Is it too late for PAKE?

John Engler, UC Berkeley
Chris Karlof, Usable Security Systems
Elaine Shi, PARC
Dawn Song*, UC Berkeley

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 [7, 18] and misunderstand or ignore browser security indicators [5, 9, 17].

We revisit the idea of applying Password Authenticated Key Exchange (PAKE) [1, 2, 3, 4, 10, 12, 13] 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 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. We investigate two categories of issues: 1) security issues related to user interface (UI) design, and 2) deployment and integration hurdles. The primary threat model we consider is an attacker who uses phishing or other social engineering techniques to steal a user's password. We hope our investigation will help raise understanding and awareness of the issues inhibiting the widespread adoption of PAKE, and stimulate further discussion in this area.

*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.

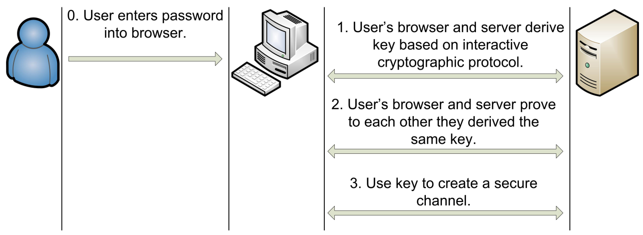
0. User enters password into browser.
1. User's browser and server derive key based on interactive cryptographic protocol.
2. User's browser and server prove to each other they derived the same key.
3. Use key to create a secure channel.

Figure 1: PAKE-based web authentication.

2 Security Issues Related to UI Design

To effectively integrate PAKE into browsers, we must carefully consider the UI provided for entering passwords. Otherwise, an attacker may be able to exploit weaknesses in the UI to fool or confuse users and steal their passwords.

UI Challenge 1 (Trusted paths)**.** *How can we design a UI that minimizes the chances of human error during attacks?*

One straightforward solution to the UI problem is to introduce a new input type for HTML forms, e.g., `<input type="pake">`. Unfortunately, this approach is dangerous. Honest sites may use the new input type, but malicious pages can continue to use legacy input types or employ JavaScript to steal users' passwords.

Therefore, it is desirable to use a *trusted path* for entering passwords. Two previously proposed trusted path mechanisms for web browsers are: 1) an *in-chrome password box*, and 2) a *trusted keystroke sequence*. The in-chrome password box approach places a protected password input (accessible only to the browser) inside the bordering frame of the browser, i.e., the "chrome". Since untrusted web pages cannot directly alter the appearance of chrome elements, there is hope that an in-chrome password interface would be harder for an attacker to spoof or hide.

An important design decision for an in-chrome password box is when to display it. For example, one approach is to display the password box only when the browser detects a login page or when a user requests to log in. A similar approach was taken by Wu et al. with a sidebar login box called Web Wallet [21]. However, their user study of Web Wallet suggests that this approach is vulnerable to a mimicry attack where an attacker spoofs the trusted in-chrome interface within the web page.

An alternative option is to always display the in-chrome password input. This method helps resist the chrome-imitation attack discussed above but requires dedicated space in the browser's chrome to always display it, which may be unsightly, cumbersome, or annoying. Unfortunately, the Web Wallet study suggests that the "always display" approach may also be vulnerable to spoofing attacks. Wu's study simulated an attack with a web page that mimicked the trusted interface, and when the real interface appeared, some users interpreted the two instances of the trusted interface as a browser error, and responded by closing the real interface. One potential defense against these attacks is to have users customize the background of the in-chrome login bar, e.g., with a personalized image [6, 17, 21, 23].

In the trusted keystroke sequence [16] approach, a user presses a special keyboard key or keystroke sequence (analogous to `ctrl+alt+del`) to notify the browser when she wishes to log in and enter her password. When the key sequence is pressed, the browser intercepts all text entry for the page or displays a trusted login box until the user has finished entering her password, protecting it from any malicious JavaScript on the page. Although this approach is promising, trusted keystroke sequences are vulnerable to "omission errors" where a user forgets to activate or intentionally omits the trusted sequence before entering the password. For usability reasons, trusted keystroke sequences often employ a feedback mechanism to indicate when the sequence has been entered correctly. If an attacker can spoof this feedback mechanism, he may be able to trick a user into believing the trusted sequence has already been activated and it is safe to enter her password. The Web Wallet study suggests that tricking users into committing these types of omission errors may be an effective attack vector.

Researchers have also proposed using trusted computing technology to build trusted paths, e.g., Bumpy, by McCune et al. [14]. This is a promising approach, but for space reasons, we do not discuss it in detail.

UI Challenge 2 (User education)**.** *How do we transition users to use a trusted path for entering their passwords?*

Many users are accustomed to entering passwords in web pages, and transitioning them to use a trusted path presents a significant challenge. In the early stages of adoption, to support legacy browsers and ease potential usability problems, we anticipate that web sites may provide both a trusted path login option (using PAKE) and the in-page login option (using traditional HTTPS). Websites could encourage users with supported browsers to start using the trusted path for higher security, and forecast that the in-page password form will be eliminated in the future. Unfortunately, some legacy browsers (e.g., IE 6) tend to have long lifetimes. If a user uses multiple browsers to access the same site, e.g., a legacy version of IE at work and a PAKE-enabled browser at home, it may not be obvious to users the circumstances under which to use the in-page login vs. the trusted path. This confusion may create opportunities for attack. We emphasize that *only when the option of entering passwords in-page is completely eliminated, can we engage the user's attention with certainty*. This relates to 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 [5, 9, 17].

A more challenging problem is the fact that many web sites will probably never transition to PAKE and continue to use in-page logins indefinitely. This will likely hinder user training for PAKE trusted paths. One potential solution is to allow users to use the trusted path for non-PAKE logins as well. However, this integration presents many challenges, including reliably detecting in-page forms, which may use Flash or JavaScript, and effectively signaling to users whether a particular login uses PAKE or not.

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?*

Users may have the temptation to fall back to traditional in-page logins when logging in with the trusted path fails. Failures could result from a variety of problems, including: 1) a network or server failure; 2) wrong username and password; 3) a fraudulent website; 4) trying to use the trusted path at a non-PAKE web site.

Ideally, messaging for PAKE should draw on lessons learned from users studies on the effectiveness of browser security warnings and error messages, e.g., for phishing detection and SSL [8, 11]. We must avoid habituating "click-through" PAKE related warnings, and design warnings that engage users' attention, make useful suggestions on how to proceed, and clearly convey security risks. However, 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 be more difficult to distinguish between a wrong username/password and a fraudulent website. One potential solution is to remember a hash of the username/password for each PAKE-enabled website and check this after a login failure. However, to avoid offline dictionary attacks in case of device capture, only a few bits of the hash should be stored. In addition, PAKE error messaging could incorporate other trustworthiness indicators, e.g., SSL certificate information from distributed reputation systems, such as Perspectives [20].

3 Deployment Challenges

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 [19], which operates at the transport layer, and HTTPS-PAKE [15], 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. This removes the trouble of having to develop and implement a unique PAKE standard for each application, thereby encouraging the use of PAKE by multiple applications. This is the approach taken by TLS-SRP, which integrates the Secure Remote Password Protocol (SRP) [22] 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.

To create a TLS connection, a user's computer and a server must first negotiate a cipher suite. A cipher suite typically specifies 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-CBC with HMAC-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 cipher suite and 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.

With 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 mechanism 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 user's authentication credentials. It would also be desirable for the application to know if a login succeeds and when the TLS-SRP connection closes.

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 separate TLS-SRP connections for each realm, further complicating matters.

PAKE over HTTPS. To address the drawbacks of TLS-SRP, Oiwa et al. propose to implement PAKE at the application layer, over HTTPS [15]. We refer to their approach as HTTPS-PAKE. HTTPS-PAKE is inspired by HTTP Digest Authentication, which authenticates users within 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 derived key to authenticate messages between the user's browser and server directly, but rather includes an "authentication value" derived from it in the HTTP headers. To address phishing attacks that forward messages between a user's browser and a server to impersonate the user, HTTPS-PAKE binds the user's password to authentication realms in way that resists these types of 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. 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 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. 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. One potential solution to this problem for HTTPS-PAKE is to leverage the PKI infrastructure and disallow PAKE authentication over HTTPS connections with an invalid certificate. A drawback of this solution is that sites with self-signed certificates would not be able to use HTTPS-PAKE.

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

Since the login page can contribute to a site's branding, some sites may be hesitant to adopt PAKE if they lose some control over users' login experience. Although it might be possible to allow websites the option of limited branding within the trusted path mechanism, this may enable new kinds of spoofing attacks against trusted sites, e.g., spoofing well-known brand images in the trusted path mechanism.

Websites may want to customize messaging for login failures as well. However, this raises additional security issues. An attacker may try to exploit this feature and convince users to enter their passwords in a conventional in-page login form on the attacker's page. It remains an open problem how to balance sites' customization requirements with effective defenses against these attacks.

4 Conclusion

Although PAKE-based web authentication offers some clear advantages over conventional in-page password authentication, it remains unclear whether these advantages are sufficient to overcome the hurdles it faces. The primary benefit of PAKE is that it offers a new, intuitive mechanism to authenticate web sites – if a login succeeds, it is probably the right web site; otherwise, it is probably not.

Unfortunately, the challenges facing widespread adoption and deployment of PAKE are non-trivial. If the trusted path mechanism for PAKE password entry in browsers is hard to use or restricts sites' branding and messaging capabilities, sites may be reluctant to adopt PAKE. Likewise, if PAKE does not integrate well with existing site architectures, sites may be reluctant to pay the cost of adopting and upgrading their infrastructure to support PAKE. Also, PAKE does not solve all the problems with password-based authentication, such as cross-site password reuse problems, weak passwords, and password reset/recovery vulnerabilities. Perhaps the most challenging aspect of deploying PAKE is the user re-education problem. Many users have become accustomed to entering passwords directly into web pages, and it may be a hard habit to break. However, at the very least, PAKE may offer security-aware users an easier way to authenticate web sites.

Acknowledgments. We thank Philip MacKenzie, Adam Barth, Serge Egelman, Markus Jakobsson, Chris Li, and the anonymous reviewers for their helpful comments.

References

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

[2] Steven M. Bellovin and Michael Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *CCS*, 1993.

[3] Victor Boyko, Philip MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *Eurocrypt*, 2000.

[4] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Security proofs for an efficient password-based key exchange. In *CCS*, 2003.

[5] R. Dhamija, JD Tygar, and M. Hearst. Why phishing works. In *CHI*, 2006.

[6] Rachna Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *SOUPS*, 2005.

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

[8] 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.

[9] 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*, 2002.

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

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

[12] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT*, 2001.

[13] Philip D. MacKenzie, Sarvar Patel, and Ram Swaminathan. Password-authenticated key exchange based on rsa. In *ASIACRYPT*, 2000.

[14] Jonathan M. McCune, Adrian Perrig, and Michael K. Reiter. Safe passage for passwords and other sensitive data. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, February 2009.

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

[16] Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C. Mitchell. Stronger password authentication using browser extensions. In *USENIX*, 2005.

[17] 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.

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

[19] 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`.

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

[21] Min Wu, Robert C. Miller, and Greg Little. Web Wallet: Preventing phishing attacks by revealing user intentions. In *SOUPS*, 2006.

[22] Thomas Wu. SRP-6: Improvements and refinements to the Secure Remote Password protocol. 2002.

[23] Ka-Ping Yee and Kragen Sitaker. Passpet: Convenient Password Management and Phishing Protection. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, pages 32–43, July 2006.