# Pharewell to Phishing : Secure Direction and Redirection over the Web

Taehwan Choi Sooel Son Mohamed G. Gouda
Department of Computer Sciences
The University of Texas at Austin
{ctlight, samuel, gouda}@cs.utexas.edu

Jorge A. Cobb
Erik Jonsson School of Engineering and Computer Science
The University of Texas at Dallas
cobb@utdallas.edu

**Abstract**

The conventional wisdom has always been that users should refrain from entering their sensitive data (such as usernames, passwords, and credit card numbers) into http(or white) pages, but they can enter these data into https (or yellow) pages. Unfortunately, this assumption is not valid as it became clear recently that, through human mistakes or Phishing or Pharming attacks, a displayed yellow page may not be the same one that the user has intended to request in the first place. In this paper, we propose to add a third class of secure web pages called brown pages. We show that brown pages are more secure than yellow pages especially in face of human mistakes and Phishing and Pharming attacks. Thus users can enter their sensitive data into brown pages without worry. We present a login protocol, called the Transport Login Protocol or TLP for short. An https web page that is displayed on the browser is classified brown by the browser if and only if this web page has been called into the browser either through TLP or from within another brown page that had been called earlier into the browser through TLP.

## 1 Introduction

When a user needs to display a web page on his browser, the user follows any one of four direction rules, described below, to request that his browser calls the page and displays it on the screen. If the requested page is an (insecure) http page, then the browser calls the page and displays it without any firm guarantee that the displayed page is the one that the user has requested. On the other hand, if the requested page is a (secure) https page, then the browser displays the page only after it has authenticated that the page is the one that the user has requested. Unfortunately, as described below, the authentication procedure is vulnerable to human mistakes, by the user, and to Phishing and Pharming attacks [11], by adversarial web sites. And so it is possible that the displayed page may not be the one requested by the user after all.

The user may not mind that the displayed page is different from the page that he has requested for two reasons. First, both the displayed page and the page that the user has requested have similar graphics and colors and the user may not notice that the displayed page is actually not the one that he has requested even in the presence of security indicators [3]. Second, the user may notice that the displayed page is not the one that he has requested, but he may believe that the displayed page is a legitimate redirection that was requested by the page that he has requested. In any case, the user may proceed to enter some sensitive data, such as his credit card number, into the displayed page which may happen to be an adversarial page.

This paper is dedicated to prevent these scenarios from occurring. Towards this end, we propose to introduce a new class of https web pages, which we refer to as brown pages. As discussed below, brown pages are secure against human mistakes and Phishing and Pharming attacks. Thus, when a user requests that his browser calls and displays an https page and then the browser displays the page and classifies it brown, the user knows that the displayed page is indeed the one that he has requested and so he can proceed to enter his sensitive data into it.

In order for the browser to be able to classify a called https page brown, the browser needs to call this page through a login protocol that is completely secure against human mistakes and Phishing and Pharming attacks. In this paper, we present and discuss the design and implementation of such a login protocol.

## 2 Attack Scenarios

In this section, we describe three attack scenarios, caused by human mistakes or Phishing or Pharming attacks[7, 10]. In each one of these scenarios, a user intends to call into his browser a particular https page, but he ends up calling a wrong https page into his browser.

1. **Human Mistakes**:
   A user intends to enter the URL `https://www.amazon.com` into the URL box of his browser. But he enters the wrong URL `https://www.anazon.com` by mistake.

2. **Phishing Attacks**:
   A user receives an email that urges the user to click on a link described as leading to the web site `https://www.amazon.com`. By clicking on this link, the user ends up in the wrong web site `https://www.anazon.com`.

3. **Pharming Attacks**:
   For the convenience of its users, the web site `https://www.amazon.com` allows its users to call the web site using the alternative insecure URL `http://www.amazon.com`. Now, the DNS of a user can be manipulated so that when the user uses this insecure URL to request the web site, the user's DNS directs the request to an adversarial web site that redirects the user's browser to the wrong web site `https://www.anazon.com`.

In each one of these three scenarios, the user intended to call into his browser the web site `https://www.amazon.com`, but he ends up calling the wrong web site `https://www.anazon.com`. The user does not notice the switch, from `https://www.amazon.com` to `https://www.anazon.com`, because the two web sites have similar logos, graphics, and colors, and maybe similar URLs. Thus the user proceeds to enter his sensitive information (such as username, password, or credit card numbers) into the wrong web site. The objective of this paper is to outline a proposal to counter these three attack scenarios.

One method to counter these scenarios is to advise the user to be careful and check the URL box of the displayed https web page on his browser before he enters his sensitive data into the displayed web page. However, it is very difficult for a user to remember and follow this advice every time he requests an https web page.

A second method to counter these scenarios is to make the browser check, before it displays an https web page, that this page is indeed the one that the user wants. Unfortunately, the browser can not tell whether or not the user wants the web page whose URL is in the URL box.

The method that we adopt in this paper to counter these scenarios is as follows. Browser $B$ of user $U$ displays an https page from a web site $S$ when and only when the following three conditions hold.

1. User $U$ has requested the page.

2. Site $S$ has verified that sometime in the past user $U$ has registered and stored his login data in site $S$.

3. Browser $B$ has verified that sometime in the past user $U$ has registered and stored his login data in site $S$.

If any one of these three conditions does not hold, then the browser of user $U$ refuses to display the requested page. The correctness of this method is based on the reasonable assumption that each web site in which user $U$ registers is a legitimate, rather than an adversarial, site. Next, we argue that this method can counter the above three scenarios.

Consider the first scenario. If user $U$ intends to request the web site `https://www.amazon.com`, but by mistake requests the web site `https://www.anazon.com`, then only one of two outcomes is possible. The most likely outcome is that user $U$ has not registered in the web site `https://www.anazon.com` and so the browser of user $U$ will not display the web page. The second outcome is that user $U$ has registered in the web site `https://www.anazon.com` and so the browser will display the legitimate web page of this site and user $U$ will notice that the displayed page is not the one that he wants.

Now consider the second and third scenarios. In these scenarios, the web site `https://www.anazon.com` is an adversarial site and so user $U$ has not registered in it. Thus, browser of user $U$ will not display the web page.

# 3 Countering the Attack Scenarios

In this section, we outline our proposal to modify the browser and some web sites in order to counter the attack scenarios, caused by human mistakes and Phishing and Pharming attacks, discussed in the previous section. Our proposal consists of three parts.

1. **White, Yellow, and Brown Pages**:
   We propose to modify the browser so that the browser classifies each displayed http web page as white, and classifies each displayed https web page as either yellow or brown. As described below, a user should regard each white page as insecure, each yellow page as mildly secure (which means that the page is vulnerable to human mistakes and Phishing and Pharming attacks), and each brown page as highly secure (which means that the page is secure against human mistakes and Phishing and Pharming attacks).

2. **A New Login Protocol**:
   We also propose to add a new login protocol to the browser and to some web sites that need to be (extra) secure against human mistakes and Phishing and Pharming attacks. We call this new login protocol the Transport Login Protocol or TLP for short. When a user invokes TLP on his browser and requests the browser to call a web page on a specified web site, the following three steps are executed. First, the browser and the specified web site use TLP to establish mutual authentication between each other. Second, if the mutual authentication between the browser and the web site succeeds, then the web site redirects the browser to an https web page. Third, the browser calls the secure web page and, upon receiving it, the browser assigns it a brown classification and displays it to the user.

3. **Classification of Web Pages**:
   The modified browser assigns a classification, white, yellow, or brown, to each displayed web page, depending on how this page has been called into the browser in the first place. Thus the same displayed https page can be assigned a yellow classification if it is called into the browser one way, and assigned a brown classification if it is called into the browser another way. We adopt the following classification rules.

   (a) Any http page, that is called into the browser, is classified white by the browser.
   (b) Any https page, that is called into the browser using our login protocol TLP, is classified brown by the browser.

(c) Any https page, that is called into the browser using the TLS protocol[2], is either classified yellow if this page is called from within a displayed white or yellow page, or classified brown if this page is called from within a displayed brown page.

When the browser displays a web page, the browser makes its classification of the displayed page clear to the user by choosing an appropriate background color for the URL box. If the displayed page is white (or yellow or brown respectively), then the background color for the URL box is white (or yellow or brown respectively). Note that the current browser already supports white and yellow classifications of web pages. So the main contributions of this project are merely the addition of brown classifications and the introduction of the new login protocol TLP which can be used in calling brown web pages into the browser.

(Recently, a green classification of https web pages has been introduced to distinguish those https pages that have extended validation certificates[4]. Clearly some green pages, like yellow pages, can still be adversarial, and can still be used in launching Phishing and Pharming attacks as described above. Henceforth, when we refer to yellow pages, we do mean yellow or green pages.)

The policy for entering sensitive data (such as usernames, passwords, and credit card numbers) into a displayed web page depends on the classification of the displayed page. This policy consists of the following three rules.

1. **The White Page Rule**:
   A user should never enter sensitive data into a white page.

2. **The Brown Page Rule**:
   A user can enter sensitive data into a brown page.

3. **The Yellow Page Rule**:
   Before a user can enter sensitive data into a yellow page, the user should have prior knowledge that this data can be entered into this particular page, and the user should check that the URL box of the displayed page has indeed the URL of this particular page.

# 4 The Current Login Protocol

Our login protocol TLP, described in the next section, enjoys a number of nice features that are not all present in any of the current login protocols. These nice features are as follows.

1. **Immunity to Attacks**:
   When a user $U$ uses TLP to log in a site $S$, then the login succeeds if and only if both browser B of user U and site S can verify that user U has registered (and stored some login data) in site S sometime in the past.

2. **No External Servers**:
   All the login data, that are needed by user $U$ to use TLP and successfully log in site $S$, are stored on site $S$. Thus TLP does not need any external servers to store some of the login data.

3. **One-Time Login Data**:
   In TLP, the login data, that are needed by user $U$ to log in site $S$, are updated after each successful login of $U$ into $S$. Therefore, if an adversary somehow steals the login data of user $U$ in site $S$, then the stolen data becomes useless after the next login of $U$ into $S$.

4. **Universal Passwords**:
   Each user $U$ needs only to memorize one password $P$, called the TLP universal password of $U$. User $U$ employs his universal password in the TLP protocol to log in every web site. No web site S can deduce the TLP universal password of user $U$ from the login data that user $U$ stores in $S$ or from the messages exchanged between the browser of $U$ and site $S$ during the execution of TLP.

5. **Standard Cryptography**:
   TLP uses only standard symmetric cryptography and standard secure hash functions. Thus, every time the standards of symmetric cryptography or of hash functions are updated, the standards of TLP are updated accordingly.

Next we argue that none of the login protocols, that have been proposed recently, enjoys all these five features.

The current login protocol over the web consists of two protocols: the standard TLS protocol [2] (which is used to authenticate a secure web site by the browser), and a non-standard password protocol (which is sometimes used to authenticate the secure web site by the user and to authenticate the user by the secure web site). As described in Section 2, this login protocol is vulnerable to human mistakes and Phishing and Pharming attacks, and so it does not enjoy feature 1.

This login protocol can be strengthened using Site Keys which allow a user to authenticate the identity of the web site being logged in [12, 1]. Unfortunately, Site Keys can be stolen using Man-In-The-Middle Attacks. Thus the strengthened protocol still does not enjoy feature 1.

The login protocol SRP[14, 13] doest not enjoy any of features 3, 4, and 5 above.

The hash-based protocols, such as [5, 9, 8, 6], enjoy the features 2, 4, and 5. They also allow the web site to verify that the user has registered in the site sometimes in the past. Unfortunately, they do no allow the user's browser to verify that the user has registered in the site sometime in the past. Thus these protocols do not enjoy feature 1. Also, some of these protocols, for example [8], do not enjoy feature 2.

The Passpet system[15] does not enjoy features 2 and 3.

# 5 The New Login Protocol

Our login protocol TLP is to be executed between browser $B$ of user $U$ and web site $S$. Prior to executing TLP, user $U$ needs to have registered with site $S$ by making its browser $B$ store in $S$ the following tuple of four data items:

$$(H(U), \quad n, \quad H(0, n, P, S), \quad H^2(1, n, P, S))$$

where

| | |
|---|---|
| $U$ | is the username of the user, |
| $B$ | is the browser of user $U$, |
| $n$ | is a nonce selected at random by browser $B$, |
| $H$ | is a standard secure hash function, |
| 0 | is the character zero, |
| 1 | is the character one, |
| $P$ | is the TLP universal password of user $U$, and |
| $S$ | is the domain name of the web site. |

Note that $H(0, n, P, S)$ denotes the application of the secure hash function $H$ to the concatenation of the four data items 0, $n$, $P$, and $S$. Also, $H^2(1, n, P, S)$ denotes two consecutive applications of function $H$ to the concatenation of the four data items 1, $n$, $P$, and $S$. After $B$ stores this tuple in $S$, $B$ forgets the tuple completely.

Executing TLP between browser $B$ and site $S$ is intended to achieve five objectives.

1. $B$ checks that $S$ is one of the sites where user $U$ had previously registered (and stored the above tuple of four data items).

2. $S$ checks that user $U$ has entered his universal password $P$ to browser $B$.

3. Both $B$ and $S$ agree on a symmetric session key that they can use to encrypt and decrypt their exchanged messages.

4. $B$ selects a new random nonce $n'$ and stores the following tuple of four data items in $S$ in place of the above tuple:

$$(H(U), \quad n', \quad H(0, n', P, S), \quad H^2(1, n', P, S))$$

   (Therefore, each successful login of browser $B$ into site $S$ causes the tuple of four data items that $B$ had previously stored in $S$ to be replaced by a new tuple of four data items also provided by $B$.)

5. $S$ sends to $B$ the URL of the next https page that $B$ needs to call, using TLS, along with a cookie identifying user $U$ and testifying that the login procedure between $U$'s browser and $S$, has been successful. When the next https page is called into $B$, $B$ assigns this page a brown classification. Moreover browser $B$ assigns any other https page, that is called using TLS from within this brown page, a brown classification .

We adopt the following notation in describing a field in a message that is sent during the execution of TLP.

$$[expression1] < expression2 >$$

This notation means that the value of *expression1* is used as a symmetric key to encrypt the value of *expression2* before the message is sent.

To start executing TLP between $B$ and $S$, user $U$ enters three data items, namely $U$, $P$, and $S$, to a local web page named `httpl` stored in browser $B$. Then the execution of TLP proceeds with the following four message exchanges between $B$ and $S$.

$B \rightarrow S$ : {Hello Message}
$U$

$B \leftarrow S$ : {Hello-Reply Message}
$n, \quad [H(0, n, P, S)] < SN >$

$B \rightarrow S$ : {Login Message}
$U$,
$[H(H^2(1, n, P, S), SN)] < H(1, n, P, S), BN, H^2(1, n', P, S) >$,
$[H(BN, SN)] < n', H(0, n', P, S) >$

$B \leftarrow S$ : {Login-Reply Message}
$[H(BN, SN)] <$ URL of next https web page $>$,
$[H(BN, SN)] <$ cookie $>$

The hello message, from $B$ to $S$, consists of the username of user $U$ who wants to log in site $S$. On receiving this message, $S$ fetches the tuple

$$(H(U), n, H(0, n, P, S), H^2(1, n, P, S))$$

that $B$ had stored previously in $S$. Then $S$ uses the data item $H(0, n, P, S)$ as a symmetric key to encrypt a new nonce $SN$ that $S$ selects at random. The result of the encryption is denoted $[H(0, n, P, S)] < SN >$ and is included in the hello-reply message that is sent from $S$ to $B$.

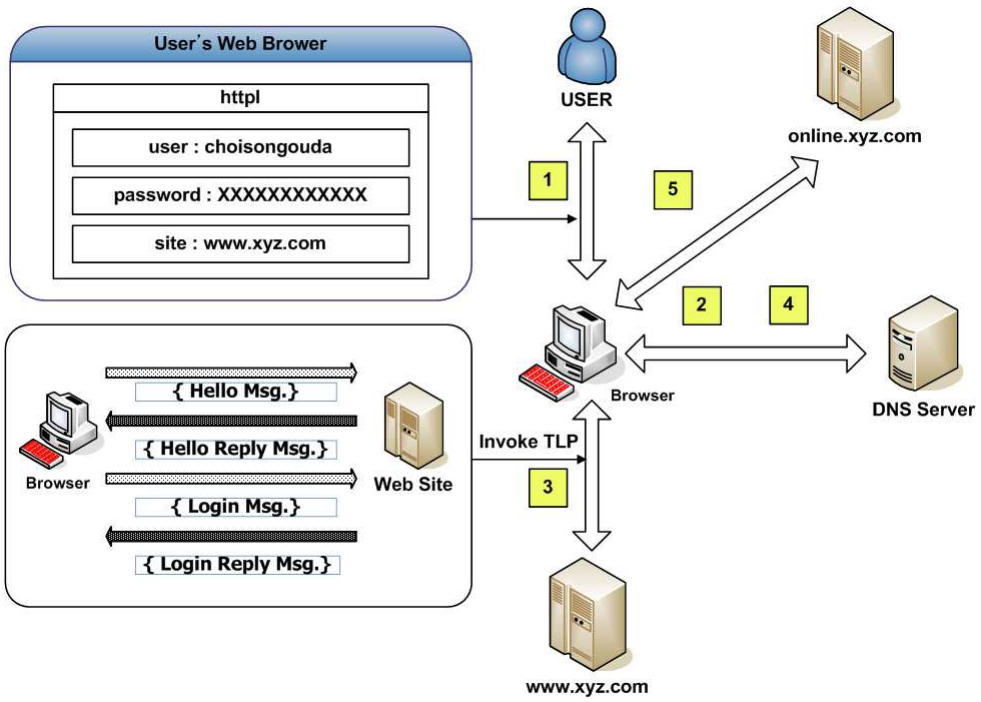


Figure 1: Using TLP

After $B$ receives the hello-reply message, it computes $H(0, n, P, S)$ and uses it to obtain the nonce $SN$ from the received message. Then, $B$ selects at random two nonces $BN$ and $n'$, and uses the received $SN$ and the computed $BN$ and $n'$ to construct the login message before sending it to site $S$.

After $S$ receives the login message, it performs four tasks. First, it checks that user $U$ has indeed entered its TLP universal password $P$ into browser $B$. Second, $S$ extracts the nonce $BN$ from the received message, and now both $B$ and $S$ know $BN$ and $SN$. Third, $S$ stores the tuple:
$(H(U), \quad n', \quad H(0, n', P, S), \quad H^2(1, n', P, S))$ in place of the earlier tuple. Fourth, $S$ constructs the login-reply message and sends it to browser $B$.

After $B$ receives the login-reply message, it concludes that $S$ is one of the web sites where user $U$ has previously registered. Moreover, $B$ gets the URL of the https page that $B$ needs to call next using TLS, along with a cookie that identifies user $U$ and testifies to the fact that the login procedure between $U$'s browser and $S$ has been successful.

Figure 1 illustrates the five steps that are needed for a user to use TLP to log in a web site in a domain say `xyz.com`.

1. The user calls a local web page, for convenience named the `httpl` page, on his browser and enters his username, his TLP universal password, and the site address `www.xyz.com` into this page.
2. The browser uses DNS to get the IP address of site `www.xyz.com`.
3. The browser and site `www.xyz.com` execute TLP. At the end, the browser receives the URL of a web page on site `online.xyz.com` and a cookie.
4. The browser uses DNS to get the IP address of site `online.xyz.com`.
5. The browser and site `online.xyz.com` execute TLS, and the browser gets the required https page at the end. The browser classifies this page brown. It also classifies any other https page, that is called using TLS from within this page, brown.

# 6 Correctness of TLP

An adversary has no chance to succeed in attacking TLP between the browser of user $U$ and web site $S$ unless the adversary acquires a login tuple of $U$ in $S$:

$$(H(U), n, H(0, n, P, S), H^2(1, n, P, S))$$

Moreover, an adversary cannot acquire such a login tuple unless the adversary succeeds in breaking into the secure database(s) where site $S$ stores the login tuples of all its users – a hard task to perform!

Even if an adversary somehow succeeds in acquiring a login tuple of $U$ in site $S$, this adversary cannot succeed in launching a user impersonation attack, a human mistake attack, or a Phishing attack against TLP between the browser of $U$ and $S$.

1. **Security Against User Impersonation:**
   If an adversary acquires a login tuple of user $U$ in site $S$, and if this adversary uses this tuple to impersonate a browser of user $U$ and execute TLP in order to log in site $S$, then the adversary's attempt to log in site $S$ will fail.

2. **Security Against Human Mistakes And Phishing Attacks:**
   If an adversary acquires a login tuple of user $U$ in site $S$, and if this adversary uses this tuple to establish an adversarial web site $S'$ whose domain is different from that of $S$, and if user $U$ requests that his browser executes TLP and logs in site $S'$, then the login attempt will fail.

Still, if it is possible for an adversary to acquire a login tuple of user $U$ in site $S$, then this adversary can launch successful Pharming and eavesdropping attacks against TLP between the browser of $U$ and $S$. To protect against this possibility, the login tuple of $U$ in $S$ is partitioned into two subtuples:

$$(H(U), n, H(0, n, P, S)) \text{ and } (H(U), H^2(1, n, P, S))$$

Each subtuple is stored in a different database. Thus, the first subtuple is stored in database 0 and the second subtuple is stored in database 1. When user $U$ initially registers in site $S$, the browser of $U$ generates the first login tuple and sends it to site $S$. Site $S$ divides the received login tuple into two subtuples and stores the first subtuple in database 0 and stores the second subtuple in database 1.

Later when user $U$ attempts to log in site $S$, site $S$ forwards each of the messages that $S$ receives from $U$'s browser to the appropriate database so that this database can process the message and return a reply to $S$ which forwards the reply back to $U$'s browser. For example, when $S$ receives the Hello($U$) message from $U$'s browser, $S$ selects a nonce $SN$ at random and sends both $U$ and $SN$ to database 0 which prepares a Hello-Reply($n$, $[H(0, n, P, S)] < SN >$) message and returns it to $S$ which forwards it back to $U$'s browser. Thus no one, not even site $S$, gets to keep track of the subtuples stored in the two databases 0 and 1.

Because the two subtuples of user $U$ in site $S$ are continuously changing, it is reasonable then to assume that an adversary, who attempts to acquire the two subtuples of the same user $U$ from the two databases 0 and 1, will do so at different times and will end up with two unsynchronized subtuples of the following form:

$$(H(U), n, H(0, n, P, S)) \text{ and } (H(U), H^2(1, n', P, S))$$

Fortunately, if an adversary who succeeds only in acquiring two unsynchronized subtuples of user $U$ in site $S$, then this adversary cannot succeed in launching a Pharming or eavesdropping attack against TLP between the browser of $U$ and $S$.

3. **Security Against Pharming Attacks:**
   If an adversary acquires two unsynchronized subtuples of user $U$ in site $S$, and if this adversary uses these subtuples to establish an adversarial site $S'$ whose domain is the same as that of site $S$, and if the DNS of $U$'s browser is manipulated so that $U$'s browser is directed to site $S'$ instead of $S$, and if $U$ requests that his browser executes TLP and logs in site $S$, then the login attempt will fail.

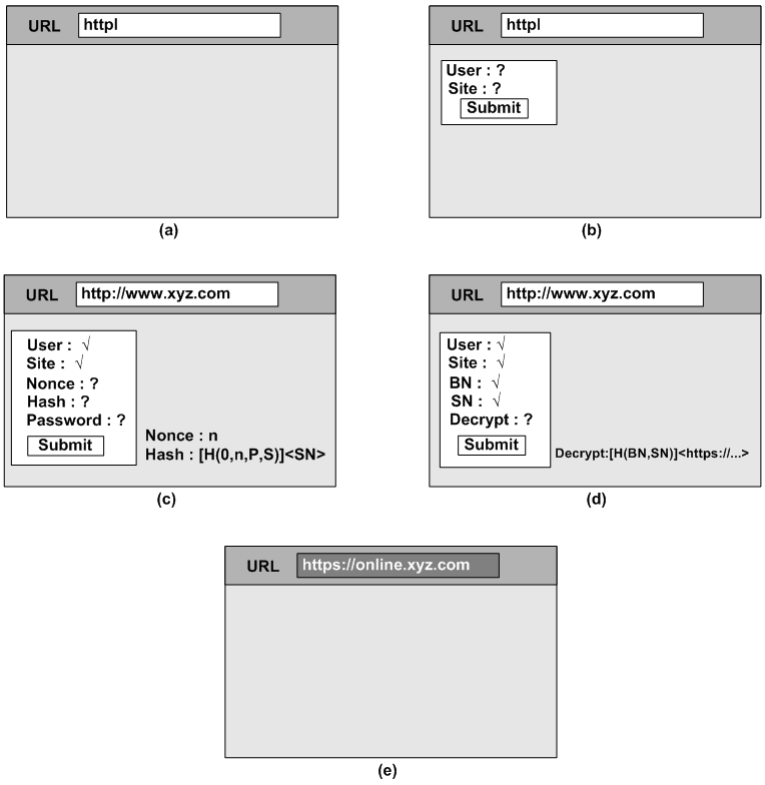


Figure 2: User Interface of TLP

4. **Security Against Eavesdropping:**
   If an adversary acquires two unsynchronized subtuples of user $U$ in site $S$, and if this adversary attempts to eavesdrop on the TLP communication between $U$'s browser and $S$ and obtain the cookie that is sent (encrypted) in the Login-Reply message of TLP, then the eavesdrop attempt will fail.

It is straightforward to prove that TLP satisfies the above four properties 1 through 4. To prove that TLP satisfies each of these properties, it is sufficient to identify a data item that the adversary will need, but will not be able to compute from its acquired data, in order to complete executing TLP successfully.

To prove that TLP satisfies Property 1, note that the adversary will not be able to compute the data item $H(1, n, P, S)$, even though it has acquired a login tuple of user $U$ in site $S$, that is needed to compute the Login Message of TLP.

To prove that TLP satisfies Property 2, note that the adversary will not be able to compute the data item $H(0, n, P, S')$, even though it has acquired a login tuple of user $U$ in site $S$, that is needed to compute the Hello-Reply Message of TLP.

To prove that TLP satisfies Property 3, note that the adversary will not be able to compute the data item $H^2(1, n, P, S)$, even though it has acquired two unsynchronized subtuples of user $U$ in site $S$, that is needed to decrypt the Login Message of TLP and obtain $BN$.

To prove that TLP satisfies Property 4, note that the adversary will not be able to compute the data item $BN$, even though it has acquired two unsynchronized subtuples of user $U$ in site $S$, that is needed to decrypt the cookie in the Login-Reply Message of TLP.

# 7 User Interface of TLP

As a proof of concept, we have developed a prototype of our Transport Login Protocol TLP. The browser side of our prototype is developed on the Firefox browser using the two technologies of Javascript and HTML. The web site side of our prototype is developed on the Tomcat web server using four technologies: Java, HTML, the JSP (Java Server Page) technology, and the MySQL database technology. Note that the MySQL database technology is used to manage the login tuples, of all users, that are stored in the web site.

We employed standard cryptography in our prototype. In particular, we employed the Secure Hash Algorithm SHA-1 for secure hash, and employed the Advanced Encryption Standard AES for symmetric key cryptography.

The guiding principle in our prototype is to ensure that the user never enters his TLP universal password into a web page that is supplied by a web site, but he can enter his password into a local web page that is supplied by his own browser. It turns out that this principle is hard to fulfill in our prototype in the light of the "Same Origin Policy" that is adopted by the Javascript technology. At the end, however, we were able to fulfill this principle by designing a novel user interface for our prototype. We discuss this user interface next.

Figure 2 details the four steps that need to be taken by a user to log in a web site `www.xyz.com`.

1. The user first enters `httpl` into the URL box of his browser and pushes $< return >$ ; see Figure 4a. This causes a display of the local page `httpl` to appear as a small window on the left site of the screen; see Figure 4b.

2. The user enters his username and the name of the site `www.xyz.com` into page `httpl` then clicks on the $< submit >$ button in this page. This causes page `httpl` to execute, update its own display, and send a Hello message (the first message in TLP) to site `www.xyz.com` which replies by sending back the web page `http://www.xyz.com`. This page contains the two fields, named nonce and hash, of the Hello-Reply message (the second message in TLP); see Figure 4c.

3. The user copies the values of the two fields nonce and hash from the displayed page `http://www.xyz.com` and enters them into page `httpl`. The user then enters his password into page `httpl` and clicks on the $< submit >$ button of this page. This causes page `httpl` to execute, update its own display, and send a Login message (the third message in TLP) to site `www.xyz.com` which replies by sending back a new web page `http://www.xyz.com`. This new page contains one field, named decrypt, of the Login-Reply message (the last message in TLP); see Figure 4d.

4. The user copies the value of field decrypt from the displayed page `http://www.xyz.com` and enters it into page `httpl`. The user then clicks on the $< submit >$ button of page `httpl`. This causes page `httpl` to execute, compute the next https page, say page `https://online.xyz.com`, that needs to be called into the browser, and redirects the browser to call this page using TLS and display it on the screen. Note that in this case the browser assigns the displayed https page a brown classification, and so the background color of the URL box of the displayed page becomes brown as shown in Figure 2e.

Because the browser has classified the displayed page `https://online.xyz.com` brown, then if the user clicks on any link (of an https page) in page `https://online.xyz.com`, then the browser will classify the newly called page brown as well.

# 8 Concluding Remarks

In this paper, we present a comprehensive proposal to counter human mistakes and Phishing and Pharming attacks that may occur when a user attempts to log in a secure web site. Our proposal is based on two ideas. First, we introduce a new classification, brown, of secure https web pages. When the browser of a user $U$ classifies a displayed page brown, user $U$ should conclude that the displayed page is secure and can enter his sensitive data into it. Second, we design a new login protocol, named TLP, that is secure against human

mistakes and Phishing and Pharming attacks. The browser of a user $U$ uses TLP to classify a displayed page brown according to two rules:

1. The displayed page is called into the browser using TLP.
2. The displayed page is called into the browser, using TLS, from within another brown page that was displayed earlier on the browser.

Note that TLP is not intended to replace TLS. On the contrary, our vision assigns complementary roles to be played by TLP and TLS: TLP can be used first to securely log in a web domain, then TLS can be used later to securely go from one web site to another within the logged in domain.

Note also that some mildly secure web domains may feel that they are in no danger of facing Phishing or Pharming attacks because adversaries have little incentive to launch such attacks against these domains. (Examples of such domains are those that host electronic reviewing and handling of submitted papers to conferences and journals.) These web domains can keep on employing TLS, as they do presently, both for logging in a domain and for going from one web site to another within this domain.

A nice feature of TLP is that a user can use the same username and same (TLP universal) password to securely log in any web site in the Internet. This means that the user needs only to memorize one username and one password for all web sites. Therefore it is reasonable to demand that each user chooses a long string, say of sixteen characters, to be his TLP universal password. And so TLP becomes naturally secure against online and offline dictionary attacks.

### Acknowledgement

The work of Mohamed G. Gouda was supported in part by the US National Science Foundation under grant CNS-0520250.

## References

[1] Bank of America. `http://www.bankofamerica.com/privacy/sitekey/`.

[2] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Transport Layer Security (TLS) Extensions. RFC 4366 (Proposed Standard), Apr. 2006.

[3] R. Dhamija, J. Tygar, and M. Hearst. Why phishing works. In *the Proceedings of the Conference on Human Factors in Computing Systems (CHI2006)*, 2006.

[4] R. Franco. Website identification and extended validation certificates in IE7 and other browsers. IEBlog, Nov 2005.

[5] E. Gabber, P. B. Gibbons, Y. Matias, and A. Mayer. A convenient method for securely managing passwords. In *Financial Cryptography*, Feburuary 1997.

[6] M. G. Gouda, A. X. Liu, L. M. Leung, and M. A. Alam. Spp: An anti-phishing single password protocol. *Comput. Netw.*, 51(13):3715–3726, 2007.

[7] A.-P. W. Group. Phising activity trends report. `http://www.antiphishing.org/reports/apwg_report_sept_2007.pdf`, Sept 2007.

[8] J. A. Halderman, B. Waters, and E. W. Felten. A convenient method for securely managing passwords. In *14th International World Wide Web Conference*, May 2005.

[9] J. Kelsey, B. Schneier, C. Hall, and D. Wagner. Secure applications of low-entropy keys. In *ISW '97: Proceedings of the First International Workshop on Information Security*, pages 121–134, London, UK, 1998. Springer-Verlag.

[10] R. McMillan. Gartner: Consumers to lose $2.8 billion to phishers in 2006. http://www.networkworld.com/news/2006/110906-gartner-consumers-to-lose-28b.html, 2006.

[11] G. Ollmann. The pharming guide. http://www.ngssoftware.com/papers/ThePharmingGuide.pdf.

[12] P. Security. http://www.passmarksecurity.com.

[13] T. Wu. Srp-6: Improvements and refinements to the secure remote password protocol. Submission to the IEEE P1363 Working Group.

[14] T. Wu. The secure remote password protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, 1998.

[15] K.-P. Yee and K. Sitaker. Passpet: convenient password management and phishing protection. In *SOUPS '06: Proceedings of the second symposium on Usable privacy and security*, pages 32–43, New York, NY, USA, 2006. ACM.