PAKE-Based Web Authentication: the Good, the Bad, and the Hurdles

John Engler
UC Berkeley

Chris Karlof
UC Berkeley

Elaine Shi
PARC

Dawn Song*
UC Berkeley

Abstract. Password Authenticated Key Exchange (PAKE) is a class of cryptographic protocols that allow two parties sharing a password to authenticate each other without explicitly revealing the password in the process. PAKE protocols offer a potential improvement over current web authentication practices, e.g., HTML form-based password authentication, but there has been little progress towards integrating PAKE into web browsers and servers. In this paper, we report the results of a systematic investigation of various practical issues and challenges in deploying PAKE for web authentication. We examine three categories of issues: 1) security issues related to UI design; 2) security issues related to the browser's same origin policy; and 3) potential hurdles to deployment. We propose potential solutions for some problems and identify areas for future work.

1 Introduction

The most common web authentication technique in use today is password authentication via an HTML form, where a user types her password directly into a web page from the site to which she wishes to authenticate herself. The problem with this approach is that it relies on the user to determine when it is safe to enter her password. To resist phishing and other social engineering attacks, a user must rely on the browser's security indicators and warning messages, e.g., the URL bar and the site's SSL certificate, to authenticate the website and determine when it is safe to enter her password. Unfortunately, studies suggest that many users habitually click through SSL certificate warnings due to the pervasiveness of certificate errors [14, 48]. Other studies show that users do not understand browser indicators [11, 20, 21, 46].

To address these vulnerabilities, we revisit the idea of applying Password Authenticated Key Exchange (PAKE) [25, 38, 39, 3, 9, 4, 40, 35, 33, 23, 22, 24, 23, 22, 41] protocols to web authentication. A PAKE protocol is a cryptographic protocol that allows two parties who share knowledge of a password to mutually authenticate each other and establish a shared key, without explicitly revealing the password in the process.

One hope of using PAKE protocols for web authentication is to help make it easier for users to authenticate websites and reduce the attack surface of social engineering attacks against their accounts. With the current approach of HTML form-based authentication, it is unsafe for a user to attempt to authenticate herself to a phisher mimicking a website she trusts, since doing so will reveal her password. With a PAKE protocol, if a user mistakenly attempts to authenticate herself to a phisher, the protocol will fail, but the user's password will remain safe. Since the phisher does not know the user's password, the phisher will not be able to successfully complete the protocol, and the browser can alert the user of the failure.

Goals and contributions. In this paper, we perform a systematic investigation of various practical issues and challenges in deploying PAKE for web authentication. Although many PAKE protocols have been proposed, there is little momentum for integrating PAKE protocols into web authentication. One contribution we hope to make in this paper is to help raise awareness of the issues inhibiting the widespread adoption of PAKE, and help stimulate future work and discussion in this area.

We investigate three categories of issues: 1) security issues related to UI design; 2) security issues related to the browser's same origin policy; and 3) potential hurdles to deployment. We propose potential solutions for some problems and identify areas for future work. An important contribution we make is to systematically lay out the issues surrounding PAKE that go beyond the cryptography. While many other issues may exist, we pick what we believe are the most important ones.

2 Problem Overview

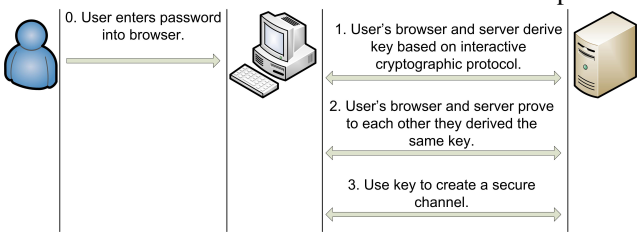
In this section, we introduce the background on PAKE, the main concept of PAKE-based web authentication, and set our problem scope.

*This material is based upon work partially supported by the National Science Foundation under Grants No. 0311808, No. 0448452, No. 0627511, and CCF-0424422, and by the Air Force Office of Scientific Research under MURI Grant No. 22178970-4170. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Air Force Office of Scientific Research, or the National Science Foundation.

2.1 PAKE-based Web Authentication

PAKE is a class of cryptographic protocols that allow two parties to establish a secret key based on a shared password. Since its proposal, PAKE has been studied extensively and many protocols have been proposed [5, 6, 25, 38, 53, 54, 39, 4, 3, 9, 40, 35, 33, 23, 22, 24, 23, 22, 41]. Some more recent protocols have provable security [40, 33, 24, 23, 22, 41]. Although many of these protocols are currently covered by patents, the patent on the EKE protocol and variants [5, 4, 10] will expire in 2011.

In PAKE-based web authentication, when a user wishes to authenticate herself to a website, she enters her password, and the browser and server run an interactive PAKE protocol to establish a session key based on the user's password without explicitly revealing the password in the process. The browser and server can prove to each other they derived the same key. If the protocol fails, the browser can alert the user, indicating that the server does not know the password that the user entered. The derived key can also be used to encrypt and authenticate their future communication between the two parties.



2.2 Problem Scope

Threats. We are concerned with the security of web authentication. In particular, we are concerned with an attacker who relies on phishing or other social engineering techniques to steal the user's password. We consider that the attacker may have network abilities, for example, the attacker can act as a man-in-the-middle and relay messages between the honest user and honest server, or the attacker may hijack the DNS of an honest website. The attacker may also exploit flaws in the browser's security policies to learn the honest user's personal and financial information.

Out-of-scope threats. A variety of threats exist in today's web many of which are system vulnerabilities beyond the scope of web authentication or this paper. For example, we do not attempt to solve attacks such as cross-site scripting or cross-frame request forgery. We also do not consider malware or kernel rootkits, as an attacker who controls the honest user's machine can potentially install a key-logger to capture the user's username and password. We point out that an extensive body of literature exists that investigates countermeasures to these threats.

In the remainder of this paper, we describe challenges and issues that need to be addressed for PAKE to be deployed in practice.

3 Security Issues Related to UI

PAKE-based web authentication is designed to reduce the attack surface of password leakage by using the PAKE protocol instead of the current form-based approach for authentication and key establishment. However, to gain such benefits in a real deployment, a few important UI issues need to be addressed beyond simply implementing the PAKE cryptographic protocol, otherwise, an attacker could exploit weaknesses in the UI to fool or confuse the user and steal the user's password. In this section, we discuss these issues.

3.1 Trusted Paths

UI Challenge 1 (Secure UI design)**.** *How can we design a user interface that is both usable, and minimizes the chance of human mistakes?*

Failure of in-page password forms. One naive solution is to have the user type in her password in in-page password forms as in the traditional password authentication approaches. However, this approach is insecure, since any design relying on in-page password input is subject to attacks—for example, a malicious JavaScript in-page may steal the user's input and sends it to the attacker. It is also difficult for the user to determine when she can trust that her password will only be used through a PAKE protocol and not be sent as cleartext or stolen by an attacker.

Trusted path. Thus, we need a trusted path for the user to enter her password and ensure that the password will only be used through the PAKE protocol. Different trusted path designs have been suggested in the past, including the use of an *in-chrome password box*, or a *trusted keystroke sequence*. The in-chrome password box approach places the password box inside the bordering frames of the browser, as it can be made difficult for the webpage to spoof the chrome. In the trusted keystroke sequence [45] approach, the user presses a special keyboard key or keystroke sequence (analogous to `ctrl+alt+del`) to notify the browser when the user wishes to login. When the key sequence is pressed, the browser disables all text entry except for in a trusted password box, successfully preventing replicate/mimicry attacks.

Each approach for establishing the trusted path also requires intricate design decisions for security. For example, one important design question for the in-chrome password box approach is when to display the password box: 1) *Display only when necessary.* The first approach is to display the password box only when the browser detects a login page. A similar approach was taken by Wu, et. al. in [52] using a sidebar login box and they found the approach was highly vulnerable to replicate or mimicry attacks. 2) *Always display.* Another option is to always display the password box, regardless of whether the page contains a password

form. This method prevents the chrome-imitation attack described above. However, we need dedicated space in the browser's chrome to display the password box at all times.

Even the "always display" option may have vulnerabilities. For example, in a replicate attack, the attacker imitates the in-chrome password box at the top of the page, causing two almost identical-looking password boxes to appear. When users fail to login through the real password box, they may be tempted to enter their passwords in the fake password box, possibly assuming that the interface is broken. A similar attack has been studied by Wu et. al. for their Web Wallet tool [52]. One way to defend against replicate attacks is to have the user customize the background of the in-chrome login bar, or display a personalized image next to the in-chrome login bar. Previous studies [12, 52, 46] have investigated and given suggestions for how to design such security skins. Given the different alternatives for UI design, one open research direction is to perform user studies to compare and evaluate their effectiveness.

It is worth noting that researchers have recently proposed methods to build trusted paths resilient to OS compromise [44] using a trusted computing approach. We can potentially make use of these designs for enchanced security.

3.2 User Education and Training

Users are accustomed to entering passwords in web pages, thus requiring them to always enter passwords through the trusted path may present significant challenges. Such challenges become even more severe during incremental deployment when legacy websites still use in-page login forms.

UI Challenge 2 (User education)**.** *What steps need to be taken to effectively migrate users to using the new trusted path (e.g., the in-chrome login bar)?*

Suggestions for adopting websites. User training can start to take place as good websites begin adopting the trusted path approach. Take the in-chrome login approach for example. In the early stages of adoption, a website may provide both the in-chrome login option (using PAKE) and the in-page login option (using traditional HTTPS). Websites could display a message next to the in-page password form that suggests users start using the in-chrome password box for higher security, and forecast that the in-page password form will be eliminated in the near future. At the end of this transition period, the login webpage removes the in-page password form completely, and explicitly requires that the user enter their passwords in the chrome. We emphasize that *only when the option of entering passwords in-page is completely eliminated, can we engage the user's attention with certainty*, and educate them to enter their password in the chrome. This is a lesson learned from security indicators: despite efforts at user education, security indicators prove to be ineffective, partly due to the fact that they fail to engage the users' attention [11, 20, 21, 46].

It is worth noting that adopting websites can provide *positive reinforcement* in the user training process. If a major website such as Google or Yahoo adopts this approach, it will immediately raise global awareness of this matter. As users go to Gmail to check their emails daily, their memory will be reinforced time and again. Such increased user awareness will pave the way for other websites to adopt this approach. Meanwhile, as more and more websites begin to adopt, they in turn contribute to the user training process, creating a positive feedback loop which will be beneficial towards wide-scale adoption.

Handling legacy websites. On the other hand, legacy sites that maintain the in-page login approach may effectively "untrain" the user. A crucial question to consider is whether we should allow users to use the in-chrome login bar (or any other trusted password box) to log into non-PAKE websites. If not, the user may think that the in-chrome login bar is unreliable, and may try to avoid using it whenever possible. It might be easier for the users if the in-chrome bar also supports non-PAKE logins. However, users now have to verify certificates even when they use the in-chrome login bar, and contrary to the belief that we are trying to instill in users, the in-chrome login bar is no longer safe at all times. Attackers can also exploit this, and trick users to enter their passwords into the in-chrome bar, while the password is actually sent in cleartext to the attacker. Another drawback of this approach is that valid websites may now have less incentive to actually deploy PAKE. In particular, websites might mimic what was done when SSL was deployed: they continue to use the unsafe approach and put in their websites messages and indicators attempting to convince the user that their connection is safe [26].

3.3 Designing Error Messages and Warnings

Users may have the temptation to fall back to the traditional in-page login approach when they fail to log in through the trusted user interface.

UI Challenge 3 (Error messages and warnings)**.** *How can we effectively communicate with the user when failures occur, so that they do not fall back to using insecure methods?*

Failure to login through the in-chrome login bar can result from the following reasons: 1) a network failure or the server fails to respond; 2) the user entered the wrong username and password; 3) the website is a fraudulent site; 4) if the in-chrome login bar does not support non-PAKE logins, in-chrome logins will fail at legacy websites.

Ideally, the browser should intelligently distinguish between these causes and give the user helpful suggestions accordingly. While it is easy to distinguish a network failure (e.g., the underlying TCP connection fails) from a login failure (e.g., the PAKE protocol fails), it may more difficult for

the browser to distinguish wrong username/password from a fraudulent website, especially when the attacker hijacks the DNS of a good website such as `bank.com`.

One way to infer if the user has entered the wrong username/password is to remember a hash of the username/password for each website every time a user successfully logs in. However, to avoid offline dictionary attacks in case of device capture, only a few bits of the hash should be stored.

Another method is to build more intelligence into the browser, so it can infer the trustworthiness of a website. This can be achieved through a combination of approaches. For example, we can build a reputation system for websites through community efforts, such as the approach taken by Perspectives [50]. SSL certificates can also serve as an indicator of the trustworthiness of a website. Utilizing these sources of information, the browser may be able to intelligently dignose the cause of a failed PAKE handshake: whether it is due to a wrong username/password combination or due to a fraudulent website. The browser should give different suggestions to the user in each case.

It is worth mentioning that the research community have conducted several interesting studies to evaluate the effectiveness of warnings and error messages [26, 15]. We can draw many lessons from these studies when designing the error messages and warnings for a PAKE-enabled browser. For example, how to prevent users' habituation to click through warnings, how to design warnings that engage the users attention, how to make suggestions, and how to make the users understand the security risks that they are taking.

3.4 Password Setup/Reset

Password setup and reset represent another weak link in the system, and must be considered.

UI Challenge 4 (Password setup and reset)**.** *How can we secure the account registration and password reset processes?*

We now describe why password setup/reset is particularly vulnerable, and suggest potential ways to secure the process.

Password reuse. PAKE does not protect users who reuse their banking password at an online game forum potentially hosted or compromised by an attacker. The password reuse problem can be alleviated through tools such as PwdHash [45, 27] which hashes the password along with the website's domain name and certificate. By hashing the password multiple times, we can potentially mitigate offline dictionary attacks [27]. Another promising approach recently proposed by Boyen [8] suggests how to retrieve different credentials from a remote server using a reusable password.

Social engineering. The attacker can set up a fake `bank.com` website and ask the user to setup/reset her password. The user is in danger if she sets up the same password at the attacker's website and at the real bank's website. Such social engineering attacks are particularly hard to prevent, especially with security-unconscious users. For example, the attacker can call up the user claiming to be the bank, and the user may be gullible enough to give away his password. It seems that the only way to address such attacks is to raise user awareness through long-term user eduction.

Suggestions. For a security-conscious user, however, we can potentially provide ways to help her authenticate the website.

1) The user and the bank can set up a password out of band, e.g., when the user goes to the bank to open an account.

2) If the user and the bank share some weak secret such as an account number or the social security number, the two parties can potentially use PAKE to authenticate each other to bootstrap the password setup/reset process.

3) In particular, for password reset, the user and the bank share knowledge of the security questions and answers. We can potentially use these as the shared secret to a PAKE handshake. To allow tolerance to error, we can imagine a fuzzy PAKE protocol, where two parties holding closely-related secrets can establish a secret key. This is a relatively new area [13], and it remains an open research problem to design an efficient and practical protocol for this purpose.

4) The browser or the system can help the user judge the trustworthiness of the website, for example, based on the site's certificate or its reputation collected through community efforts. This is in fact similar to the problem we encounter in today's web authentication where users need to consciously authenticate the website. Therefore, we can borrow techniques and learn lessons from existing approaches. It is an open research challenge how the browser can effectively give users suggestions.

5) To prevent man-in-the-middle attacks in the password setup/reset process, we can potentially use out-of-band channels such as SMS messages, which are hard to intercept and cost money for the adversary to send and receive.

4 Security Issues Related to SOP

As attackers may try to launch attacks by exploiting the browser's Same-Origin Policy (SOP), additional caution is needed when integrating PAKE into the browser.

SOP is the browser's access control policy, and prevents pages from different sites from accessing each other's web objects, including HTTP cookies, HTML documents, JavaScript, etc. The SOP works by assigning all received content an "origin identifier" which generally consists of the application layer protocol (e.g., HTTP, HTTPS, etc), the domain name: (e.g., `www.google.com`), and the TCP port. Browsers consider two resources to be from the same origin if their origin identifiers are the same.

SOP Challenge 1 (Working with SOP)**.** *How can we securely integrate PAKE into the browser to work with the browser's SOP?*

Isolating PAKE and non-PAKE sessions. A good website may offer both high security PAKE pages and low security non-PAKE pages (including traditional HTTPS and unencrypted pages). The low security pages may contain a vulnerability, and an attacker essentially becomes a man-in-the-middle if she successfully injects malicious scripts into these pages. Therefore, we must prevent non-PAKE pages from accessing PAKE-enabled pages.

One promising approach is to introduce a new protocol name for PAKE-enabled sessions, such as HTTPVS (Hyper-Text Transport Protocol - "Very Secure"). In this way, the browser will assign PAKE and non-PAKE sessions different origin identifiers, and isolate them.

Defense against the network attacker. If a user establishes two simultaneous sessions under the same origin and one of them is controlled by an attacker, then the malicious session will be able to access the real one. Recent research has shown that such attacks are possible if the attacker has control of the network [32, 29]. For example, in a DNS poisoning attack, the attacker hijacks the DNS of `bank.com`, and opens a connection with the user. Then, the user connects again to `bank.com` (in a separate window or in an iframe), and the attacker acts as a man-in-the-middle and forwards the user's connection to the real server. Now the attacker's session is able to read and write the session to the real server. Recent studies [32, 29] also demonstrate other possible ways to launch such an attack including the use of DNS rebinding.

Such attacks are possible due to inherent flaws in the browser's SOP, and are relevant regardless of whether PAKE is employed. One promising defense is to augment the same-origin identity with the hash of the password used to create a PAKE session [32, 29].

5 Deployment Challenges

We now consider deployment issues, including architectural choices and website customization of the login UI.

5.1 Architectural Choices

Deployment Challenge 1 (Web Application Architecture)**.** *What is the appropriate layer in the networking stack to integrate PAKE protocols?*

To examine this issue we analyze and compare two existing proposals for PAKE-based web authentication: TLS-SRP [49], which operates at the transport layer, and HTTPS-PAKE [42], which operates at the application layer.

TLS-SRP. Since PAKE may be needed by multiple applications, it would be desirable to implement it below the application layer from a system design perspective. This removes the trouble of having to develop and implement a unique PAKE standard for each application, thereby encouraging the use of PAKE by other applications. This is the approach taken by TLS-SRP, which integrates the Secure Remote Password Protocol (SRP) [53, 54] into the Transport Layer Security (TLS) suite. The TLS suite and its predecessor, the Secure Sockets Layer (SSL), are transport layer cryptographic protocols for establishing end-to-end secure channels for Internet traffic, most notably for HTTP traffic.

To create a TLS connection, a user's computer and a server must first negotiate a cipher suite. A cipher suite typically specificies the key negotiation method (e.g., Diffie-Hellman with RSA) and how the parties will use the negotiated key to create a secure channel (e.g., AES in CBC mode and HMAC with SHA-1). TLS-SRP extends TLS by supporting additional cipher suites that use SRP for the key negotiation phase. To employ TLS-SRP, the user's computer and the server first use SRP to interactively derive a symmetric key based on the user's password, and then create a secure channel using the latter part of of the cipher suite and the derived key.

Arguably, one reason why TLS has been so successful is that it is largely transparent to applications that use it. For example, after some initial configuration, many websites can typically just "turn on" TLS in their server software to enjoy many of its benefits. One advantage of this design is that websites can develop HTTPS applications that are independent from the choice of server hardware and software.

To employ TLS-SRP, this benefit of transparency may be lost. Since web applications typically manage users' identities and authentication credentials, TLS-SRP will require an inter-layer communication menchanism between application software and the TLS software to create and manage TLS-SRP sessions. For example, if a user's browser attempts to initiate a TLS-SRP session, the TLS software must communicate with the web application to obtain the the user's authentication credentials. It would also be desirable for the appplication to know if logins succeed and when TLS-SRP connections close.

This problem is exacerbated in site architectures that employ load balancing HTTPS front-ends. A common site architecture is to deploy dedicated load balancing machines that terminate TLS connections. These machines service TLS connections and route the underlying HTTP requests to the appropriate web servers, which may run on separate machines. Deploying TLS-SRP in this type of architecture will require new network protocols to manage user sessions between HTTPS front-ends and back-end application servers.

Another problem with TLS-SRP is that it does not cleanly support authentication to multiple realms within a single domain. For example, Google Apps provides "software-as-a-service" for email, document management, and information

sharing. The appearance is that each Google Apps customer gets a separate site hosted at `google.com` with unique user names and authentication credentials for each of their users. Supporting TLS-SRP authentication in this context is problematic. Current browser implementations of TLS will re-use the same TLS connection for all HTTPS requests to a particular domain. However, authenticating to different realms hosted at the same domain requires browsers to initiate seperate TLS-SRP connections for each realm, further complicating matters.

PAKE over HTTPS. To address the drawbacks of TLS-SRP, Oiwa et. al. proposed to implement PAKE at the application layer, over HTTPS [42]. We refer to their approach as HTTPS-PAKE.

HTTPS-PAKE is inspired by HTTP Digest Authentication, which authenticates users using the HTTP protocol. HTTPS-PAKE works by tunneling the PAKE protocol messages over HTTP, in additional HTTP headers. This approach gives the application layer control over authentication policies without any interaction with TLS. However, HTTPS-PAKE still relies on TLS to provide a secure channel between browsers and servers. With HTTPS-PAKE authentication, a user's browser first establishes a TLS connection with the server. Then, over the encrypted TLS session, the browser and the server perform a PAKE handshake and derive a PAKE key derived from the user's password.

In contrast to TLS-SRP, HTTPS-PAKE does not use the PAKE key to authenticate messages between the user's browser and server directly, but rather includes an "authentication value" derived from it in the HTTP request and response headers. To address phishing attacks that forward HTTPS-PAKE messages between a user's browser and a server in an attempt to impersonate the user, HTTPS-PAKE binds a user's password to authentication realms in way that resists these types of phishing attacks.

One drawback of HTTPS-PAKE is that by integrating at the application layer, it is vulnerable to stronger threats, such as pharmers and network attackers, and relies on users to detect these attacks. For example, suppose a user attempts to authenticate herself to `bank.com` with HTTPS-PAKE and a pharmer has hijacked DNS for `bank.com`. When the user initially connects to `bank.com`, she will reach the adversary. Assuming the adversary is unable to obtain a valid certificate for `bank.com`, the user will see a certificate warning, but studies suggest many users ignore these warnings. If a user ignores the certificate warning and attempts to authenticate herself with HTTPS-PAKE, the attacker can man-in-the-middle the PAKE protocol, which occurs over HTTP. The authentication will appear to succeed from the perspective of the user's browser and the server, but the session has been compromised by the attacker.

In contrast, TLS-SRP does not rely on users to detect pharmers and network attackers. If an adversary does not know the user's password, he cannot successfully complete the protocol, and the connection will fail.

5.2 Website Customization of Login UI

Since a website's login page can contribute to its branding, some sites may be hesistant to adopt a PAKE-based web authentication scheme if they lose some control of users' login experience.

Deployment Challenge 2 (Website customization of the login UI)**.** *How can we provide a secure user interface while still allowing websites to customize and brand the user experience?*

Customizing the anchor page. With a PAKE-enabled browser using an in-chrome login box, websites can still offer a customizable anchor page which informs the users that the page requires logging in, and suggests that the user enter their password in the in-chrome bar. The anchor page is also likely to provide a link for users who have forgotten their passsword.

It might also be possible to safely allow websites the option of limited branding within an in-chrome login mechanism. For example, each website can ask the chrome to display a customized image next to the in-chrome password box.

Customizing login failures. Websites may wish to be able to customize the login failure page as well. However, this raises a security issue since a malicious website can exploit this and ask the user to enter her password in the webpage instead.

One potential solution is for the browser to display a generic failure page or warning page before the user can proceed to the website's customized failure page. A danger of this approach is that users may tend to click through such warnings. Another approach is to allow websites to customize the failure pages in a restricted way. For example, the browser offers a template of a login failure page and the website fills in restricted forms of contents. Section 3.3 has more discussion on how to design error messages and warnings.

6 Implementation

To further study the problems surrounding PAKE deployment, we developed an open-source PAKE prototype for Mozilla Firefox [16]. Our implementation consists of the following components: 1) a component to handle the HTTPVS protocol; 2) a plugin that creates an in-chrome login bar for retrieving the user's credentials; 3) a plugin that implements the PAKE Session Manager (PSM); and 4) a patch that implements TLS-PAKE. For the TLS-PAKE implementation we chose to use Steffen Schulz's implementation of TLS-SRP in Mozilla's Network Security Services

(NSS) layer [47, 54]. The Secure Remote Password Protocol (SRP) is one specific instantiation of PAKE that does not have provable security. We leave it as future work to implement a version of PAKE that has proven security. In particular, our implementation makes the following contributions. 1) We implement the UI component missing from Schulz's implementation of TLS-SRP, and the PSM to connect the UI and TLS-PAKE. 2) We propose and implement the HTTPVS protocol handler, to defend against attacks on SOP.

7 Related Work

Authenticating the website. An extensive body of literature exists that helps users to decide when to trust a website. These approaches can be largely divided into three classes: 1) Security indicators and improved security indicators [17, 36, 37, 56, 28]; 2) Trusted paths or secure bookmarks with known sites [12, 52, 55, 43, 31]; 3) Automated detection and blacklisting of known phishing sites [1, 2]. Studies have shown that security indicators are ineffective [11, 20, 21]. Even with improved security indicators and warnings, users still find it difficult to interpret them [30, 46, 51, 52]. Identification and blacklisting of known phishing sites still suffer from inaccuracy problems [57], and browser vendors are unlikely to deploy this approach due to liability issues. PAKE-based web authentication is designed to precisely solve the difficult problem of how a user can authenticate a website.

Password management. Researchers have also proposed browser extensions to help users manage and strengthen their passwords [34, 45, 27, 55]. Password hashing can relieve users the burden of remembering too many passwords. These password management schemes are complimentary to PAKE-based web authentication.

HTTP Digest Authentication. HTTP Digest Authentication is an RFC standard that allows a client and a web server to negotiate a key without sending the password in the clear. Digest authentication [19, 18] was meant to supercede the basic HTTP authentication [7] in which passwords are sent in the clear. Digest authentication failed to take off due to many reasons. First, the cryptography is unsound, and it is susceptible to man-in-the-middle attacks and dictionary attacks. Second, its original implementations in IE were incompatible with the RFC standard. Third, HTTP digest authentication uses a trusted password box provided by the browser, and websites could no longer customize their login user interface. We speculate that this was another reason why HTTP digest authentication was not well-received.

8 Conclusion

We revisit the problem of PAKE-based web authentication, and investigate various issues that might inhibit its widespread adoption. We believe that it is important to ask and think about these questions, and we hope to stimulate discussions on this topic. We conclude by summarizing the good, the bad and the challenges of deploying PAKE-based web authentication.

The good. PAKE-based web authentication does have clear benefits over today's approach. First, when combined with the right UI, PAKE can reduce the attack surface of web-based authentication. Second, although careless users may still fall prey to social engineering attacks, at the very least, we can protect security-conscious users, and relieve them of the burden of deciding whether to trust a website. Moreover, with proper user education, we can hope to raise the global awareness of users over time.

The bad. On the other hand, PAKE-based web authentication will not address all the security problems with web authentication. Although proper user education can alleviate this problem, careless users may still enter their passwords in unsafe webpages. In particular, one weak link is social engineering attacks involving password setup and reset.

The challenges. We acknowledge that certain hurdles have to be overcome to deploy PAKE-based web authentication. First, we need to draw lessons from the failures of current web authentication and make careful UI design choices from the very beginning. Second, not only do browser vendors and websites need to collaborate in this effort, websites may also need to upgrade their TLS stack (including TLS accelerators) and application logic simultaneously to provide PAKE support.

Acknowledgement

We would like to thank Philip MacKenzie, Adam Barth, Serge Egelman, Markus Jakobsson and Chris Li for helpful discussions and valuable suggestions in preparing the paper.

References

[1] Earthlink toolbar featuring scamblocker for windows users. `http://www.earthlink.net/software/domore.faces?tab=toolbar`.

[2] Netcraft toolbar. `http://toolbar.netcraft.com/`.

[3] M. Bellare and P. Rogaway. The AuthA protocol for password-based authenticated key exchange. *Contributions to IEEE P*, 1363, 2000.

[4] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *Eurocrypt*, 2000.

[5] SM Bellovin and M. Merritt. Encrypted key exchange: password-based protocols secure againstdictionary attacks. In *1992 IEEE Computer Society Symposium on Research in Security and Privacy, 1992. Proceedings.*, pages 72–84, 1992.

[6] S.M. Bellovin and M. Merritt. Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and

password file compromise. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 244–250. ACM New York, NY, USA, 1993.

[7] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – http/1.0, 1996.

[8] Xavier Boyen. Hidden credential retrieval from a reusable password. In *ACM Symposium on Information, Computer & Communication Security—ASIACCS 2009*, 2009.

[9] Victor Boyko, Philip MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *Eurocrypt*, 2000.

[10] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Security proofs for an efficient password-based key exchange. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, 2003.

[11] R. Dhamija, JD Tygar, and M. Hearst. Why phishing works. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 581–590. ACM New York, NY, USA, 2006.

[12] Rachna Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *SOUPS '05: Proceedings of the 2005 symposium on Usable privacy and security*, 2005.

[13] Yevgeniy Dodis, Jonathan Katz, Leonid Reyzin, and Adam Smith. Robust fuzzy extractors and authenticated key agreement from close secrets. In *CRYPTO*, pages 232–250, 2006.

[14] Inc. E-Soft. Ssl server survey. `http://www.securityspace.com/s_survey/sdata/200701/certca.html`, 2007.

[15] Serge Egelman, Lorrie Faith Cranor, and Jason I. Hong. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In *CHI*, 2008.

[16] John Engler. Httpvs and the pake session manager for mozilla's firefox browser, March 2009. `http://www.freewebs.com/jengler/pake.htm`.

[17] R. Franco. Better Website identification and extended validation certificates in IE7 and other browsers. *Microsoft Developer Network's IEBlog. http://blogs. msdn. com/ie/archive/2005/11/21/495507. aspx*, 2004.

[18] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. Http authentication: Basic and digest access authentication, 1999.

[19] J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, and L. Stewart. An extension to http : Digest access authentication, 1997.

[20] Batya Friedman, David Hurley, Daniel C. Howe, Edward Felten, and Helen Nissenbaum. Users' conceptions of web security: a comparative study. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, 2002.

[21] Batya Friedman, David Hurley, Daniel C. Howe, Helen Nissenbaum, and Edward Felten. Users' conceptions of risks and harms on the web: a comparative study. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, 2002.

[22] Craig Gentry, Philip Mackenzie, and Zulfikar Ramzan. Password authenticated key exchange using hidden smooth subgroups. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, 2005.

[23] Craig Gentry, Philip Mackenzie, and Zulfikar Ramzan. A method for making password-based key exchange resilient to server compromise. In *CRYPTO*, 2006.

[24] Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. *J. Cryptol.*, 2006.

[25] L. Gong, MA Lomas, RM Needham, JH Saltzer, SRI Int, and M. Park. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, 1993.

[26] P. Gutmann. Security Usability Fundamentals (Draft).

[27] 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*, pages 471–479. ACM New York, NY, USA, 2005.

[28] A. Herzberg and A. Gbara. Trustbar: Protecting (even naive) web users from spoofing and phishing attacks. *Computer Science Department Bar Ilan University*, 2004.

[29] Collin Jackson, Adam Barth, Andrew Bortz, Weidong Shao, and Dan Boneh. Protecting browsers from dns rebinding attacks. *ACM Trans. Web*, 2009.

[30] Collin Jackson, Daniel R. Simon, Desney S. Tan, and Adam Barth. An evaluation of extended validation and picture-in-picture phishing attacks. In *Financial Cryptography*, 2007.

[31] Markus Jakobsson and Steven Myers. Delayed password disclosure. *SIGACT News*, 38(3), 2007.

[32] Chris Karlof, Umesh Shankar, J. D. Tygar, and David Wagner. Dynamic pharming attacks and locked same-origin policies for web browsers. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, 2007.

[33] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, 2001.

[34] John Kelsey, Bruce Schneier, Chris Hall, and David Wagner. Secure applications of low-entropy keys. In *ISW '97: Proceedings of the First International Workshop on Information Security*, 1998.

[35] T. Kwon. Authentication and key agreement via memorable password. In *ISOC Network and Distributed System Security Symposium*, volume 20, pages 31–33, 2001.

[36] C.S. Ltd. SpoofStick home page.

[37] N. Ltd. Netcraft Toolbar. *http://toolbar.netcraft.com*, 2005.

[38] S. Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In *Proc. of the Security Protocols Workshop, LNCS 1361*, 1997.

[39] P. MacKenzie and R. Swaminathan. Secure network authentication with password identification. *Submitted to the IEEE P*, 1363, 1999.

[40] Philip D. MacKenzie, Sarvar Patel, and Ram Swaminathan. Password-authenticated key exchange based on rsa. In *ASIACRYPT '00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security*, 2000.

[41] Minh-Huyen Nguyen and Salil Vadhan. Simpler session-key generation from short random passwords. *J. Cryptol.*, 21(1), 2008.

[42] Yutaka Oiwa, Hiromitsu Takagi, Hajime Watanabe, and Hideki Imai. Pake-based mutual http authentication for preventing phishing attacks (extended abstract). In *eCrime*, 2007.

[43] B. Parno, C. Kuo, and A. Perrig. Phoolproof phishing prevention. *Lecture Notes in Computer Science*, 4107:1, 2006.

[44] J.M.M.C.A. Perrig and M.K. Reiter. Safe Passage for Passwords and Other Sensitive Data.

[45] Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C. Mitchell. Stronger password authentication using browser extensions. In *SSYM'05: Proceedings of the 14th conference on USENIX Security Symposium*, 2005.

[46] Stuart E. Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. Emperor's new security indicators: An evaluation of website authentication and the effect of role playing on usability studies. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2007.

[47] Steffen Schulz. Implementierung sicherer passwort-authentisierung in mozilla tls, August 2007. `https://cbg.dyndns.org/store/srp/report_tls-srp.pdf`.

[48] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. Crying wolf: An empirical study of ssl warning effectiveness. manuscript, 2008.

[49] D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin. Using the Secure Remote Password (SRP) protocol for TLS authentication. Technical report, RFC 5054, Nov. 2007. `http://www.ietf.org/rfc/rfc5054.txt`.

[50] Dan Wendlandt, David Andersen, and Adrian Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *Proc. USENIX Annual Technical Conference*, 2008.

[51] Min Wu, Robert C. Miller, and Simson L. Garfinkel. Do security toolbars actually prevent phishing attacks? In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*,

2006.

[52] Min Wu, Robert C. Miller, and Greg Little. Web wallet: Preventing phishing attacks by revealing user intentions. In *Symposium On Usable Privacy and Security (SOUPS)*, July 2006.

[53] T. Wu et al. The secure remote password protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, volume 1, pages 97–111, 1998.

[54] Thomas Wu. Srp-6: Improvements and refinements to the secure remote password protocol. October 2002.

[55] Ka-Ping Yee and Kragen Sitaker. Passpet: convenient password management and phishing protection. In *SOUPS '06: Proceedings of the second symposium on Usable privacy and security*, 2006.

[56] K.P. Yee. Designing and Evaluating a Petname Anti-Phishing Tool. In *Poster presented at Symposium on usable Privacy and Security (SOUPS)*, pages 6–8, 2005.

[57] Yue Zhang, Serge Egelman, Lorrie Cranor, and Jason Hong. Phinding phish: Evaluating anti-phishing tools. In *In Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS)*, 2007.