TPP: The Two-Way Password Protocol

Taehwan Choi
Department of Computer Science
The University of Texas at Austin
Email: ctlight@cs.utexas.edu

H. B. Acharya
Department of Computer Science
The University of Texas at Austin
Email: acharya@cs.utexas.edu

Mohamed G. Gouda
National Science Foundation
The University of Texas at Austin
Email: mgouda@nsf.gov

***Abstract*—The need for secure communication in the Internet has led to the widespread deployment of secure application-level protocols. The current state-of-the-art is to use TLS, in conjunction with a password protocol. The password protocol, which we call a one-way password protocol (OPP), authenticates a user to a server, using a particular secret called the password. TLS has two functions: (1) It ensures secure communication between a client and a server (2) It allows a user to authenticate a server. The first function effectively provides a secure channel for end-to-end communication between a client and a server. However, the second function is frequently compromised by a variety of Phishing attacks. In this paper, we address this problem by developing a password protocol which we name the Two-way Password Protocol (TPP). TPP, when used in conjunction with TLS, ensures that users correctly authenticate servers, and are protected from Phishing attacks. The first contribution of this paper is to develop a protocol, called the Universal Password Protocol (UPP), which ensures that a user's password is kept safe even in the case of a successful Phishing attack. However, it may be noted that a user, after logging in, frequently shares other secrets (such as credit card number) over the secure connection, and UPP cannot protect these. Our second contribution is to build on UPP and develop, first, the Two-Way Password Protocol (TPP), and finally an improved version named the Dynamic Two-Way Password Protocol (DTPP), which ensures that both a server and a client are properly authenticated to each other. This ensures the security of all secrets which should be known only to the client and the server, including, of course, the password.**

I. Introduction

A problem of immense importance in any scenario where users, with different privileges, use a system, is authentication. Clearly, it is necessary for a party, that controls access to a resource, to verify the identity of the party requesting access; otherwise, there is no way to determine which requests to allow, and which to deny. Thus, authentication is essential whenever an entity provides specific other parties with access to special resources – for example, by sharing secrets with them. A major application of this is seen in the use of secure remote protocols in the Internet, where the resources provided range from email to auctions to banking. In general, authentication on the Internet involves two parties: a user and a server.

The current state-of-the-art in providing online authentication involves a combination of two protocols: the Transport Layer Security protocol, or TLS [1], and a simple password protocol, which we have named the One-Way Password Protocol OPP. TLS certifies to the user that the server is not *spoofing*, i.e. presenting a false address; it also ensures that a user and a server have a cryptographically secure channel of communication. OPP authenticates the user to the server.

Unfortunately, this combination is not enough to ensure security against some classes of attack. For example, by simple human error, or by use of a Phishing attack, a user U may end up going to a malicious server M (say eebay.com) instead of the actual server S she has a relationship with (in this case, ebay.com). TLS does not protest – the site is not spoofing; the address bar indeed says eebay.com, which is correct. OPP is useless here; it authenticates a user to a server, not a server to a user. Thus, the adversary M can harvest secrets from U – most importantly the password authenticating U to S, but also other secrets she might share with S, such as credit card number, home address, and so on.

Earlier work in the area of providing security against such attacks, such as the TLP protocol, have focused how to strengthen TLS so that it can defend against these attacks. In the system developed by Choi et al. [2], TLP must be used for the first login into a secure page; TLS can be used to authenticate pages subsequently reached from a secure page.

In this paper, we study the problem of ensuring secure authentication that is robust against Phishing attacks (and related problems such as user error). We achieve this goal using the standard TLS protocol, simply by modifying the password protocol so it achieves two-way authentication between a server and a user. Our solution is developed step by step. We start with OPP and modify it to the server password protocol (passwords for a user on a server and for a server on a user). This simple protocol can be defeated by using a Phishing attack in conjunction with a Man-In-The-Middle attack. Hence, we modify it and develop the Universal Password Protocol UPP, which is adequate for the purpose of protecting passwords. (It may be noted in passing that UPP also solves the problem of password reuse – the user only has to remember one password for access to all secure sites, without the problem of reusing a password at multiple servers.) However, we note that even UPP can be beaten: a malicious server can simply log the user in, and steal other secrets from her, when it fails to obtain her password. In answer to this problem, we develop the Two-Way Password Protocol TPP and improve it to our final version, the Dynamic Two-Way Password Protocol DTPP. DTPP ensures that all the secrets shared between a user and a server – and in particular the user's password on the server – are secure from Phishing attacks.

We begin with a detailed description of TLS and the current state of the art, in the following section.

II. Background

Before a user U can communicate with a secure website S over the web, both U and S need to authenticate one another by executing two protocols: the standard TLS protocol [1] which allows U to authenticate S, and a usually one-way password protocol which allows S to authenticate U. For completeness, in this section we briefly review the standard TLS protocol.

A certificate of a website S is a data structure that has the following format:

$$(R, S, K_S, t, sig)$$

where R is an issuer, S is a website, K_S is a public key of S, t is an expiration date, sig is a signature.

This certificate can be viewed as the statement "R asserts that the public key of website S is K_S, from now until date t." When a user U receives this certificate, U needs to perform two checks. First, U needs to check that the certificate is current by checking that the expiration date t has not yet been reached. Second, U needs to check that the certificate is valid by using the signature of the certificate, as discussed below, to validate that R has indeed issued the certificate. If C concludes that the certificate is both current and valid, then C accepts that K_S is the public key of S.

The signature of the certificate is computed by the issuer R as follows:

$$sig := K_R^{-1} < H(R, S, K_S, t) >$$

where K_R^{-1} is the private key of R, H is a standard secure hash function, $H(R, S, K_S, t)$ is the result of applying the hash function H to the concatenation of the four items R, S, K_S, and t in the certificate, and $K_R^{-1} < H(R, S, K_S, t) >$ is the encryption of $H(R, S, K_S, t)$ using the private key K_R^{-1} of issuer R.

When a client C receives this certificate, C needs to use the signature of the certificate to validate that R issued the certificate. To perform this check, client C needs to know a priori the public key K_R of issuer R. (Public keys from trusted Certification Authorities, for example "Verisign", come pre-loaded with all browsers in use today.) User C performs this check as follows:

1) Client C decrypts the signature of the received certificate using the public key K_R of issuer R.
2) Client C applies the secure hash function H to (the concatenation of) the four fields R, S, K_S, and t in the certificate.
3) If the values computed in the two previous steps are equal, then user C concludes that R has indeed issued the received certificate. Otherwise, client C concludes that R has not issued the certificate.

We will now present the execution of a simple scenario of the TLS protocol. A client C and a server S are executing the protocol, using the certificate of S.

$$\begin{array}{ll} C \rightarrow S: & client\text{-}hello(nc) \\ C \leftarrow S: & server\text{-}hello(ns), \\ & certificate((R, S, K_S, t, sig)) \\ C \rightarrow S: & key\text{-}exchange(K_S < pms >), \\ & finished(H(nc, ns, pms, ms)) \\ C \leftarrow S: & finished(H(ns, nc, pms, ms)) \end{array}$$

where nc is a nonce selected at random by client C and sent in the clear to website S, ns is a nonce selected at random by website S and sent in the clear to client C, pms is a premaster secret selected at random by client C and sent in private to website S after it is encrypted by the public key K_S of S, and ms is the master secret computed by both client C and website S using the three values nc, ns, and pms.

The $key\text{-}exchange(K_S < pms >)$ message, sent from C to S in the third step of this scenario, is intended to challenge S, if it is indeed S, to use its private key K_S^{-1} to decrypt the message, obtain the premaster secret pms, and use pms to compute the master secret ms. When client C checks, in the fourth step, that S was able to correctly compute ms, C knows that it is indeed communicating with S.

The $finished(H(nc, ns, pms, ms))$ message, sent from C to S in the third step of this scenario, is intended to assure website S that client C was able to compute the master secret ms correctly. Similarly, the $finished(H(ns, nc, pms, ms))$ message, sent from S to C in the fourth step of this scenario, is intended to assure client C that website S was able to compute ms correctly.

At the end of this execution, the following two outcomes are achieved.

1) Client C knows that it is indeed communicating with website S.
2) Both C and S agree on a master secret ms that they can use to encrypt and decrypt all the messages that they need to exchange next.

Note that the authentication is not symmetric: server S does not know the client with whom it is communicating. In order to make up for this shortcoming, TLS is usually paired with a simple password protocol to authenticate client C to server S.

Unfortunately, this arrangement is not adequate to provide security. In the next section, we show that despite the security provided by the TLS protocol, there exist attacks that can circumvent the security provided by a combination of TLS with normal password authentication.

III. The One-Way Password Protocol

In this section, we argue that the authentication procedure that is based on the standard TLS protocol and the traditional one-way password protocol are vulnerable to Phishing attacks.

After executing the TLS protocol as in section II, and in order for S to know user U with whom it is communicating, U and S execute the following two steps of the one-way password protocol. (Note that the messages exchanged in these two steps are encrypted using the master secret ms that is computed in the TLS protocol.)

$U \leftarrow S$: $ms <$ enter (user id, password) $>$
$U \rightarrow S$: $ms < (U, pw) >$

Prior to executing these two steps, user U has registered the pair $(U, H(pw))$ in website S where pw is a password of U. Thus, when S receives the message $ms < (U, pw) >$ from U, S concludes that it is indeed communicating with user U.

Clearly, OPP ensures that the user is authenticated to the server. TLS ensures that the server is authenticated to the user's browser; when the URL for site S is displayed in the location bar of the browser, the user is indeed at site S.

However, the combination of OPP and TLS has one subtle weakness. Authenticating the server to the user's browser is not the same as authenticating the server to the user. In fact, an adversary M can defeat this security measure simply by not spoofing (i.e. M does not claim to be at the URL of server S) and using other means to make the user U associate M with S.

We will now describe an example of a Phishing attack that can defeat the two authentication protocols, the TLS protocol and the one-way password protocol, discussed above.

Simple Phishing Attack:

A user U receives the following email:

> "For being a good customer of the website https://www.ebay.com, we offer you a special deal. Please log into the website https://www.specialdeals.com to check out our great deal to you."

From now on, we refer to the website https://www.ebay.com as website S, and we refer to the website https://www.specialdeals.com as website M.

Excited by this email, user U proceeds to log into website M. First, the TLS protocol is executed between client C of user U and website M so that U can be certain that it is indeed communicating with M. Second, M sends to U the message $ms <$ enter (user id, password) $>$, but this message is displayed on a webpage that has the same logo and graphics as that of website S. Third, user U enters into the displayed webpage his user id U and his password pw, which U has registered earlier in website S. Fourth, user U receives from website M a webpage that offers U to purchase a good collection of DVDs for a cheap price and instructs U to enter his credit card number if he is interested in purchasing this collection. Fifth, user U decides to purchase the DVD collection and enters his credit card number into the webpage.

Unfortunately for user U and website S, website M is not related in any way to website S. (The fact that sites S and M belong to different domains, as site S belongs to the domain ebay.com and site M belongs to the domain specialdeals.com, should have implied that these two sites are not related.) In fact, M is an adversarial website that has just launched a successful Phishing attack against user U and website S and obtained the pair (U, pw), which user U has registered earlier in website M, along with the credit card number of user U.

After obtaining this information, website M can launch two more attacks against user U and website S.

1) A User Impersonation Attack:
   Using the pair (U, pw), M can successfully log into website S pretending to be user U.
2) An Identity Theft Attack:
   Using the credit card number of U, M can purchase many items over the web.

This Phishing attack is successful because neither the TLS protocol nor the one-way password protocol attempted to authenticate the fact that websites S and M are related to one another.

IV. The Server Password Protocol

In the previous section, we see clearly that the standard practice (of using OPP and TLS) can be broken by Phishing attacks. However, we note that the use of OPP in conjunction with TLS does in fact ensure that the following two conditions hold:

1) The server is in fact the server whose address is currently displayed in the user's address bar.
2) The user is authenticated to the server.

The problem is that the server is not properly authenticated to the user; user U can be sure that it is indeed communicating with server M, but has no way of knowing whether M is in fact associated with server S with which U has a relationship of trust.

We note that the user is properly authenticated to the server. This asymmetry is caused by the fact that OPP checks to make sure that the user has the correct password to log on to the server, but there is no corresponding check for the server.

Based on the above observation, it is natural to ask whether simply making the password protocol more symmetric would solve the problem. Just as the user is authenticated to the server by knowledge of a password, the server is authenticated to the user by knowledge of a secret called the *server password*. (For example, a server password can be a unique image, a phrase etc.) The user stores the server password on server S. In subsequent interaction, S authenticates itself to U by sending U its server password.

We name this protocol the server password protocol, and show its working below.

User U stores in website S the triplet:

$$(U, ps, H(pw))$$

where U is a user id, ps is a server password, and pw is a password of user U.

$U \leftrightarrow S$: execute TLS and compute ms
$U \leftarrow S$: $ms <$ enter user id $>$
$U \rightarrow S$: $ms < U >$
$U \leftarrow S$: $ms < ps,$ enter password $>$
$U \rightarrow S$: $ms < pw >$

Unfortunately, the attractive hypothesis, that this protocol is robust against Phishing attacks, is incorrect. We demonstrate that a Phishing attack, combined with a Man-In-The-Middle attack, succeeds in compromising the site-key password protocol.

Phisherman in the Middle Attack:

1.	$U \leftrightarrow$	M	: execute TLS and compute ms
2.		$M \leftrightarrow S$	: execute TLS and compute ms'
3.		$M \leftarrow S$	: $ms' <$ enter user id $>$
4.	$U \leftarrow$	M	: $ms <$ enter user id $>$
5.	$U \rightarrow$	M	: $ms < U >$
6.		$M \rightarrow S$	: $ms' < U >$
7.		$M \leftarrow S$	: $ms' < ps,$ enter password $>$
8.	$U \leftarrow$	M	: $ms < ps,$ enter password $>$
9.	$U \rightarrow$	M	: $ms < pw >$
10.		$M \rightarrow S$	: abort login procedure
11.	$U \leftarrow$	M	: $ms <$ enter credit card # $>$
12.	$U \rightarrow$	M	: $ms < cc >$
13.		M	: gets both pw and cc

The reason for the insecurity of this protocol, is that both authentications (a user to a server and a server to a user) do not happen simultaneously. At some step, one party, the user or the server, has to take a "leap of faith" and go first, sending its secret (password or server password, respectively) to the other party before it is authenticated. In this case, it is the server that sends its server password to the user before it has seen the user password; consequently, the adversary M can obtain the server password from S and break the protocol, as shown above.

V. The Universal Password Protocol

In this section, we present a protocol, called the Universal Password Protocol, which ensures that the password of U on S cannot be stolen by Phishing attacks. The primary idea is that the user only has to remember one universal password; the password for a server is generated when needed.

User U stores in website S the triplet:

$$(U, ps, H(pw))$$

where U is a user id, ps is a server password, and the password pw is computed as follows.

$$pw := H(upw, ds)$$

where H is a standard secure hash function, upw is the *universal password* of user U, ds is the domain name of web site S (for example if S is the website https://www.amazon.com, then ds is amazon.com), and $H(upw, ds)$ is the application of function H to the concatenation of the universal password upw of user U, and the domain name ds of server S.

The execution of the universal password protocol proceeds as follows:

$U \leftrightarrow S$	: execute TLS and compute ms
$U \leftarrow S$	: $ms <$ enter user id $>$
$U \rightarrow S$	: $ms < U >$
$U \leftarrow S$	: $ms < ps,$ enter password $>$
$U \rightarrow S$	: $ms < H(upw, ds) >$

It may be noted that the user sends the server its password to authenticate itself, but the server stores only the secure hash of the password. The reason for this measure is to ensure that, even if the server is compromised – for example, by disgruntled employees – and the store of hashed passwords is stolen, the attacker cannot start using this database of stolen passwords to impersonate U.

We can now make a very interesting observation. As TLS prevents spoofing, U knows the domain name of M, and will use dm rather than ds to compute the password sent to server M; consequently, M cannot learn the password of U on S by means of Phishing attacks.

However, this protocol, while perfectly adequate for the purpose of protecting the password of U, does not serve to protect any other secrets shared between U and S.

To see why, we consider the following subtle Phishing attack.

The Persevering Phisherman Attack:

1.	$U \leftrightarrow$	M	: execute TLS and compute ms
2.		$M \leftrightarrow S$	: execute TLS and compute ms'
3.		$M \leftarrow S$	: $ms' <$ enter user id $>$
4.	$U \leftarrow$	M	: $ms <$ enter user id $>$
5.	$U \rightarrow$	M	: $ms < C >$
6.		$M \rightarrow S$	: $ms' < C >$
7.		$M \leftarrow S$	: $ms' < skc,$ enter password $>$
8.	$U \leftarrow$	M	: $ms < skc,$ enter password $>$
9.	$U \rightarrow$	M	: $ms < H(upw, dm) >$
10.		$M \rightarrow S$	: abort login procedure
11.	$U \leftarrow$	M	: $ms <$ enter credit card # $>$
12.	$U \rightarrow$	M	: $ms < cc >$
13.		M	: gets cc

On close observation, we see that the weakness in UPP is again due to the problem that authentication of U to S and of S to U does not happen in a single step, and consequently, there remains a possibility that the server password of S can be stolen.

However, we have seen in UPP that, by incorporating the domain name ds of server S into the password, we can protect the password from being stolen by the adversary: M can get a password, but it will not be the password of U on S. It is natural to ask if the same idea, using ds in the server password of S, can protect the server password from being stolen and used by M. We develop this idea in the following section.

VI. The Two-Way Password Protocol

In the previous section, we demonstrated that while UPP is adequate for protecting passwords, the security of a system using UPP can still be compromised by an adversary using a more subtle attack, which we call the Persevering Phishing attack. A secure session involves not only the password, but also other secrets; even if the adversary M cannot acquire the password, it can acquire the other secrets shared between U and S – such as the credit card number of U.

In this section, we address this vulnerability to develop a more advanced version of our protocol, which we call the Two-Way Password Protocol (TPP). This protocol ensures that, unlike in UPP, the malicious server M cannot simply pass on to U a server password from S. Thus, TPP (in conjunction

with TLS to prevent spoofing) protects not only the password, but also any other secrets shared between U and S.

The fundamental insight behind this protocol is the fact that the hash of the user's password $H(pw)$, stored on the server, can in fact itself be used as a server password. In our exposition on UPP, we showed how password pw can be made site-specific by incorporating the server domain name ds. By using $H(pw)$ as the server password of S, we make the server password domain-specific also. To authenticate itself to U, the adversary server M needs to produce the corresponding server password $H^2(upw, dm)$. But M cannot obtain this server password: its value is not present on server S (because S stores $H^2(upw, ds)$, not $H^2(upw, dm)$) and M cannot calculate it without knowledge of upw (which U does not share). Hence we solve the problem of the server M stealing the server password of S and authenticating itself to the user.

User U stores in website S the pair:

$$(U, H(pw))$$

where U is a user id, and pw is computed as discussed in the Universal password protocol in the previous section.

$$pw := H(upw, ds)$$

$U \leftrightarrow S$: execute TLS and compute ms
$U \leftarrow S$: $ms <$ enter user id $>$
$U \rightarrow S$: $ms < U >$
$U \leftarrow S$: $ms < H^2(upw, ds),$ enter password $>$
$U \rightarrow S$: $ms < H(upw, ds) >$

The Failure of Phishing Attack:

1.	$U \leftrightarrow$	M	: execute TLS and compute ms
2.		$M \leftrightarrow S$	: execute TLS and compute ms'
3.		$M \leftarrow S$	: $ms' <$ enter user id $>$
4.	$U \leftarrow$	M	: $ms <$ enter user id $>$
5.	$U \rightarrow$	M	: $ms < U >$
6.		$M \rightarrow S$	: $ms' < U >$
7.		$M \leftarrow S$	: $ms' < H^2(upw, ds)$, enter password $>$
8.	$U \leftarrow$	M	: The attack fails at this point since M cannot compute $ms < H^2(upw, dm) >$ to send it to U

VII. The Dynamic Two-Way Password Protocol

In the previous section, we demonstrated the Two-way password protocol, which is secure against even sophisticated Phishing attacks. For all practical purposes, the Two-way password protocol achieves our goal of being immune to Phishing attacks.

However, unlike some advanced login protocols such as TLP [2], the Two-way password protocol does not have the feature of one-time login data. In TLP, the login data needed to authenticate U to S is updated on each login, so even if an adversary manages to acquire the login data of U to S, the stolen data becomes useless after the next login of U into S.

Under the assumption of secure TLS, the password is secure in TPP – it is only sent to the authenticated server S, and is encrypted to make sure that it cannot be stolen by an eavesdropping attack. However, in practice, there do exist attacks that break the security assumptions of TLS; for example, the root certificate authorities accepted by browsers are not always trustworthy. A password protocol cannot protect against such an attack (where TLS is broken and the attacker M can spoof as S). However, in order to minimize the damage if even such an attack is carried out, we incorporate into the Two-way protocol the additional feature of one-time login data. The secrets needed to authenticate U to S, and S to U, are updated on each login; hence, even if the adversary does manage to steal the password of U, the stolen password is only useful until the next time U logs into S. We call this final version of our protocol the two-way password protocol, TPP.

User U stores in website S the triplet:

$$(U, n_i, H(pw_i))$$

where U is a user id, n_i is (the i'th value of) a nonce chosen by the user, and pw_i is

$$H(upw, n_i, ds)$$

where upw is the universal password of user U and ds is the domain name of server S.

$U \leftrightarrow S$: execute TLS and compute ms
$U \leftarrow S$: $ms <$ enter user id $>$
$U \rightarrow S$: $ms < U >$
$U \leftarrow S$: $ms < n_i, H^2(upw, n_i, ds),$ enter password $>$
$U \rightarrow S$: $ms < H(upw, n_i, ds), n_{i+1}, H^2(upw, n_{i+1}, ds) >$

We see that the working of the Two-way password protocol is almost exactly similar to that of the Two-way password protocol. The main difference is that the password is not some fixed pw, but pw_i: it varies with each login.

Server S stores the last value of the nonce n_i and the corresponding $H(pw_i)$. When user U tries to log in, she is given her n_i as well as the corresponding $H(pw_i)$. As U knows $H, upw, n_i,$ and ds, she can check that $H(pw_i)$ is correct, and authenticate the server. Now she chooses the next value of the nonce to be n_{i+1}. In the last step, she sends to the server the password $pw_i = H(upw, n_i, ds)$ (so user is authenticated to server), and the pair n_{i+1} and $H^2(upw, n_{i+1}, ds)$, i.e. $H(pw_{i+1})$. S replaces the stored n_i and $H(pw_i)$ with n_{i+1} and $H(pw_{i+1})$; these values will be used the next time a user tries to log in with user name U. Thus, the password and server password change with every use in this protocol.

VIII. Related Work

Secure remote authentication of parties over the Internet is an extremely important problem, and has been the focus of considerable research. In this section, we discuss a few relevant protocols, and specify the contribution of this paper in the context of earlier work.

As TPP is a password protocol, it is most natural to consider it in the context of earlier password protocols. From the development of the protocol, it is clear that the most interesting feature of TPP is its immunity to Phishing attacks, which break ordinary password protocols, simple challenge-handshake authentication protocols such as site key [3] and message digest protocols [4], and hash-based protocols [5].

However, TPP, thanks to its use of a universal password, has several other highly desirable features. Early password protocols such as Lamport's [6] and Rubin's [7] one-time password protocols are forced to use a list of passwords, which the client uses one time only, to guard against the threat of eavesdropping attacks. This, of course, leads to the serious inconvenience of having to remember and register a huge list of passwords. On the other hand, protocols that depend on one central server to authenticate clients for multiple servers [8] have a single point of failure and require a high cost of integration. TPP circumvents all of these problems; it ensures that login data is for one-time use only, but requires the user to remember only one (strong) password, and, unlike the Passpet system [9], does not require any external server for authentication.

Another important feature of TPP is that it uses no exotic computation such as modular exponentiations etc; the only required computation, (in addition to the standard encryption/decryption done by TLS) is the computation of one secure hash at the client and one at the server. Thus, it does not use any non-standard operations or require much processing power, unlike other strong password protocols such as EKE [10] and SRP [11]. Moreover, as it is built to be used in conjunction with TLS, it benefits directly from improvements to TLS. For example, TLS is currently being upgraded to use SRP as an underlying layer. This will improve the security of TLS, and thus strengthen a system running TPP, as any such system also runs TLS.

The closest ancestor to TPP is our own earlier protocol SPP [12]. However, SPP is a single password protocol, whose aim is simply to safeguard the user's single (universal) password. Thus, SPP is vulnerable to subtle attacks that try to steal other secrets besides the user's password (such as the Persevering Phisherman attack). TPP provides complete protection of all secrets from Phishing attacks.

IX. Concluding Remarks

Standard authentication over the web, using TLS and a password protocol, is easily compromised by user error and Phishing attacks. In this paper, we present a strong protocol, UPP, which ensures that the user's password cannot be compromised. Next, building on UPP, we develop TPP and finally DTPP, a password protocol which (in conjunction with TLS) provides mutual authentication between client and server, and protects all shared secrets between client and server from Phishing attacks. It may be noted that this protocol is fairly lightweight; it takes only four messages (the original password protocol itself takes two), and imposes little additional computational or storage load on the client or on the server. We suggest that, given the widespread prevalence of Phishing attacks [13], there is good reason to deploy the TPP protocol and replace the one-way passwords that are used on the Internet today.

References

[1] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246 (Proposed Standard), Internet Engineering Task Force, Aug. 2008, updated by RFCs 5746, 5878, 6176. [Online]. Available: http://www.ietf.org/rfc/rfc5246.txt

[2] T. Choi, S. Son, M. G. Gouda, and J. A. Cobb, "Pharewell to phishing," in *Proceedings of the 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, ser. SSS '08, 2008, pp. 233–245.

[3] J. Youll. (2006) Fraud vulnerabilities in sitekey security at bank of america. [Online]. Available: www.cr-labs.com/publications/SiteKey-20060718.pdf

[4] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication," RFC 2617 (Draft Standard), Internet Engineering Task Force, Jun. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2617.txt

[5] J. A. Halderman, B. Waters, and E. W. Felten, "A convenient method for securely managing passwords," in *Proceedings of the 14th international conference on World Wide Web*, ser. WWW '05, 2005, pp. 471–479.

[6] L. Lamport, "Password authentication with insecure communication," *Communications of the ACM*, vol. 24, pp. 770–772, November 1981.

[7] A. D. Rubin, "Independent one-time passwords," in *Proceedings of the 5th conference on USENIX UNIX Security Symposium - Volume 5*, 1995, pp. 15–15.

[8] Microsoft. (2010) .net passport. [Online]. Available: http://www.passport.net

[9] K.-P. Yee and K. Sitaker, "Passpet: convenient password management and phishing protection," in *Proceedings of the second symposium on Usable privacy and security*, ser. SOUPS '06, 2006, pp. 32–43.

[10] S. M. Bellovin and M. Merritt, "Encrypted key exchange: Password-based protocols secure against dictionary attacks," in *IEEE SYMPOSIUM ON RESEARCH IN SECURITY AND PRIVACY*, 1992, pp. 72–84.

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

[12] M. G. Gouda, A. X. Liu, L. M. Leung, and M. A. Alam, "Spp: An anti-phishing single password protocol," *Computer Networks*, vol. 51, pp. 3715–3726, September 2007.

[13] Trusteer. (2009) Measuring the effectiveness of in-the-wild phishing attacks. [Online]. Available: http://www.trusteer.com/sites/default/files/Phishing-Statistics-Dec-2009-FIN.pdf