is `gov.mb.ca`. Passpet uses a list of two-level TLDs to make this determination. The intention is that the site identifier refers to the domain at the level of specificity where domains are usually purchased and owned, so that sites like `www.example.net`, `example.net`, and `server2.example.net` will get the same label.

The site label is allowed to contain Unicode characters so that it can be written in the user's own language. Homographs (similar-looking glyphs) in Unicode are not a security vulnerability in site labels, since the site labels are supplied only by the user, not an external source.

### 5.3 Password Generation

To generate passwords, Passpet employs the two-level iterated hashing strategy of Halderman, Waters, and Felten's Password Multiplier [12]. $k_1$ and $k_2$ are parameters that control the degree of resistance to dictionary attack.

When the master secret is first entered on a particular machine, Passpet computes

$$V = H^{k_1}(master\_address \mid\mid \text{“\textbackslash 0”} \mid\mid master\_secret)$$

where $H$ is a secure hash function to be iterated $k_1$ times and "\0" is a single null byte used as a separator. The value $V$ is then cached on the machine for future use.

In order to fill in a password field, Passpet computes

$$P = H^{k_2}(site\_label \mid\mid \text{“\textbackslash 0”} \mid\mid master\_secret \mid\mid \text{“\textbackslash 0”} \mid\mid V)$$

where the $site\_label$ is the user-assigned label for the site. $V$ and $P$ depend on the $master\_address$ in order to prevent an attacker from precomputing a table of common $master\_secret$ values with corresponding $P$ values.

In our implementation, the hash function $H$ is SHA-256. To meet the password format requirements of most websites, Passpet generates passwords that always contain a digit, a lowercase letter, and an uppercase letter. Many sites do not permit characters other than letters and digits in passwords; for the relatively few sites that require punctuation in passwords, the user will need to hit a punctuation key after clicking on Passpet.

Passpet converts $P$ into a site password string by taking the $n$ least significant digits of the base-62 representation of $P$, using characters in the set `[0-9a-zA-Z]` as digits. $n$, the length of the site password, is the number of characters allowed in the password field (as specified by the `maxlength` attribute of its <`input`> tag), up to maximum of 12. If this password string does not include at least one digit, one lowercase letter, and one uppercase letter, then Passpet computes $P' = H(P)$, $P'' = H(H(P))$, and so on, until it arrives at a password string that contains all three types of characters.

Passpet permits any Unicode characters to be used in the master secret, to allow the possibility of greater password entropy. For these hash computations, the master secret and site label are encoded in UTF-8.

## 5.4 Variable Strengthening

The computation of $V$ and $P$ above are the same as in Password Multiplier; Password Multiplier uses fixed values for $k_1$ and $k_2$, where $k_1$ can be much larger because it only incurs a one-time delay per machine, whereas $k_2$ incurs a delay for each login attempt. In our scheme, $k_1$ is variable and $k_2$ is fixed. The setup procedure repeatedly hashes the master address and master secret, allowing $k_1$ to increase for as long as the user is willing to wait. In other words, $k_1$ is limited only by the user's patience.

As the user enters the master secret during the setup procedure, Passpet makes a rough estimate of the entropy in the secret. This entropy value is used together with $k_1$ and an estimate of the attacker's computing power, in hashes per second, to calculate the average time required for a successful dictionary attack. As computers become faster, the estimate of the attacker's computing power can be updated without changing the rest of the software.

## 5.5 Local Storage

While a Passpet persona is awake, it maintains the list of site labels in memory; each entry in the list is a pair $(site\_identifier, site\_label)$. The list of site labels is saved locally on disk in a file containing the encrypted list and a message authentication code (MAC):

$$site\_label\_file = E_{W_1}(site\_label\_list) \mid\mid M_{W_2}(site\_label\_list)$$

where $E$ is AES-CBC, $M$ is AES-CMAC, and

$$\begin{aligned} W &= H^{k_2}(master\_secret \mid\mid \text{“\textbackslash 0”} \mid\mid V) \\ W_1 &= high\ 128\ bits\ of\ W \\ W_2 &= low\ 128\ bits\ of\ W \end{aligned}$$

Passpet keeps a cache on the local disk for each persona. The cache contains $master\_address$, $index$ (see next section), $V$, and $site\_label\_file$. A persona that knows its $index$ and $V$ is said to be *initialized*. When the user removes a persona from the toolbar, its cache is deleted.

For completeness, we note that the inputs to $H$ used for computing $V$, $W$, and $P$ can be uniquely decomposed into their constituent parts because the strings $master\_address$, $master\_secret$, and $site\_label$ cannot contain null bytes and $V$ has a fixed length.

## 5.6 Remote Storage

To regenerate the correct passwords on another computer, Passpet needs to know the master address, master secret, $k_1$, $k_2$, and the site label. The master address and master secret are entered by the user and $k_2$ is fixed.

Passpet relies on a remote server to store $k_1$ and a copy of the site label file. The master address is in the same format as an e-mail address, *username*`@`*hostname*, which allows the user to choose the storage server. Access to the file is authenticated using the Secure Remote Password protocol [24]. The storage server keeps a set of six-item records in the form:

$$(username, index, k_1, salt, verifier, site\_label\_file)$$

where $index$ is an integer and $salt$ and $verifier$ are used for SRP authentication. The purpose of $index$ is to let one user store multiple files during the process of changing from an old master secret to a new master secret.

The server supports two unauthenticated commands:

- `create`($username$, $k_1$, $salt$, $verifier$)
  The server picks a new $index$ such that ($username$, $index$) is not among its records, creates a new record with an empty $site\_label\_file$, and returns $index$.

- `list`($username$)
  The server returns a list of ($index$, $k_1$) pairs for all the records with the given $username$.

To begin an authenticated session, the client selects a record by sending ($username$, $index$) and carries out the SRP protocol to authenticate against the $salt$ and $verifier$ in that record. In the encrypted SRP session, the server accepts one of three commands and terminates the session after completing the command:

- `delete`()
  The server deletes the record.

- `read`()
  The server returns the record's $site\_label\_file$.

- `write`($old\_mac$, $site\_label\_file$)
  If the MAC at the end of the record's $site\_label\_file$ matches $old\_mac$, the server atomically replaces the $site\_label\_file$. Otherwise, it returns a failure message.

When the user initializes a persona with a new master secret, the strengthening process yields $k_1$. Passpet then determines $salt$ and $verifier$ according to SRP, using $username$ as the SRP username and $W$ as the SRP password. It then sends a `create` command to the server and caches the returned $index$, yielding an initialized persona.

When the user awakens an initialized persona, it computes $W$ from its cached $V$ and the entered $master\_secret$. It authenticates with the server using its cached $username$ and $index$ and this $W$, issues a `read` command to get the site label file, and checks the MAC on the file.

To start using Passpet on another machine, the user creates a persona without choosing a new master secret, yielding an uninitialized persona.

The first time an uninitialized persona is awakened, Passpet issues a `list` command to the server and makes a login attempt for each ($index$, $k_1$) in the returned list until a login succeeds. At this point the persona caches $index$ and $V$ and becomes initialized; the persona issues a `read` command to get the site labels, as before.

When changes are made to the site labels, Passpet encrypts the updated site labels and sends them back to the storage server with a `write` command. Because the database is fairly small (perhaps 40 to 80 bytes per site), it is feasible to transmit the entire database in a single transaction. If the `write` fails because the MAC has changed since the last `read`, the persona issues another `read` to get the server's current $site\_label\_file$, reapplies the changes made since the last successful `write`, and retries the `write`.

# 6. SECURITY ANALYSIS

This section examines Passpet's effects on four classes of attacks: dictionary attacks against the user's passwords, attacks against the user's site label file, phishing attacks intended to fool the user into revealing site-specific passwords, and spoofing attacks intended to fool the user into revealing the master secret. (For the purposes of this discussion, "phishing" refers to an attack that imitates a website, whereas "spoofing" refers to an attack that imitates some other part of the user interface.)

We also discuss Passpet's design with regard to cross-site login forms and cross-site scripting issues.

## 6.1 Dictionary Attacks

Here we analyze the effort required for various types of attackers to guess the user's password. The previously published analysis of Password Multiplier [12] covers the following cases:

**Case 1: Attacker has no information.** The attacker can perform only an online attack on the password at a particular site. The target site can impede the attack by limiting the rate of login attempts or disabling an account after some number of failures.

**Case 2: Attacker has obtained one or more site passwords.** The attacker can perform an offline attack against the master secret, costing at least $k_1 + k_2$ hash computations for each guess.

**Case 3: Attacker has obtained the cached value of $V$.** The attacker can perform an offline attack against the master secret, costing at least $k_1$ hash computations for each guess.

**Case 4: Attacker has obtained $V$ and one or more site passwords.** The attacker can perform an offline attack against the master secret, costing at least $k_2$ hash computations for each guess.

To the Password Multiplier scheme, we have added the remote storage of the site label file. Since the SRP password $W$ depends on the master secret, an attacker who doesn't know the master secret cannot authenticate with the storage server (even if the attacker has $V$ or one or more site passwords). So, unless the attacker has insider access to the storage server, the four cases above apply.

What if the attacker has broken into the storage server, or the owner of the storage server is malicious? In this case, the attacker has access to a file that has been encrypted with $W$, and can conduct an offline dictionary attack against the master secret. For each guess at the master secret, the attacker performs $k_1$ hashes to obtain $V$ and $k_2$ more hashes to obtain $W$, then attempts to decrypt the file, then examines whether the result yields the correct MAC. The total number of hashes per guess is $k_1 + k_2$, the same as Case 2 above.

If the attacker has obtained the cached value of $V$ as well as the data on the storage server, then a faster offline dictionary attack is possible. For each guess at the master secret, the attacker performs $k_2$ hashes to compute $W$ and then attempts decryption. The number of hashes per guess is $k_2$, the same as Case 4 above.

Thus, obtaining a copy of the remotely stored data offers about the same benefit to an attacker as obtaining a site password. With respect to dictionary attacks, the storage server can be considered just another site where the user has an account, since a break-in or malicious administrator at the storage server incurs no more risk than a break-in or malicious administrator at any other site where the user has an account.

We conclude that Passpet provides resistance to dictionary attacks that is about as strong as Password Multiplier for the same value of $k_1$, and can be stronger, if the user waits long enough to obtain a larger $k_1$. If the user's patience remains more or less constant, then as the user periodically upgrades to new computers and updates the master secret, $k_1$ will increase to track increases in computing power.

## 6.2 Attacks on the Storage Server

We do not consider the user's site labels themselves to be secret, since the secrecy of site passwords depends on the master secret. However, the entire list of sites where the user has accounts does reveal information about the user's browsing habits, which poses a privacy risk. The site label file is stored encrypted so that the user's privacy remains protected even if the storage server is compromised or the owner of the storage server is untrustworthy.

Applying a MAC to the site label file prevents others from altering the list of site labels. Even though the server owner can modify the file itself, no one can produce a valid altered file without the key $W$. If the server owner corrupts the file, the client will detect that the MAC at the end of the file does not match, so this would merely be equivalent to denying service.

Access to the user's file is authenticated in order to prevent attackers from getting a copy of the encrypted file to use for a dictionary attack against the master secret via $W$, and to prevent attackers from corrupting or erasing the data. Again, write access would not allow an attacker to make a valid change to the site label list, but would allow an attacker to erase it; authentication prevents this type of denial-of-service attack.

What repercussions are there when the storage server is unavailable? If the user attempts to use Passpet on a new computer for the first time, Passpet will fail to log in and will be unable to determine $k_1$. However, in all other cases, Passpet has a local copy of the site label file and the cached value of $V$. Thus, after the user enters the master address and master password, Passpet can still function (although some site labels may be out of date and updates to the site labels will not be stored).

## 6.3 Phishing Attacks on Site Passwords

In a typical phishing attack, the user is lured to an impostor webpage that looks like the login page for the target website. Believing it to be the actual target website, the user logs in, and in the process submits a password to the attacker.

If the user is using Passpet to manage the password for a particular site, then the user never types or sees the password; the password is always generated by clicking on the Passpet button. So the user is incapable of actually typing in the password to give to an attacker.

Therefore, to steal a site password, an attacker must cause Passpet to generate the password. The password depends on the site label, and the site label is associated with the SSL certificate. It follows that the attacker must either (a) modify Passpet's site label file to associate the attacker's SSL certificate with the target's site label, (b) obtain an SSL certificate that Passpet recognizes as associated with the same site label, or (c) convince the user to assign the same site label to the attacker's site.

To achieve (a), the attacker has to make a coherent change to the user's site label file, which can only be achieved if the attacker obtains the key $W$, as explained in the preceding section.

In case (b), the SSL certificate must have exactly the same Organization Name (or Common Name, if the original had no Organization Name) and have a signing chain ultimately signed by the same root CA (certificate authority). The attacker must either steal a legitimate certificate with its private key, or convince some CA authorized to act on behalf of the same root CA to issue another certificate with the same name. It is reasonable to consider secure websites responsible for protecting their own private keys, and we consider it outside of Passpet's scope to prevent CAs acting on behalf of the same root authority from issuing duplicate certificates.

Consider case (c). To support our claim that Passpet reduces the risk of phishing attacks, we argue that it is more difficult for an attacker to persuade a Passpet user to reassign an existing site label to the attacker's site than to fool a non-Passpet user into filling in an imitation login form.

Today's phishing attacks depend on repeating a familiar login process that users go through all the time: the user sees the login form, types in the username and password, and submits the form. The perceived user experience of being phished is the same as the perceived user experience of legitimately logging in.

On the other hand, being asked to reuse an existing site label is an unusual request, and assigning the site label requires more effort than logging in normally. In the event that the user actually does edit a site label to collide with an existing site label, Passpet warns the user (see Figure 7). We expect that legitimate situations requiring reuse of an existing site label will be quite rare; this expectation has been confirmed so far by the fact that users of Passpet for IE have never needed to assign the same label to two different sites. Based on further experience with deploying Passpet, we may adjust the intrusiveness of the site label collision warning, or forbid collisions altogether to prevent any possibility of success for this type of attack.

## 6.4 Spoofing Attacks on the Master Password

An attacker could also try to fool the user by spoofing the browser interface. For example, past attacks have used JavaScript to hide the browser address bar and replace it with a convincing image of an address bar to make the user think the site is at a legitimate domain.

With Passpet in use, password security no longer depends on the address bar, but it is conceivable that an attacker might try to imitate the Passpet user interface in order to get the user to enter the master secret. Passpet obstructs this type of attack in three ways:

1. The Passpet user interface component is recognized by its icon. Since the icon is randomly selected on installation and differs from user to user, the attacker is less likely to be able to produce a convincing replica.

2. When the user clicks on the icon, Passpet requests the secret using the customized persona name, for example, "Betty wants your secret:". Since this name is also different from user to user, the attacker is less likely to be able to produce the correct label.

3. The user enters the master secret only once per session, and only after clicking on the button, never upon external prompting.

The purpose of the Passpet persona is to create a specific trust relationship between the user and the Passpet tool. The Passpet interface is intentionally minimal: Passpet is to be recognized by its icon, unlike password forms, which are recognized by the presence of two text fields labelled "username" and "password." The customized icon is a direct part of the workflow — the user is reminded of the icon every time he or she clicks on it, and is therefore less likely to use a spoofed Passpet tool.

## 6.5 Cross-Site Issues

Some websites have pages containing login forms that submit to a different target website. Since Passpet's label is associated with current SSL certificate, the label identifies the source of the page, not the form target. We argue that this is a better choice from a security perspective, because the purpose of the label is to help the user make trust decisions about the visible page, not the invisible form target. For example, in a phishing attack, it is the page that misleads the user, not the form target. If the label identified the form target, an attacker could gain the appearance of legitimacy simply by creating a fraudulent page that points at the same target.

Since the site password is generated from the site label, it also corresponds to the page and not the form target. Again this is the safer option, since any information entered on the page is vulnerable to JavaScript commands in the page. If the site password corresponded to the form target, an attacker could steal a site password by creating a page that points at the same target with some JavaScript that reads the contents of the login form.

Even though it is possible for legitimate organizations to have login forms at two different sites (say, A and B) that submit to the same target site C, site A and site B will have independent site labels. This is as it should be, since the form at site B could just as well be a page containing JavaScript designed to capture passwords from users who believe they are logging in at site A. For unusual cases like this, the user can choose to give the same label to A and B after inspecting their certificate information (Figure 7).

In a “cross-site scripting” attack, an attacker injects JavaScript code into a page on the target site itself (for example, by supplying it as a query parameter or posting it on a message board). Passpet has no effect on these attacks; it is up to the site administrator to make sure that the site itself does not attack its own users.

## 7. USABILITY ANALYSIS

Passpet’s most obvious advantage is the improved convenience of logging in. Instead of having to enter a username and password for every login, the user only has to enter the master secret once in a browser session and can thereafter fill in login forms with a single click.

According to a Pew Internet Research study in 2000 [8], 68% of users use multiple different passwords when registering at websites. A survey in 2003 conducted by Protocom Development Systems [23] found that over 64% of corporate users memorize more than five passwords for their work. Therefore, most users would benefit from only having to memorize one master secret instead of many passwords.

Another advantage that Passpet provides is the ease of generating new passwords. When faced with an account registration form, users must take a moment to decide which password to use for the new account, or think about whether they should invent a new password. Entering a site label and clicking on Passpet is more convenient than choosing a password and typing it in.

For users that change passwords periodically or sites that require password changes, Passpet provides a significant improvement in convenience. The Protocom survey reported that 68% of corporate users change their passwords at least once a year, and half of those (34%) did so once a month or more. With Passpet, the user can generate a completely new password by editing the site label (for example, by appending or incrementing a number). After changing a password, the user doesn’t have to remember that it was changed; upon returning to the website, the new site label automatically reappears.

The earlier implementation of Passpet, an extension to Internet Explorer, has been informally tested in a deployment to 15 users at HP Labs. Feedback was very positive; one user even wrote to say, “Passpet has changed my life. No longer do I agonize with passwords... it is so easy!” After three months, ten of the users are still using Passpet for IE regularly. The main complaint from those who stopped using it was that it’s inconvenient to use on multiple machines, a problem addressed by Passpet for Firefox. Passpet for IE includes the site labelling and password hashing features but not the password strengthening and customized icon, and it only supports SSL sites. Passpet for IE has a very similar user interface to the current Firefox implementation, though it pops up a separate dialog window to request the master secret, whereas Passpet for Firefox does this in the toolbar area for improved convenience and security.

We are currently designing a formal usability study of Passpet for Firefox. The study will compare the effectiveness of a standard installation of Firefox, Firefox with the Passpet extension, and at least one other anti-phishing alternative. Effectiveness will be evaluated by measuring subjects’ performance, error rate, and level of satisfaction at four tasks: registering for new accounts, logging in to accounts, changing account passwords, and detecting phishing attacks. In addition, we plan to test two hypotheses of our design: (a) that users will be less likely to proceed when presented with the wrong persona icon or persona name; and (b) that having to click on the Passpet button during the login process makes the user more likely to notice the site label (in comparison with other tools that offer indicators in the toolbar area).

## 8. EVALUATION AND COMPARISON

In this section, we evaluate Passpet in terms of the 16 goals we originally listed, and compare Passpet to a representative sample of other password management and anti-phishing tools. Refer to Figure 7 for a table that summarizes our findings. The first column, labelled “plain browser,” is a baseline condition — a browser that has no password management or anti-phishing tools added, with the password autofilling feature turned off. The remaining columns describe various other tools that address these problems; each one was described briefly in Section 3.

**Goal 1. Improve the convenience of logging in to websites.** Only password autofilling, LPWA [10], Password Multiplier [12], and Passpet enable users to log in with less effort than the standard login procedure of typing in a username, typing in a password, and clicking a button. With autofilling, the user doesn’t have to do anything to have the password field filled; with LPWA, the user types in “\P” instead of a password; and with Passpet, the user clicks a button. HP Site Password [14] and PwdHash [20] require slightly more effort than a normal login, as the user has to perform an activation step (clicking a button or pressing a key combination) before typing in a password.

With Password Multiplier, the user double-clicks in the password field to pop up a dialog box; by default the user has to re-enter the master secret for each login. However, the user can ask Password Multiplier to remember the master secret, and then filling in a password only requires a double-click in the password field followed by a click on the dialog’s “OK” button.

The remaining tools leave the login procedure itself unchanged, though SpoofStick, the Netcraft Toolbar, the Earthlink Toolbar, SiteKey, DSS, the Petname Tool, TrustBar, and IE7 all expect users to check an additional indicator if they want to avoid being phished.

**Goal 2. Work with existing websites and login forms.** All the tools work with unmodified websites except for SiteKey [19] and DSS with SRP [4]. SiteKey requires the website to switch to a two-stage login process with a custom image displayed in the second stage; DSS with SRP involves switching to a new cryptographic authentication protocol.

**Goal 3. Permit site-by-site migration to using the tool.** All the tools allow the user to start using them with existing accounts one at a time.

**Goal 4. Allow the user to change passwords for individual sites.** LPWA generates passwords solely based on the user’s identity, secret, and the website domain name, so the only way to change any password is to change the secret, which changes all the passwords. PwdHash and Password Multiplier do not prevent the user from changing individual site passwords, but doing so conflicts with Goal 6. With PwdHash, the only way to change a site password is to enter a different password into the login form to be hashed; then the user will have to remember multiple secrets.

| | | plain browser | password autofill | LPWA | HP Site Password | PwdHash | Password Multiplier | SpoofStick | Netcraft Toolbar | Earthlink Toolbar | SiteKey | DSS with SRP | Petname Tool | TrustBar | IE7 with autofill | **Passpet** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **USABILITY** | | | | | | | | | | | | | | | |
| 1 | Make logging in more convenient | no | yes | yes | no | no | yes | no | no | no | no | no | no | no | yes | **yes** |
| 2 | Work with existing websites | yes | yes | yes | yes | yes | yes | yes | yes | yes | no | no | yes | yes | yes | **yes** |
| 3 | Allow site-by-site migration to tool | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | **yes** |
| 4 | Change individual site passwords | yes | yes | no | yes | if not 6 | yes | yes | yes | yes | yes | yes | yes | yes | yes | **yes** |
| 5 | Log in from other computers | yes | yes* | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes* | yes* | yes* | **yes** |
| 6 | Only need to memorize one secret | no | no | yes | yes | if not 4 | yes | no | no | no | no | no | no | no | no | **yes** |
| 7 | Enable changing the master secret | - | - | no | yes | yes | yes* | - | - | - | - | - | - | - | - | **yes** |
| | **SECURITY** | | | | | | | | | | | | | | | |
| 8 | Unique password for each site | no | no | yes | yes | yes | yes | no | no | no | no | no | no | no | no | **yes** |
| 9 | Resist offline dictionary attacks | no | no | no | no | no | yes | no | no | no | no | no | no | no | no | **yes** |
| 10 | Adapt to increasing CPU power | no | no | no | no | no | no | no | no | no | no | no | no | no | no | **yes** |
| 11 | Avoid storing passwords | yes | no | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | no | **yes** |
| 12 | Avoid a single central authority | yes | yes | yes | yes | yes | yes | yes | no | no | yes | yes | yes | yes | no | **yes** |
| 13 | Resist phishing by fake login forms | no | no | yes | no | yes | yes | maybe | maybe | maybe | maybe | maybe | maybe | maybe | maybe | **yes** |
| 14 | Resist mimicry of browser UI | no | no | no | no | no | no | no | no | no | no | yes | no | no | no | **yes** |
| 15 | Help the user identify websites | no | no | no | no | no | no | yes* | yes* | no | no | no | yes | yes | yes* | **yes** |
| 16 | Stop entering secrets in webpages | no | no | no | yes | no | yes | no | no | no | no | yes | no | no | no | **yes** |

**Figure 8: Comparison of password management and anti-phishing tools.**

Password Multiplier's dialog box contains text entry fields for both the master secret and the site's domain name, so the user could conceivably change a site password by editing the domain name. However, the dialog box comes up with the original domain name each time, so the user would have to re-edit the domain name every time. Thus, exercising the ability to change individual site passwords introduces an additional memory burden on the user.

**Goal 5. Allow the user to log in from more than one computer.** None of the tools prevent a user from logging in with another computer. However, the ones marked with an asterisk become less useful: Password autofilling and the site labelling features of the Petname Tool and TrustBar depend on locally stored information that is not carried over when the user switches to another computer.

**Goal 6. Let the user only have to memorize one secret.** The password hashing schemes (LPWA, HP Site Password, PwdHash, Password Multiplier, and Passpet) are designed to enable the user to have to memorize only one secret. For PwdHash, there is a conflict with Goal 4; if the user wants to memorize only one secret, then the user cannot change a single site's password without also changing all the others.

For the remaining tools, keeping to just one secret would mean having the same password for all sites. None of them prevent the user from doing this, but we have marked them "no" because in practice most users use multiple passwords [8, 23].

**Goal 7. Allow the user to change the master secret.** A change in the master secret affects all of the site-specific passwords, which must be changed one at a time at their websites. During the changeover process, some passwords will be based on the new secret and some on the old. Of the five schemes that support hashing, all but Passpet require the user to remember which ones use the new secret and which are still using the old secret.

HP Site Password and PwdHash let the user enter any master secret each time a site-specific password is generated. On each account's "change password" page, the user would enter the old master secret when filling in the "old password" field and the new master secret when filling in the "new password" field.

With LPWA, the user can end the current session and start a new session with a different secret, but only one master secret can be active at a time. Thus, it is not possible to get LPWA to fill in both the "old password" and the "new password" fields on a password change page, one using the old master secret and one using the new master secret.

With Password Multiplier, changing the master secret would require bringing up the Password Multiplier dialog to fill in the "old password" field, then "deauthorizing" the current session and "reauthorizing" with a different master secret, then bringing up Password Multiplier again to fill in the "new password" field. Because it takes about 100 seconds to perform each reauthorization, it would be tedious to repeat this process for every account.

With Passpet, the user creates a new Passpet persona with the new master secret. When the user is at a site that has not yet been changed over, the old persona will show a site label but the new persona will show that the site is "unknown". The user updates an account by entering a site label, clicking on the old persona to fill in the "old password" field, and clicking on the new persona to fill in the "new password" field. The user doesn't have to memorize which sites have already been changed over because the new persona will keep track of their site labels.

**Goal 8. Use a unique password for every site.** The password hashing schemes (LPWA, HP Site Password, PwdHash, Password Multiplier, and Passpet) all use cryptographic algorithms to generate site passwords that impede an attacker who has obtained one or more site passwords from finding the master secret or any other site password. The other tools allow users to choose the same password or related passwords for many sites.

**Goal 9. Resist offline dictionary attacks on user-**

**chosen secrets.** Only Password Multiplier and Passpet take password-strengthening measures to provide resistance against dictionary attacks.

**Goal 10. Adapt to the development of faster computers.** Assuming that the user is willing to wait for about the same length of time, Passpet's password-strengthening scheme increases in strength when the user upgrades to a faster computer and selects a new master password. In order for this adaptation to keep up with increases in the computing power of attackers, the user must upgrade and change passwords from time to time.

**Goal 11. Avoid storing passwords in long-term storage.** Password autofilling involves storing passwords on the local disk; the other schemes do not. None of the schemes store passwords remotely.

**Goal 12. Avoid introducing a centralized dependency.** The site rating features in the Earthlink Toolbar, Netcraft Toolbar, and Internet Explorer 7 refer to a single authority that dictates whether sites are good or bad. This introduces a central point of failure for the entire system, as well as a central locus of trust that not all users may be willing to accept. The Earthlink Toolbar downloads the list to the local machine [6], but the others collect information on the user's browsing history: the Netcraft Toolbar collects plaintext domain names and hashes of visited URLs [18]; Internet Explorer 7 sends URLs (with query strings removed) in real-time to Microsoft, except URLs on a client-side list of "known good" sites [21].

**Goal 13. Resist attacks based on fake website login forms.** For this goal we have classified some tools as "yes" and others as "maybe," depending what happens to a user who proceeds with normal login workflow. SpoofStick, the Netcraft Toolbar, SiteKey, DSS, Petname, TrustBar, and IE7 are all marked "maybe" because their defense against phishing relies upon the user to notice an indicator and respond with a change in behaviour. LPWA, PwdHash, Password Multiplier, and Passpet are marked "yes" because, even if the user attempts to log in, oblivious to an attack, the user's password is not sent to the attacker.

**Goal 14. Resist attacks based on imitating the browser UI.** Only DSS and Passpet customize the browser UI to protect it from being spoofed. We hypothesize that Passpet's customization of the button icon is more effective than the custom image overlaid on the login form in DSS. Our reasoning is that filling in the DSS login form isn't directly related to the image; the user's task can be described as "type into the two text fields," a task that can be performed just as well when the image is different or missing. However, Passpet's custom button icon is a way of identifying which button to click. If the persona is a giraffe, then we hypothesize that many users will conceptualize their task as "click on the giraffe," a task that cannot be carried out if there is no giraffe visible. Our planned user tests will help us understand whether our expectations are valid.

**Goal 15. Help the user reliably identify websites.** It is useful not only to protect the user's password but also to prevent the user from being misled as to the identity of the site. For example, if the user is using PwdHash and is lured to a phishing site that imitates the user's bank, PwdHash will fill in a different password than the real bank password, so the user's bank password is safe. However, the phishing site can still accept the login, continue to masquerade as the bank, and proceed to ask the user for private information. If the user believes that the browser has just successfully logged in to the bank's website, the user is vulnerable to attack.

SpoofStick, the Netcraft Toolbar, the Petname Tool, TrustBar, Internet Explorer 7, and Passpet all provide some kind of identifier for the website that is easier to read than a plain URL. However, they are marked with an asterisk because the identifier (domain name or certificate name) is provided by the website, and so could be chosen by an attacker to be misleading. The Petname Tool and Passpet use an identifier that is under the user's control. TrustBar displays both kinds of identifiers.

Of all these schemes, only Passpet incorporates clicking a button right next to the site identifier as part of the login process. We hypothesize that this significantly improves the likelihood of noticing an impostor site, for which the text field will show the word "unknown" or the wrong site label.

**Goal 16. Break the habit of entering passwords into webpages.** Training users to enter secrets into webpages leaves users highly vulnerable to phishing, since any aspect of a webpage's appearance can be imitated by an attacker. HP Site Password and Password Multiplier pop up a separate window for entering the master secret, but these windows could also be imitated by webpages. DSS with SRP uses a separate pane in the browser window, and Passpet uses part of the toolbar in the browser chrome area.

# 9. LIMITATIONS

As explained in section 5.3, $V$ and $P$ depend on $master\_address$, in order that an attacker cannot detect that different users have the same $master\_secret$ and $k_1$ and cannot precompute values of $V$ or $P$ for common master secrets. This has the unfortunate side effect that moving to a different storage server changes every site password — the effect is similar to choosing a new master secret. To avoid this problem, Passpet could use SRP's salt in place of $master\_address$ and define $V = H^{k_1}(salt \,||\, master\_secret)$.

Attacks on DNS ("pharming attacks") can hijack connections to non-SSL sites and steal their site passwords; Passpet does not reduce this inherent vulnerability of non-SSL sites.

Since the `list` command applied to a nonexistent username returns an empty list, an attacker can repeatedly query a server to determine which of a list of likely usernames actually exist. A smart server could detect and prohibit exhaustive querying from a single source.

The storage server makes $k_1$ available to anyone who asks, which enables an attacker who knows some usernames to choose the most vulnerable users when conducting a dictionary attack. An earlier design of Passpet avoided this weakness by requiring the client to try progressively larger values of $k_1$ until authentication succeeds. However, in such a scheme, a user initializing a Passpet persona on a new computer might never find out whether they mistyped their master secret, so we decided to make $k_1$ available to the client.

If the user initially sets up Passpet on a fast computer, they might choose a larger $k_1$ than they would have the patience for on a slow computer. They may find this frustrating if they later try to initialize a Passpet persona on a slow computer.

Finally, like all password-hashing systems, Passpet is vulnerable to an offline dictionary attack against the master secret from an attacker who has captured some site passwords — for example, the administrator of a site where the user has an account (see section 6.1). A few possible ways to provide additional security against this attack are:

1. Passwords transmitted to non-SSL sites are more vulnerable to being captured. One way to further protect SSL site passwords would be to keep two master secrets, one for non-SSL sites and one for SSL sites, so that capture of any non-SSL site passwords would not help an attacker obtain SSL site passwords.
2. Most password-hashing systems, including Passpet, impose some arbitrary limit on site password length. This limit could provide some additional security against this attack, at the cost of easier online dictionary attacks against any particular site. If the site passwords known to the attacker contain less entropy than the master secret, the attacker would have to perform an online test of each apparent hit from the offline dictionary attack. In the case of Passpet, the site password usually contains 12 base-62 characters, so it gives an attacker up to 71.1 bits of information. Most users' master secrets will contain less entropy.
3. If we were to relax Goal 11, another way to defend against these attacks would be to abandon password hashing, generate truly random site passwords, and store them with the site labels; then, captured site passwords would convey no information at all about the master secret.

## 10. FUTURE WORK

As described in section 7, we are planning a formal usability evaluation of Passpet.

We also plan to develop a version of Passpet that is implemented with HTML pages and unprivileged JavaScript, so that Passpet users can obtain their site passwords even on computers where they are not able to install software.

The password entropy estimator in Passpet is somewhat unrefined, and we hope to improve it. We would also like to let users choose their own images to use for the persona icon, and to expand the included set of available icons.

The current Firefox extension is entirely platform-independent, but we are considering adding platform-specific binary modules to speed up hash computations, to make Passpet stronger against dictionary attacks.

## 11. CONCLUSION

Passpet builds on Password Multiplier and the Petname Tool, also drawing upon ideas from Dynamic Security Skins and HP Site Password. The result is a practical, convenient tool that relieves users of the burden of choosing and memorizing many passwords, protects users' other accounts when some accounts are compromised, and reduces the risks of phishing and dictionary attacks, while addressing a wider range of real-world use cases than other website password security tools. We plan to make Passpet available for public download at `http://passpet.org/`.

## 12. ACKNOWLEDGEMENTS

Thanks to Alan Karp for conducting the preliminary user tests of Passpet for IE and to David Wagner for extremely helpful feedback on drafts of this paper and on Passpet's use of cryptography. We are also grateful to Tyler Close and Alan Karp for their contributions to Passpet's design.

## 13. REFERENCES

[1] M. Abadi, T. M. A. Lomas, and R. Needham. Strengthening Passwords. Technical Report 1997-033, SRC, 2005.

[2] T. Close. Petname Tool. `http://petname.mozdev.org/`.

[3] CoreStreet. Spoofstick. `http://www.spoofstick.com/`.

[4] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic Security Skins. In *Proc. 2005 Symposium on Usable Privacy and Securtiy*, pages 77–88, 2005.

[5] R. Dhamija, J. D. Tygar, and M. Hearst. Why Phishing Works. In *Proc. CHI 2006 Conference on Human Factors in Computing Systems*, 2006.

[6] Earthlink. Earthlink Toolbar and ScamBlocker FAQ. `http://kb.earthlink.net/case.asp?article=30492`.

[7] Earthlink. Earthlink Toolbar Featuring ScamBlocker for Windows Users. `http://www.earthlink.net/software/free/toolbar/`.

[8] S. Fox, L. Rainie, J. Horrigan, A. Lenhart, T. Spooner, and C. Carter. Trust and privacy online: Why Americans want to rewrite the rules. August 2000. `http://www.pewinternet.org/report_display.asp?r=19`.

[9] R. Franco. Better Website Identification and Extended Validation Certificates in IE7 and Other Browsers. November 2005. `http://blogs.msdn.com/ie/archive/2005/11/21/495507.aspx`.

[10] E. Gabber, P. B. Gibbons, Y. Matias, and A. Mayer. How to Make Personalized Web Browsing Simple, Secure, and Anonymous. In *Proc. Financial Cryptography 1997*. Springer-Verlag, February 1997.

[11] E. Gabrilovich and A. Gontmakher. The Homograph Attack. *Comm. of the ACM*, 45(2):128, February 2002.

[12] J. A. Halderman, B. Waters, and E. W. Felten. A Convenient Method for Securely Managing Passwords. In *Proc. 14th International World-Wide Web Conference*, 2005. `http://www.cs.princeton.edu/~jhalderm/projects/password/`.

[13] A. Herzberg and A. Gbara. TrustBar: Protecting (even Naïve) Web Users from Spoofing and Phishing Attacks. Cryptology ePrint Archive, Report 2004/155, 2004. `http://www.cs.biu.ac.il/~herzbea/TrustBar/`.

[14] A. Karp. Site-Specific Passwords. Technical report, HP Labs. `http://www.hpl.hp.com/personal/Alan_Karp/site_password/`.

[15] J. Kelsey, B. Schneier, C. Hall, and D. Wagner. Secure Applications of Low-Entropy Keys. *Lecture Notes in Computer Science*, 1396:121–134, 1998.

[16] R. Naraine. Microsoft Downgrades Claria Adware Detections. July 2005. `http://www.eweek.com/article2/0,1895,1834607,00.asp`.

[17] Netcraft. Netcraft Anti-Phishing Toolbar. `http://toolbar.netcraft.com/`.

[18] Netcraft. Netcraft Toolbar Privacy Policy. `http://toolbar.netcraft.com/privacypolicy.html`.

[19] Bank of America. Sign up for the SiteKey Service. `http://www.bankofamerica.com/privacy/passmark/`.

[20] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger Password Authentication Using Browser Extensions. In *Proc. 14th Usenix Security*, 2005.

[21] T. Sharif. Phishing Filter in IE7. September 2005. `http://blogs.msdn.com/ie/archive/2005/09/09/463204.aspx`.

[22] M. Stiegler. An Introduction to Petname Systems. `http://www.skyhunter.com/marcs/petnames/IntroPetNames.html`.

[23] Protocom Development Systems. Global Password Usage Survey. September 2003. `http://www.protocom.com/html/whitepapers/biz_password_survey.html`.

[24] T. Wu. The Secure Remote Password Protocol. In *Proc. 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, March 1998.