2010-03-10

Convenient Decentralized Authentication Using Passwords

Timothy W. Van Der Horst
Brigham Young University - Provo

Convenient Decentralized Authentication using Passwords

Timothy W. van der Horst

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Kent E. Seamons, Chair
Daniel M. A. Zappala
Dan R. Olsen, Jr.
Mark J. Clement
David W. Embley

Department of Computer Science

Brigham Young University

April 2010

ABSTRACT

Convenient Decentralized Authentication using Passwords

Timothy W. van der Horst

Department of Computer Science

Doctor of Philosophy

Passwords are a very convenient way to authenticate. In terms of simplicity and portability they are very difficult to match. Nevertheless, current password-based login mechanisms are vulnerable to phishing attacks and typically require users to create and manage a new password for each of their accounts. This research investigates the potential for indirect/decentralized approaches to improve password-based authentication. Adoption of a decentralized authentication mechanism requires the agreement between users and service providers on a trusted third party that vouches for users' identities.

Email providers are the de facto trusted third parties on the Internet. Proof of email address ownership is typically required to both create an account and to reset a password when it is forgotten. Despite its shortcomings (e.g., latency, vulnerability to passive attack), this approach is a practical solution to the difficult problem of authenticating strangers on the Internet. This research utilizes this emergent, lightweight relationship with email providers to offload primary user authentication from service providers; thus reducing the need for service provider-specific passwords. Our goal is to provide decentralized authentication that maintains the convenience and portability of passwords, while improving its assurances (especially against phishing).

Our first step to leverage this emergent trust, Simple Authentication for the Web (SAW), improves the security and convenience of email-based authentications and moves them from the background into the forefront, replacing need for an account-specific password. Wireless Authenticationg using Remote Passwords (WARP) adapts the principles of SAW to authentication in wireless networks. Lightweight User AUthentication (Luau) improves upon WARP and unifies user authentication across the application and network (especially wireless) layers. Our final protocol, pwdArmor, started as a simple wrapper to facilitate the use of existing databases of password verifiers in Luau, but grew into a generic middleware framework that augments the assurances of conventional password protocols.

Keywords: authentication, email-based authentication, passwords, password-authenticated key exchange, single sign-on, authentication in wireless networks

ACKNOWLEDGMENTS

I thank my wife Laura for her ever-present support and patience through many, many years of graduate school. Thanks to Dr. Kent E. Seamons for being a supportive advisor, friend and mentor. Thanks also to: Andrew Harding, Phillip Hellewell, Ryan Segeberg, Reed Abbott, and Trevor Florence. Finally, special thanks goes to my parents for helping me focus on the important things of life.

This research was supported by funding from the National Science Foundation under grant no. CCR-0325951, prime cooperative agreement no. IIS-0331707, and The Regents of the University of California.

Contents

Chapter 1

Introduction

People have too many passwords. Services on the Internet regularly require users to create *yet another* username and password. The attractiveness of passwords is clear: their basic concept is accessible to even the most novice user, they are easy to configure, and dealing with lost or forgotten passwords is completely automatable. As the number of accounts per user increases, the limitations of human memory forces users to balance the desire to use a strong password with the desire to log in without a lot of hassle. The latter usually wins, resulting in weak passwords that are reused across multiple accounts.

The goal of this research is to reduce the need for service-specific passwords, while improving the security and convenience of user authentication. There are several existing solutions designed to address the scalability of password-based logins (see Section 1.1). This research explores the middle ground between these existing solutions, and focuses on an emergent trust relationship (between service providers and email providers) that has the potential to dramatically simplify user authentications and improve their security (see Section 1.2).

1.1 Existing Solutions

Password managers, such as Password Safe [66] and the Mozilla Firefox Password Manager [28], solve the problems associated with multiple passwords by remembering the passwords on behalf of the user. A single password is then used to protect the password manager. Nevertheless, password managers generally lack portability and require significant setup and

account-level maintenance (i.e., the user still has to create/manage separate accounts across many sites).

Systems based on a public key infrastructure (PKI) have long been viewed as a solution to the shortcomings of passwords. In these approaches, trusted authorities are established and issue digital credentials, which contain certified assertions. Authentication is accomplished by the possessor of the certificate proving knowledge of the private key associated with the public key embedded in the credential.

At a high level, client-side digital certificates are an attractive replacement for service-specific passwords. The scalability and "offline" capabilities provided by public key cryptography are very desirable. This technology has also weathered well the test of time. The RSA algorithm is over 30 years old [71] and this year marks the twentieth anniversary of the X.509 certificate format [20]. Yet, despite the worthy success of PKI in server deployments (e.g., HTTPS), the proliferation of client-side certificates is remarkably underwhelming. The "imminent" arrival of client-side certificates seems always just out of reach. This delayed arrival is due, in part, to burdensome configuration and usage requirements as well as certificate distribution, management, and revocation issues [35]. High costs, the requisite user and administrator training in a complex technology, and the difficulties of trusted roots and cross-certification also make PKI hard to adopt [32].

A significant non-technological problem, as illustrated by Ellison and Schneier [27], is determining "Who do we trust, and for what?" As a certificate is essentially a secure container for assertions made by an issuer, clearly, the utility of these attestations is bounded by the authority of the issuer to make those claims.

If overcoming the problems barring their adoption was simple, PKI vendors would have crushed these obstacles to increased profits long ago. This is a difficult problem. Decades have passed and still client-side certificates remain the exception rather than the norm. This tepid acceptance leads one to question, given the current landscape, whether certificates are a *practical* replacement to service-specific passwords.

The goal of our research is to devise more convenient approaches with the potential for an immediate impact on this authentication challenge.

1.2 Our Approach

PKI-based systems follow a decentralized approach to authentication, i.e., *relying parties* (e.g., web sites) "rely" on *identity providers* to authenticate users on their behalf. This eases the burden on users and administrators by facilitating the reuse of login credentials across multiple domains. These systems also reduce risk for relying parties as user-specific authenticators (e.g., password files) are replaced with access control lists composed of public identifiers. In general, decentralized approaches have the potential to facilitate single sign-on, dynamic user bases with guest or temporary access, as well as cross-domain access control.

As noted by Ellison and Schneier (see Section 1.1), establishing trusted third party identity providers is a difficult problem. This research relies on the emergent trust between email providers and relying parties to address this challenge.

1.2.1 Emergent Trust in Email Providers

Proving ownership of an email address by retrieving a message sent to it has become a valuable approach to authenticating unknown entities on the Internet; it is typically required to create an account and to reset a password when it is forgotten. Despite its shortcomings (e.g., latency, vulnerability to passive attack), this approach is a practical solution to the difficult problem of authenticating strangers on the Internet.

Garfinkel [33] coined the term email-based identification and authentication (EBIA) to describe this imperfect, but extremely popular tool. This mechanism is typically relegated to a *secondary* means of authentication; used to verify ownership of an email address specified during account creation and when, almost inevitably, the account password is forgotten. Pragmatically speaking, EBIA more than adequately fills its current role, and will likely continue to do so, without any modification, within the foreseeable future.

1.2.2 Password-based Decentralized Authentication

Password-based decentralized authentication is a middle ground between client certificates and relying party-specific passwords. Several schemes (see Sections 3.7 and 4.6) have been proposed and are gaining popularity and acceptance (especially OpenID [62]). Nevertheless, these approaches are typically geared specifically for web-based logins and require users to connect directly to their identity providers *during* an authentication attempt; a process usually accomplished via browser redirects. This reliance on browser redirects introduces attractive avenues for password phishing. Also, this web-centric focus makes these systems unsuitable for use in scenarios where direct communication between users and their identity providers is infeasible, such as authentication to a wireless network. Lastly, none of these systems advocate, as we do, the specific use of email providers as identity providers.

1.3 Our Contributions

Password authentication is overused. Despite being a fundamental, and very popular, method of user authentication, typical users have no desire to remember the complicated, account-specific passwords necessary to secure this approach. Current alternatives require significant increases in overhead and complexity that are prohibitively expensive for many services. This dissertation explores, develops, and evaluates the potential that email, and other personal messaging providers offer as a simple and practical form of user authentication.

This research leverages the existing identifiers and authenticators used by personal messaging providers to define several authentication protocols that cover a spectrum of convenience, security, and cost-of-deployment. These systems are generalized so that they can operate at, and below, the application-layer (e.g., web sites logins) making it suitable for network-layer scenarios (e.g., authentication to wireless networks). This dissertation also identifies and explores several services that can take advantage of this new approach to user authentication.

1.3.1 Dissertation Outline

Chapters 2 to 5 are published papers and one technical report. Where noted, these papers have been expanded to include informative content that was omitted from the final published version, typically due to length requirements.

Chapter 2, "Simple Authentication for the Web (SAW)" [81], introduces the concept of personal messaging-based authentication as a primary means of authentication. This paper shows that, by extending and improving email-based password resets, personal messaging identifiers and authenticators can be conveniently, and securely, leveraged to provide authentication without requiring relying party-specific login credentials. This approach enables unmodified personal messaging providers to act as identity providers and requires no modification to client-side software. Although client-side software is not required, this paper demonstrates that the convenience and security of this approach can be improved through the adoption of client-side automation tools.

Chapter 3, "Wireless Authentication using Remote Passwords (WARP)" [39], expands the principles developed for SAW and explores the improvements that are possible with the active participation of personal messaging providers in the authentication process. WARP specifies a new protocol that addresses several issues that could significantly affect the convenience of SAW: 1) The latency inherent in personal message delivery; and 2) The requirement for direct client-side connectivity with the identity provider at authentication time. These issues are addressed by enabling the relying party to facilitate a secure, in-band authentication method that reuses the existing identifiers and authenticators shared between users and their messaging providers. These improvements require client-side protocol support as well as active participation by the personal messaging provider. This approach also provides improved protections against active attack.

Chapter 4, "Lightweight User Authentication (Luau)", improves the framework created for WARP and significantly improves its flexibility and assurances. In Luau, the SAW token exchange mechanism is replaced with key exchange (optionally, authenticated key ex-

change) techniques. The in-band authentication method is generalized so that a variety of existing methods can be used to authenticate the user to their identity provider. As it is not coupled to a personal messaging medium, this framework can be leveraged by a variety of identity providers. Luau provides strong protections to users' login credentials and can be tailored to meet the needs and requirements of specific organizations and their users and enables convenient escalation (or de-escalation) of assurances should the need arise.

Chapter 5, "pwdArmor" [82], was initially developed as simple wrapper to facilitate the use of existing password verifiers in Luau; a small feature for facilitating the adoption of Luau within very specific scenarios. This system grew into a generic framework that could improve the assurances of a wide variety of conventional password protocols. In addition to fulfilling its intended role in Luau to a more valuable extent that was originally believed, pwdArmor can be used to improve the privacy and augment the assurances of existing service-specific password-based logins that rely on encrypted tunnels. Because pwdArmor treats server authentication as an added bonus, rather than the linchpin of its assurances, it is a valuable tool to combat password phishing.

Chapter 6 briefly explores the various systems that others have built using these technologies. One system, EPAK [40] extends Kerberos to enable the integration of our new authentication schemes. A second system, CPG [1] builds an anonymous discussion board for a closed group. Only legitimate group members can participate in the discussion and their identities are concealed, not only from the other members of the group, but from the administrators as well. As the actions of a group member are linked together, an abusive group member can be banned, without compromising anonymity, to prevent future abuses. A third system, EASEmail, provides a secure email scheme designed to simplify key distribution and portability issues and improve usability.

Chapter 7 summarizes the contributions of this research, presents conclusions, and discusses potential directions for future work.

Chapter 2

Simple Authentication for the Web (SAW)

T. W. van der Horst and K. E. Seamons. Simple Authentication for the Web. *3rd International Conference on Security and Privacy in Communication Networks (SecureComm)*, Nice, France, September 2007.

Additional Content: The published version is augmented here with Figure 2.3 (the administrative interface to add users to a SAW-enabled blog). A discussion of one-round SAW is also added by way of Section 2.5.5 and Figure 2.8.

Abstract

Automated email-based password reestablishment (EBPR) is an efficient, cost-effective means to deal with forgotten passwords. In this technique, email providers authenticate users on behalf of web sites. This method works because web sites *trust* email providers to deliver messages to their intended recipients. Simple Authentication for the Web (SAW) improves upon this basic approach to user authentication to create an alternative to password-based logins. SAW: 1) Removes the setup and management costs of passwords at EBPR-enabled sites; 2) Provides single sign-on without a specialized identity provider; 3) Thwarts passive attacks and raises the bar for active attacks; 4) Enables easy, secure sharing and collaboration without passwords; 5) Provides intuitive delegation and revocation of authority; and 6) Facilitates client-side auditing.

2.1 Introduction

Password logins are overused. Reusing a password across all web sites is risky, but managing multiple passwords results in frequently forgotten passwords. Many web sites handle forgotten passwords by emailing the user a password or a hyperlink to a facility to reset the password, a technique we refer to as email-based password reestablishment (EBPR).

This paper introduces Simple Authentication for the Web (SAW), a new web site login approach that improves both the security and the convenience of EBPR. SAW utilizes email for all authentications and not just for recovering from forgotten passwords. SAW also provides single sign-on and can be fully automated so that the login details are transparent to users.

2.1.1 Alternatives to Passwords

Password managers, such as Password Safe [66] and those built into popular web browsers, solve the problems associated with multiple passwords by remembering users' passwords. A single password is then used to protect the password manager. Nevertheless, password managers generally lack portability and require significant account-specific maintenance.

Other alternatives to passwords (see Section 2.7) require a specialized identity provider. As is evidenced by their lack of widespread adoption amongst web sites, finding a mutually trusted identity provider is difficult.

In 2003, Garfinkel [33] coined the term email-based identification and authentication (EBIA) to describe the general concept of using an email address as an identifier and the ability to receive email messages sent to that address as an authenticator. In evaluating this current trend, Garfinkel argued that EBIA's widespread use is evidence that the risks of this system are manageable, especially given that the alternatives are prohibitively expensive for many web sites.

The remainder of this paper is organized as follows. Section 2.2 provides an in-depth look at the goals and design of SAW. In Section 2.3, we discuss a prototype implementation

of SAW for web site logins. Section 2.4 presents a detailed evaluation of the threats to this system. Section 2.5 presents some advanced features of SAW. Section 2.6 shows how SAW overcomes the obstacles facing email-based authentication. Section 2.7 discusses related work. Section 2.8 contains conclusions and a discussion of our future work.

2.2 SAW

As password-based, EBPR-enabled sites already demonstrate their willingness to offload user authentication to email providers, the presentation and evaluation of the SAW protocol focuses on deployment to web sites inclined to assume the risks of EBPR. Sites reluctant to accept these risks (e.g., online banks) can still benefit from SAW by deploying it as an additional factor of authentication.

For simplicity, this paper defines a secure login as one that uses HTTPS and an insecure login as one that does not. We acknowledge the existence of password-based authentication mechanisms (e.g., SRP [87], DH-EKE[76]) that securely operate over insecure channels, however use of these protocols by web sites is rare. Both secure and insecure logins are widely used and would benefit from the adoption of SAW.

2.2.1 Obstacles to Adoption

At EBPR-enabled web sites, users prove their identity by using their password or by demonstrating ownership of their email address. If the ability to receive email messages is sufficient to circumvent users' passwords, why not make email the primary means of authentication and remove site-specific passwords? We identify four obstacles:

Latency In some cases, email message delivery and retrieval may require a relatively long period of time.

Lack of privacy Email messages are typically sent without cryptographic protection and are therefore susceptible to passive eavesdropping and active modification.

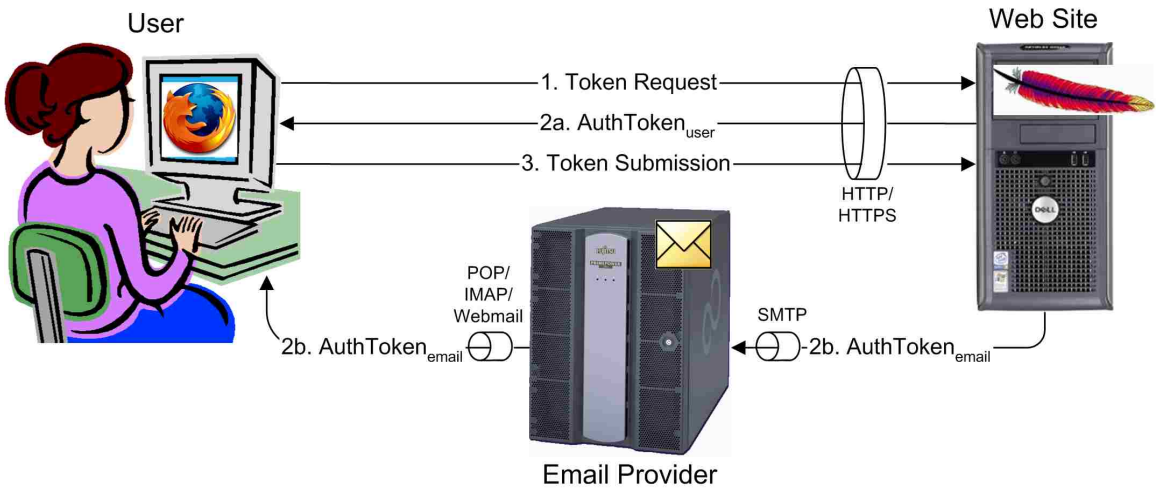


Figure 2.1: The SAW protocol. Based on the user's email address, specified in (1), a web site creates and distributes two authentication tokens. AuthToken$_{user}$ (2a) is sent directly the user as an HTTP cookie. AuthToken$_{email}$ (2b) is emailed to the user. Both tokens must be returned to the web site (3) to successfully authenticate. Each web site login attempt involves its own unique, short-lived, single-use tokens.

Convenience Password-based systems are pervasive and accepted by both users and web sites. Changing a web site's login system often requires significant time and resources as well as additional user training.

Reliance on a third party By involving an email provider in the authentication process, a dependency upon a third party is introduced. If the email provider is unavailable, the authentication process cannot succeed.

2.2.2 Goals

The design goals for SAW are:

- Remove the need for users and web sites to setup and manage passwords
- Provide web single sign-on via email
- Require no modification to email providers
- Thwart all passive attacks
- Raise the bar for active attacks

- Overcome the obstacles identified in Section 2.2.1
- Reduce user involvement through automation to make logins more convenient and reduce the attack surface for phishing and social engineering attacks
- Offer advanced features usually unavailable in password-based systems
  - Easy, secure sharing and collaboration without passwords
  - Intuitive delegation and revocation of authority
  - Client-side auditing

The remainder of this section describes SAW's improvements in security over existing email-based authentication systems. In-depth discussion of the advanced features of this system and how it addresses all of the obstacles listed above is left to Sections 2.5 and 2.6, respectively.

2.2.3 Protocol

The steps to authenticate using SAW (see Figure 2.1) are as follows:

Token Request

The user submits (e.g., via a webform) her email address to a web site in the *Token Request* message. This request *may* be sent over HTTPS (see Section 2.4.1).

Token Response

The web site, based on the permissions of the email address, creates several short-lived, single-use *Authentication Tokens*, hereafter referred to as $AuthToken_x$, where x is the identifier of the specific token. These tokens are created based on a security parameter k.

If the address is authorized, $\text{AuthToken}_{complete}$ is chosen at random from $\{0,1\}^k$, and split into two shares as follows:

$$AuthToken_{user} = AuthToken_{email} \oplus AuthToken_{complete}$$

where AuthToken_{email} is another random value from $\{0,1\}^k$. This method of token creation is a conventional secret splitting scheme [73] and provides the perfect security of one-time pad encryption. No amount of computation will recover one share without the other two.

$\text{AuthToken}_{complete}$ is cached by the web site in a temporary look-up table. AuthToken_{user} is returned directly to the user as an HTTP cookie. An email message containing AuthToken_{email}, and the identity of the web site that created it, is sent to the user's email address.

If the address is not authorized, a random member of $\{0,1\}^k$ is returned as AuthToken_{user}, and, in lieu of AuthToken_{email}, a human readable explanation of the failure is emailed. Always returning an AuthToken_{user} prevents an impersonator from learning anything about the email address owner's permissions. This is reminiscent of password systems that do not disclose that a username is invalid.

Token Submission

The user retrieves the email message sent by the web site, extracts AuthToken_{email}, and returns both tokens to the site. If these values combine to equal the $\text{AuthToken}_{complete}$ for that particular user and token identifier (see Section 2.2.4), then the authentication is successful and the system uses a session-level trust preservation mechanism for the remainder of the session (e.g., a session cookie).

2.2.4 Token Identifiers

Each set of tokens is given a unique identifier, $transID$, to facilitate the matching of an AuthToken_{user} with its corresponding AuthToken_{email} and to enable web sites to look up the correct $\text{AuthToken}_{complete}$. As every Token Request results in the creation of a new set of tokens, this identifier allows a user to have multiple concurrent authentications in progress with the same web site, thus permitting a user to retry the authentication process without invalidating a previous attempt. Appending $transID$ to the *name* of the HTTP cookie used

to convey AuthToken$_{user}$ prevents the cookies from current authentication attempts from being overwritten.

2.2.5 Client-side Automation

Manually polling an email account for a specific message is inconvenient and unnecessary. Additionally, a carefully crafted phishing email resembling a token message could lure an unsuspecting user to a malicious site.

Automating the process of retrieving and submitting AuthToken$_{email}$ enhances the convenience and security of SAW. The user's software agent verifies a token by checking its sender and identifier with its list of outstanding authentication attempts. It then submits the tokens to the desired site. Although a user could perform these same tasks, the user's software agent is able to do it much faster and more accurately.

Ideally, web browsers would have native support for SAW to allow single sign-on to all SAW-enabled web sites while the browser is open by having the browser cache the email account password for a limited time. Section 2.3 presents a prototype browser toolbar that implements these benefits.

2.2.6 Alternatives to Email

SAW provides personal messaging-based authentication. AuthToken$_{email}$ is easily delivered over a variety of personal messaging mediums (e.g., instant and text messaging). SAW provides a platform to explore the potential that these personal messaging systems or a hybrid combination of these mediums have for authentication.

Instant messaging, in addition to providing an attractive, low latency alternative for delivering AuthToken$_{email}$, also facilitates the use of SAW by those who rely on free web mail accounts (e.g., Hotmail, Yahoo! Mail, Gmail) since the programmatic access (i.e., POP/IMAP) to these accounts necessary for client-side automation is often only available to "premium" accounts. As these providers associate free instant messenger accounts (e.g.,

From: SAW_TokenGenerator@securecomm.org
To: TestSubject@some.edu
Subject: [SAW-https://securecomm.org/login] TransID=fc9f7de...

This email can be used in one of two ways:
If you are using the SAW toolbar:
This message will be handled and deleted automatically.

If you are NOT using the SAW toolbar:
Click on the link below ONLY if you recently initiated a
request to log in to https://securecomm.org/login:
https://securecomm.org/login?TransID=fc9f7de2d1f6e5f4f93fa42ff
aa9c13151450ffd4e1f73786346fed86e84d851&ATemail=2fe31bf9
c5d91e3d2ebc7688c812ba51d203ef6cfda5d99cbe7f5472b8e3ad57

Figure 2.2: An email message with AuthToken_{email}. The hyperlink is a convenient means for users who without the SAW toolbar to return the token to the site that created it and prevent inadvertent token disclosure to phishing sites.

MSN Messenger, Yahoo! Messenger, Google Talk) with each email address, users are able to leverage the same password as their email account to enjoy the benefits of SAW and client-side automation.

2.3 Implementation

There is a myriad of web sites that are ideal candidates for adopting SAW. These systems include: blogs, e-commerce sites, photo sharing sites, digital libraries, forums, conference program committee sites, private wikis, mailing lists, and personal web sites.

SAW provides several advanced features (see Section 2.5) that make it an even more attractive alternative to passwords. For example, since SAW provides decentralized authentication of users, it enables web sites to specify permissions for users outside its local security domain. This is powerful tool for sharing and collaboration (see Section 2.5.1).

2.3.1 Blog Access Without Passwords

We added support for SAW to Wordpress [85], a popular web log platform, by creating a plug-in (see Figure 2.3). Based on email addresses supplied by the login page, this plug-

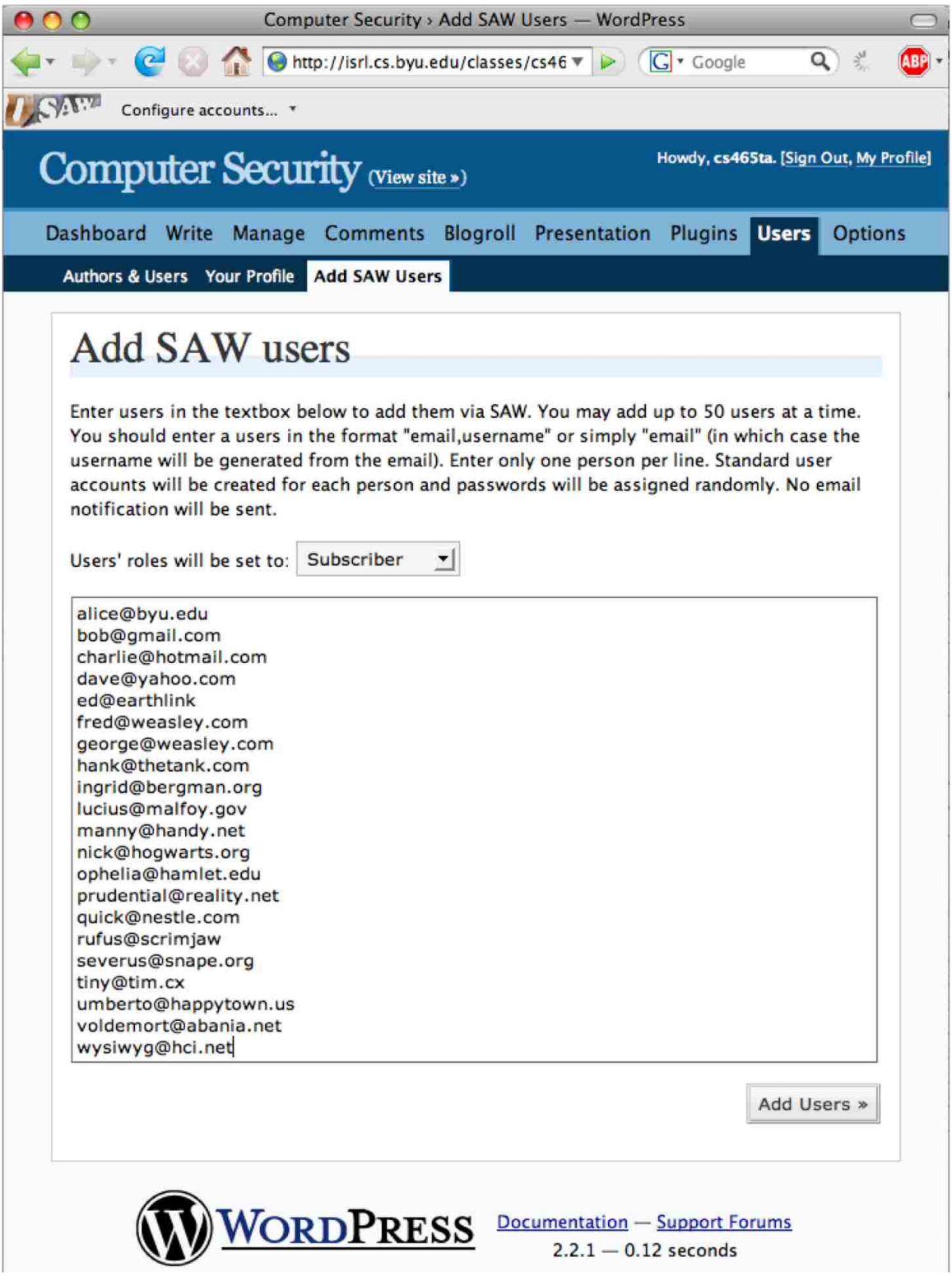


Figure 2.3: This interface enables an administrator to easily add a large number of users to this blog. As these users will authenticate using SAW, it not necessary to create/distribute a password for each account.

Figure 2.4: A login page and the SAW toolbar. Clicking the toolbar's login button submits the selected email address, retrieves the email sent by the web site, and submits both authentication tokens. Without the toolbar these actions must be performed manually. (Note that only an email addresses is required.)

in creates and distributes the authentication tokens and then displays a page containing instructions to complete the authentication. With the client-side toolbar, the remainder of the login process is automated.

Without the client-side toolbar, the user must authenticate to the email provider and retrieve the message containing AuthToken_{email} (see Figure 2.2). To facilitate the transfer of this token to the web site, this message includes a hyperlink that contains the token. Clicking on the link submits AuthToken_{email} along with AuthToken_{user} since it is a cookie set by that site. Current browser designs protect cookies from being sent to any unintended site, thus eliminating the possibility of submitting AuthToken_{user} to a phishing site.

The hyperlink in the email message exists as a convenience to users without a client-side automation tool. When such a tool is used, the entire message body can be dropped because all required information is contained in the subject line.

The client-side toolbar (see Figure 2.4) is implemented for both Internet Explorer and Mozilla Firefox. These toolbars share a common service that manages email account

	Time to check for messages	Total login time using email	Total login time using IM
Gmail account	2.5 sec	4 sec	<1 sec
Private account	1 sec	1.5 sec	n/a

Table 2.1: Initial performance results using a high-traffic, free webmail account (Gmail) and a lower-traffic, private email account.

information and communicates with email providers via POP or IMAP. It can also receive instant messages using XMPP [72].

These toolbars provide a recognizable, uniform interface for authenticating to SAW-enabled web sites. They detect sites that accept SAW and provide a "Log in" button that enables one-click authentication to the web site.

Clicking on this button prompts the user for the email account password (once per session), submits the desired email address/IM handle to that site and begins checking the specified email, or IM, account for the message containing AuthToken_{email}. By remembering the email account password, the toolbar provides transparent single sign-on to all SAW-enabled sites for the current session.

2.3.2 Performance

An initial performance analysis shows promising results (see Table 2.1). Before examining specific numbers, it is important to note that the time required to authenticate in SAW includes not only the time it takes for an email message to be delivered to an email provider, but also the time required to retrieve it from the user's mailbox. This analysis used two different email accounts and results are an average of 50 iterations.

The first account is a high-traffic, free webmail address provided by Gmail. This account is accessed using POP over TLS and requires 2.5 seconds for the toolbar to connect

and retrieve an email. It takes 4 seconds from the moment the user clicks the log in button until authentication completes.

The second account is a lower-traffic, private email address that is also accessed via POP over TLS. This account requires one second to connect and retrieve an email and takes 1.5 seconds for completion of the entire login process.

When instant messaging is used (only the Gmail account provides IM capability) the entire login process takes less than a second.

2.4 Threat Analysis

This section analyzes the threats and attacks to SAW. First, it considers the affects of passive eavesdropping on this protocol's communication channels. Next, it examines the ability of an attacker to actively impersonate a valid user. Then, it shows how this system mitigates the risks of password phishing, denial of service attacks, and spam. Finally, it examines the threats to login information stored at the web site, user, and email provider and how to address stale email addresses.

2.4.1 Passive Eavesdropping

Current EBPR methods are vulnerable to passive attacks. By emailing plaintext passwords, or a link to facilitate a password reset, anyone sniffing the network will be able to compromise the user's account. This section discusses the protections SAW required for each of its three communication channels:

User and Web Site

Although HTTPS provides confidentiality and integrity to the login process as well as authentication of the web site, a significant number of EBPR-enabled sites lack this protection. If an insecure channel is used for communication between the user and the web site, then

the submission of a user's email address and the contents of the Token Submission message are passively observable.

Under these conditions, SAW closely resembles a one-time password system; authentication tokens become worthless to an attacker once users have submitted them to the site. SAW has the added benefit that these tokens must be used within a short time frame. This is a significant improvement over EBPR-enabled web sites with insecure login pages.

Note that without an HTTPS connection between the user and the web site, it is possible for an eavesdropper to see everything users see when they interact with the site and it may be possible for an attacker to hijack the user's session.

Also, without HTTPS on this link SAW cannot detect or prevent an active man-in-the-middle (e.g., an attacker with a spoofed domain name) and a replay attack would be possible.

Web site and Email Provider

Ideally, a secure channel should be also used for the communication between web sites and email providers, however, because the vast majority of email traffic currently does not have any cryptographic protections in place, it is not feasible, at this time, to make this a requirement.

Fortunately, the AuthToken_{email} that is sent over this link is useless without the AuthToken_{user} that was delivered directly to the user. When an HTTPS connection is employed between the user and the web site, AuthToken_{user} is never visible to an eavesdropper and the risks of sending AuthToken_{email} in the clear are mitigated.

Email provider and User

The link between the email provider and the user requires a connection that provides confidentiality, integrity, and authentication of the email provider (e.g., POP/IMAP over TLS). This prevents local eavesdroppers from collecting the email account password.

2.4.2 Active Impersonation

Provided at least one token is securely sent to the user, SAW eliminates the potential for passive attack. Nevertheless, attackers able to monitor the communications between the web site and users' email providers are positioned to mount an *active impersonation* (see Figure 2.5).

By submitting a Token Request, an active attacker obtains a valid AuthToken_{user}. By eavesdropping on the unprotected link between the web site and the email provider it is also possible for the attacker to intercept the corresponding AuthToken_{email}. Email providers, or more likely malicious insiders, are also able actively impersonate their users.

We believe that SAW is workable and usable, even in light of this attack. Sites that employ EBPR are also susceptible to a similar of attack in which an attacker requests a password reset for the victim's account and then eavesdrops the resulting email message sent by the web site. Nevertheless, as Garfinkel [33] states, the widespread adoption of EBPR indicates that these risks are manageable.

This potential for active impersonation brings to light a problem that SAW shares with PKI-based systems. Both systems are vulnerable to malicious or compromised trusted third parties; a certificate authority (CA) or email provider has the ability to undetectably impersonate its clients. Just as a PKI-based system is only as strong as its CAs, SAW is only as strong as its email providers.

The ability to deliver email to only its intended recipients is a required characteristic for email providers who want to retain their users. Some email providers will be more desirable than others in the same way that some CAs are more trusted than others because of their reliability and privacy practices.

Section 2.5.4 describes several optional mechanisms that will prevent active impersonations by email providers and make it more difficult for an external entity to mount this attack.

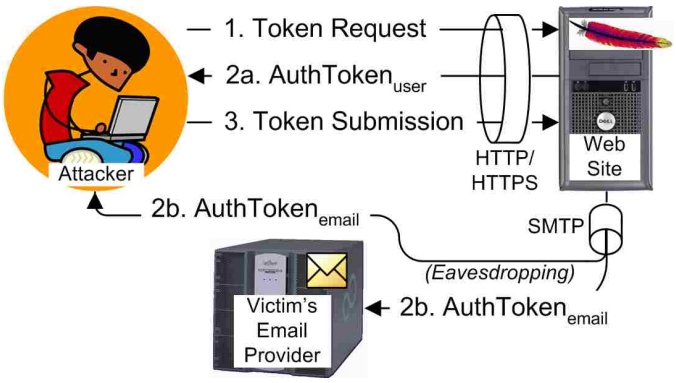


Figure 2.5: In an active impersonation, the attacker submits the victim's email address in a Token Request (1). AuthToken$_{user}$ (2a) is returned directly to the attacker and, by eavesdropping the communication between the web site and the victim's email provider, the attacker obtains the corresponding AuthToken$_{email}$ (2b). The attacker is now able to impersonate the victim to that web site (3). EBPR-enabled sites are also vulnerable to a similar attack.

It is possible to tunnel SMTP through a TLS connection; however it is not a common practice to do so. This is most likely due to the overhead involved in creating and accepting a large number of TLS connections. If, however, TLS was used to transfer messages between email providers this would eliminate the ability for an external entity to perform an active impersonation. Unfortunately, it does not thwart a malicious or compromised email provider from implementing this attack[1].

Tunneling SMTP through a TLS connection reduces the attack surface for active impersonation attacks, raising the bar significantly compared to current EBPR practices. However, current EBPR practices indicate that SAW is already workable and usable at many web sites without this advanced protection. Requiring secure email delivery strengthens SAW and could make it attractive to web sites that require stronger guarantees against active impersonation.

[1]Splitting AuthToken$_{email}$ amongst multiple email will prevent non-colluding email providers from mounting this attack (see Section 2.5.4).

2.4.3 Password Phishing and Denial of Service

Phishing lures potential victims into divulging sensitive information (e.g., passwords) by emailing an official-looking message containing a link to a site that looks like a web site used by the victim. Since SAW eliminates site-specific passwords, the only thing a phisher gains from tricking users into attempting to authenticate a phishing site is an email address, which the attacker already knows because she sent the phishing message to it.

Unfortunately, although SAW protects users' ability to authenticate to specific web sites from phishers, it does not prevent an attacker from tricking users into believing that they have successfully authenticated and are now interacting with the real web site. This remains a difficult open problem, not just for SAW, but for web site logins in general.

As clicking links found in email messages is potentially dangerous, such links should only be followed when they are the result of a Token Request. The short time span between requests and delivery aids users in following this practice. Using hyperlinks is more secure than having the user cut-and-paste the token because it ensures that the token will be returned to the web site that issued it.

A race condition is possible when emailing hyperlinks to users. If an attacker is able to email a spoofed message before the web site's email message arrives, the attacker may be able to fool users into following a link to a malicious site. Although the users visit the attacker's site, none of the authentication tokens are disclosed to it.

Unfortunately, a quick visit to a malicious web site can be quite dangerous due to browser vulnerabilities, or sly social engineering, that result in the installation of malicious software. This is a very real threat and demonstrates the advantage of taking the user out of the loop and allowing these actions to be performed by the toolbar, which employs strict token matching to remove this race condition.

There are several ways to mitigate denial of service attacks. For example, a web site can limit the number of messages that it sends out to a single address. Also, client-side

software can recognize unsolicited authentication emails as spam and automatically discard them or file them away for later analysis.

Also, it is possible for the token messages themselves to be categorized as spam. Garfinkel [33] suggests that emails used for authentication purposes should be signed by the sender to prevent this. This approach, however, may be too expensive for many web sites. In SAW, token messages are very easily identified and it is trivial to teach a spam filter that they are not spam. Any unrequested messages that do get through can be handled by the client-side software as stated above.

2.4.4 Storage of Login Information

In SAW, there is no long-term storage of sensitive user-specific authentication information at web sites. Although values for $\text{AuthToken}_{complete}$ are cached for users currently attempting to authenticate, these values are short-lived and are only used once.

There is also no long-term storage of authentication information by users. Although client-side automation tools may store email account information, the account password should never be stored outside of a session.

It is probable that email providers will maintain a long-term storage of messages containing an AuthToken_{email}. Fortunately, this token is useless without its corresponding AuthToken_{user} and both tokens are short-lived and single-use. This mitigates the risks of storing AuthToken_{email} at the email provider for auditing purposes. Offline analysis of the individual tokens yields no new information because each $\text{AuthToken}_{complete}$ is as likely as the next. Token size, determined by k, should be chosen to be sufficiently large to thwart any brute force attempt to guess them. Online, brute force attacks for the two tokens are trivially detected.

2.4.5 Stale Email Addresses

Allowing users to change their primary address at a web site is easy when their original email addresses are still accessible. When doing so, it is important that this process require an authentication challenge to the original address. A similar concept (requiring a user to enter the original password before it is changed) exists in many password-based systems.

A *stale email address* is an email address that has become inaccessible to the user. One approach to address this problem is to have a password backup or secret question/answer. These methods switch the traditional roles of emails and passwords as primary and secondary authentication mechanisms. Lamentably, this method negates many of the benefits of SAW, as it reintroduces passwords. We identify two approaches to deal with this problem.

First, if an email address is no longer accessible, a user can use whatever out-of-band techniques that were used to create the account at the web site in the first place. For example, if the email address of a member of a conference program committee goes stale, that member can contact the committee chair and have the email address changed to a new one.

Systems with open user registration usually do not have an out-of-band means to manage accounts. The simplest solution is to create a new account. However, it may be the case that the user has information stored at the web site under the old account, e.g., a collection of photos. In this situation, a secondary email address, preferably in a completely different domain than the primary one, registered with the web site during account creation presents an elegant solution for the vast majority of users. This method also mitigates the problem of relying on a single third party at authentication time by providing a "backup" that can be used if the primary email provider is currently unavailable.

2.5 Advanced Features

Currently, email-based authentication is predominantly relegated to automated password reestablishment. By extending and significantly improving this type of authentication, SAW

brings it to the forefront and demonstrates its innate power as a primary authentication mechanism and as a viable alternative to passwords at EBPR-enabled web sites.

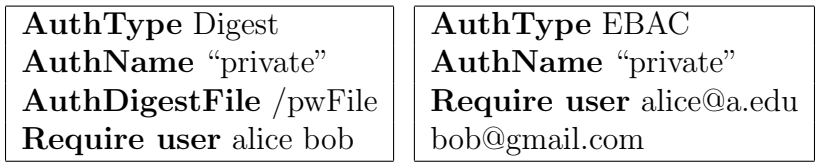
A major benefit of SAW is that email addresses, unlike digital certificates, are very easy to obtain and are useful in a variety of different applications in a myriad of security domains. It is also easier for a web site to trust almost any email provider because the amount of trust placed in them is much less than that of a traditional CA. Correspondingly, the implicit trust that email providers will only deliver an email message to its intended recipient allows them to be completely oblivious to the fact that they are performing the duties of an identity provider. This enables web sites to unilaterally deploy this authentication mechanism without modifying the email provider or entering into any legal or business relationships with them. Finally, in its simplest form, it requires no special user software.

2.5.1 Sharing and Collaboration

Users need a secure way to share files (e.g., photos, documents) with friends and collaborators. There is no easy-to-use, universal mechanism available to do this. To securely share files in password-based systems, each participant must have a username and password. The account creation process necessary for this mechanism is commonly a source of frustration.

Email messages are also commonly used to transfer and share files. We call this approach *informal sharing* because it does not require the recipient to possess a local account like the password-based approach. This method is easy, intuitive, and there are reasonable assurances that files will reach their destination. Emailing content is a *push-based* approach and due to the informal nature of the transaction, it is unlikely that the content will be sent with any cryptographic protection. Also, files sent via email can clog the recipient's mailbox.

SAW provides the foundation to build a secure, *pull-based* informal sharing mechanism. Since email addresses are necessarily unique they serve quite effectively as unique identifiers in access control lists. Using SAW it is possible to securely and efficiently use proof of email address ownership as an authenticator. Sharing, or allowing collaborative



(a) (b)

Figure 2.6: A traditional .htaccess file (a) restricts access to a directory to set of known users, e.g., alice and bob. These users, and their passwords, are defined in the AuthDigestFile. With EBAC (b), this password file is eliminated and users prove ownership of their email addresses using SAW to gain access to a directory. This removes the need to create/distribute user-specific passwords.

access, is accomplished by specifying the email address, or list of email addresses, that are permitted to access a resource. We call this approach *email-based access control* (EBAC).

This method of specifying permissions is simple and easy to use. It is also desirable for systems with discretionary access control (DAC), where users are allowed to specify access control for the content they control, e.g., photo sharing applications. These users can now securely share content, e.g., private photo albums, with their friends and family without requiring them to create accounts with the site or to manage long-lived links. This method is also well suited for use with the Apache Web server's .htaccess files, which permit user-specified, directory-level access control (see Figure 2.6).

We have implemented a module for Apache that provides this functionality and currently use it to protect a private collaborative wiki. In addition to enabling secure access control without having to create user accounts and distribute passwords, this module allows us to use SAW to protect access to an application without having to modify that application.

2.5.2 Delegation

Decentralized authentication systems (e.g., PKI-based approaches, SAW) lend themselves well to *server-based delegation* because delegates can be easily specified with public identifiers in a manner similar to the approach to sharing and collaboration described above. When servers take an active role in providing delegation, they have the potential to pro-

vide powerful, feature-rich delegation options. Nevertheless, server support for delegation is rare as it typically requires extensive modifications to existing applications to integrate an appropriate permissions model.

In general, delegation of authority in password-based systems is accomplished by sharing the password. To the eyes of the system, the delegate is the delegator.

There are several problems with this approach. First, once a password has been given away, it cannot be revoked, with a reasonable degree of certainty, without changing it. By giving the delegate the same password as the delegator, too much authority is delegated. The delegate now has the power to change the password and revoke access from the delegator, at least temporarily. The delegate can also share the password with others without the approval of the original delegator. Often times it is hard to remember who has received the delegated password and why.

A key benefit of this approach to delegation in password-based systems is that the site does not need to be aware of the delegation. We call this capability *client-based delegation.*

Client-based delegation in SAW leverages email forwarding rules. A delegate provides the delegator's email address to the web site and, through a forwarding rule at the delegator's email provider, AuthToken_{email} is sent to the delegate's email address. By removing this forwarding rule, a delegator immediately revokes permission for future authentications by a delegate. The delegator's list of forwarding rules provides an up-to-date record of delegations, which facilitates delegation management.

Client-based delegation in SAW does not require support from the web site. This type of delegation described works best when the user's email provider enforces the forwarding rules. Forwarding rules specified and processed in a local email client (e.g., Outlook Express) are problematic because the delegate must rely on the delegator to retrieve the email message from the server before the client forwards it to the delegate. Although web site support for delegation is not required, such support could provide more control over which permissions

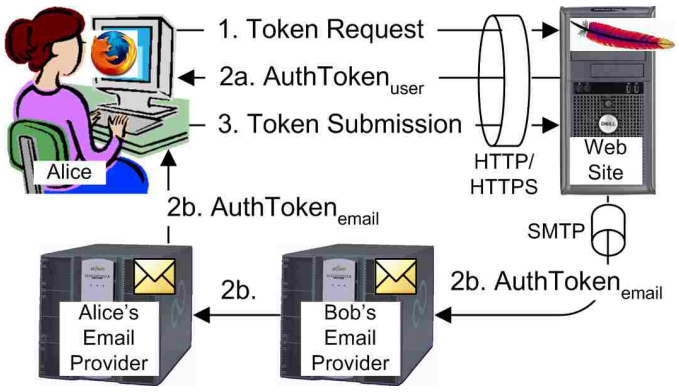


Figure 2.7: Delegation SAW using email forwarding rules. Bob delegates access to a web site to Alice by creating a forwarding rule with his email provider. To access the site, Alice submits Bob's email (1). She then receives AuthToken$_{user}$ directly from the site (2a). Once AuthToken$_{email}$ (2b) arrives at Bob's email provider, the forwarding rule sends this message to Alice. Once Alice receives the message, she completes the authentication process (3).

are delegated and enable users that lack access to email forwarding facilities at their email provider to delegate their authority.

Various types of delegation are achievable through this technique. The simplest form is *complete delegation*, where all token messages are forwarded to the account of the delegate. This can also be accomplished by specifying the delegate's email address as the delegator's secondary email address, but this method is not as desirable as the former because it requires modification to data stored at the web site and is not as scalable. In this model, it should be possible for the delegate to change the primary/secondary email accounts associated with the site. This form of delegation is helpful for users with multiple email addresses.

Multiple email addresses significantly improve the privacy of the user. For example, by using a free email address as opposed to a more identifying address, e.g., a business email, users avoid leaking information about themselves, e.g., their place of employment or real name. Using multiple addresses for authentication also limits the information that colluding web sites glean from pooling their information. Using the complete delegation model the token requests for multiple accounts funnel into a single one, enabling the user to authenticate to multiple email identities, while only having to retrieve authentication tokens from a single account, thus providing a form of multiple-identity single sign-on.

Selective delegation provides finer-grained delegation. In this model, only token messages that meet a certain criteria are forwarded. For example, only messages with tokens from site Y are forwarded to delegate X. This method allows the user to specify whether or not attempts to change the primary/secondary email address registered at a provider should be forwarded, thus preventing delegates from assuming complete control, unless explicitly allowed by the delegator. When selectively delegating authority it is important to intelligently forward tokens to the authorized user who actually initiated the request. This avoids superfluous messages to all involved.

A valid email address [70] allows for the addition of a sequence of words, called a *phrase*, before the actual email address. This has traditionally been used to create a pretty-printed form of a real name that corresponds to the email address, e.g., "William Henry Gates III" <bgates@microsoft .com>. By modifying the contents of this phrase, the intended target of the authentication mechanism is easily identifiable. For example, suppose Linus Torvalds has been authorized by Bill Gates to access a particular web site. When Linus supplies the value:

"Delegate to: torvalds@osdl.org"<bgates@microsoft.com>

as his email address, Bill's email filter is able to trivially determine whether a particular authentication message is meant for him or for Linus.

This method also facilitates the creation of groups of users. Although this approach has the potential to remove the anonymity of group members, it also avoids everyone in the group from getting the authentication messages from everyone else in the group. It must be noted that this solution also allows web sites to be aware that delegation is occurring and possibly act on it. This may or may not be desirable to the user.

One-time delegation is a subset of selective delegation. The delegator begins the authentication process and collects the required authentication tokens, but instead of submitting them to the web site, the delegator supplies them to the delegate. Due to the short lifespan of the tokens, this process is very time-sensitive.

2.5.3 Client-side Auditing

Since current password authentication only involves the web site and the entity attempting to authenticate, which may or may not be the user, there is no means, without the support of the web site, to provide audit information to the user concerning authentication attempts and failures.

In SAW, it is possible for the user to audit the authentication process without any support from the web site because any such attempt must necessarily pass through the user's email account. This auditing capability remains available, even when authority is delegated using client-based delegation. By saving a copy of the messages that are forwarded, a client-side audit trail is established. Alternatively, with minimal modification to the site, an audit message (e.g., an authentication message with the AuthToken removed) sent to a user's secondary email address also creates a client-side audit trail. If a web site has built-in delegation capabilities, the client-side auditing of SAW may not work unless the site notifies the user each time a delegate accesses the system.

2.5.4 Active Impersonation Countermeasures and Privacy-Enhancing Features

This section describes some optional mechanisms that prevent active impersonations by email providers and add *confidentiality* and *anonymity* to the email messages. These mechanisms are not mandatory because we believe that the increased overhead and complexity they add is too costly to address risks that the large number of EBPR-enabled sites have already demonstrated are manageable.

First, it must be noted that in order to avoid detection an active attacker must intercept and remove email messages destined for the user. Otherwise, a client-side auditing mechanism would detect this breach. This is arguably more difficult than simply passively observing this link.

To prevent active impersonations, SAW can concurrently involve multiple email accounts in a single authentication. The secret splitting scheme used to create the authentication tokens is easily extended to n email accounts as follows:

$$AuthToken_{user} = AuthToken_{email_1} \oplus \cdots \oplus AuthToken_{email_n} \oplus AuthToken_{complete}$$

If each $\text{AuthToken}_{email_i}$ is sent to a different email account, active impersonations require the eavesdropping of, or the collusion of, all the providers of these accounts. Choosing email providers in unrelated domains greatly decreases the possibility of this occurring.

Also, by using a threshold secret sharing scheme [73], it is possible to split the secret such that the user only has to retrieve m of n messages to successfully authenticate. This has the potential to improve latency when one or more email providers are busy or unavailable. Nonetheless, splitting AuthToken_{email} between multiple accounts diminishes the single sign-on potential of SAW.

Note that the location where the actual passive eavesdropping occurs affects the payoff to an attacker. If an attacker is observing the incoming traffic to a particular email provider, she is easily thwarted by using the multiple email account scheme described above. On the other hand, if the attacker is listening to all outgoing email traffic of a particular web site the multiple email account scheme would not be effective, but the attacker can only impersonate the user at that particular site, not at the other sites that can be accessed by the user.

To optionally provide confidentiality and anonymity to the delivery of AuthToken_{email}, the message containing it is encrypted and delivered via an anonymous remailer. Users decrypt this message using the decryption key that accompanies AuthToken_{user}. By encrypting this message and hiding its origin, users prevent their email providers from gleaning information about an authentication at the expense of increased latency.

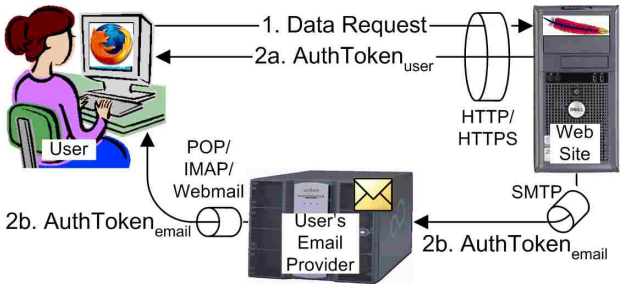


Figure 2.8: The one-round SAW protocol. If a user-supplied email address is authorized to obtain the data requested (1), a web site creates and distributes two authentication tokens. AuthToken$_{user}$ (2a) is sent directly to the user while AuthToken$_{email}$ (2b) is emailed. Both tokens must be combined to obtain the requested resource.

2.5.5 One-Round SAW

Step 3 of SAW, *Token Submission*, is eliminated in *one-round* SAW by leveraging SAW's existing token delivery mechanism (Step 2) to provide both authentication and data delivery. One-round SAW (see Figure 2.8) operates as follows:

Step-1 The user submits her email address and her request for *data* to the web site in the *Token Request*. This request should be sent over HTTPS.

Step-2a The web site returns $AuthToken_{user}$ (a randomly-generated value).

Step-2b If an address is authorized the web site emails $AuthToken_{email}$, computed as follows:

$$AuthToken_{email} = data \oplus AuthToken_{user}$$

The user can then combine $AuthToken_{user}$ and $AuthToken_{email}$ to recreate *data*. If an address is not authorized $AuthToken_{email}$ is set to a randomly-generated value.

Since only authentic users can obtain $AuthToken_{email}$ and reconstruct it, only they will obtain the item. One-round SAW increases user privacy because the server never learns the result of the authentication; the server is unable to prove whether the requester is a legitimate user or an impostor.

Large Resource Distribution When using one-round SAW, a server must take into account the size of the requested item. The token splitting used by SAW creates shares the same size of the secret it splits. If items are small, the impact is negligible. For large items, it is advised to encrypt the item, split the encryption key, and deliver the encrypted item with only one of the encryption key shares as follows:

Step-1 Unchanged.

Step-2a The web site returns $AuthToken_{user}$ (a randomly-generated value) and $E_k(data)$, which is the requested *data* encrypted with key k.

Step-2b If an address is authorized the web site emails $AuthToken_{email}$, computed as follows:

$$AuthToken_{email} = k \oplus AuthToken_{user}$$

The user can then combine $AuthToken_{user}$ and $AuthToken_{email}$ to recreate k and use it to decrypt *data*. If an address is not authorized $AuthToken_{email}$ is set to a randomly-generated value.

One-round SAW is an application of one-time pad encryption [73]. The "key" is $AuthToken_{user}$ and the "ciphertext" is $AuthToken_{email}$. Key distribution is a major difficulty of one-time pad encryption because it requires that the key and the ciphtertext be the same size. In other words, if you can securely distribute the key, why not save some effort and deliver the ciphertext instead. One-round SAW does not assume that there is a secure method to communicate a key to the user, rather it relies on the premise that both channels must be observable by the user, and not an attacker, to provide its security.

Hidden Credentials [15] also provides the ability for servers to distribute encrypted data with the assurance that only authorized parties will be able to decrypt it. Hidden credentials requires a third party to issue asymmetric decryption keys. In one-round SAW symmetric encryption keys are generated by the service provider, thus eliminating the need for user-specific asymmetric key pairs.

2.6 Overcoming Email Obstacles

Latency

Although typical email message delivery is nearly instantaneous, SAW is not immune to high traffic conditions and other factors (e.g., virus/spam filters, archiving) that delay message delivery. Some corporate email systems support the designation of high priority messages to reduce latency. When such support is unavailable, using instant messaging in lieu of email is an effective method to avoid the potentially high latency of email. While SAW may not suitable for every email system, our initial performance results demonstrate feasibility of SAW as a primary means of authentications, even on a highly loaded email system like Gmail.

Privacy

Section 2.5.4 enumerates the privacy enhancing features available to SAW. These features, as well as the protections provided by the protocol itself, allow SAW to detect and prevent threats to a user's privacy, even when the link between the web site and the email provider has no cryptographic protection.

Convenience

While it is difficult to compete with something as pervasive and embedded as password-based authentication, SAW provides significant benefits to justify the switch and is designed to minimize the cost of such a transition. Most notably, no modification or changes are required to email providers. No specialized client-side software is required, although the convenience and benefits of this system are greatly enhanced through client-side automation. The only entity that requires modification is the web site.

Web sites already have the necessary hooks to both the local applications and the email system to implement SAW. Transitioning to SAW requires a repurposing of services

that are already in use. The most significant modification to this module is the addition of the ability to create authentication tokens, which is purposely straightforward and computationally light-weight.

Pervasive user training on email-based authentication began many years ago with the adoption of EBPR. Users are already well aware of the utility and convenience their email accounts hold for authentication. SAW exploits this foundation to extend and improve this form of authentication.

Reliance on a third party

The reliance of SAW on email providers is mitigated by the distributed and decentralized nature of email providers. The use of a secondary email address alleviates the problems caused by temporary lapses in availability of a user's primary email provider. Although an email provider could selectively drop or refuse authentication messages, this issue is between the email provider and the user and is outside the scope of SAW.

2.7 Related Work

PKI-based systems overcome many of the problems associated with passwords. By specifying trusted CAs, and using certificates issued by those CAs for client authentication, systems like client certificates in TLS have the potential to overcome the weaknesses of password-based systems. Unfortunately the adoption of PKI-based systems by businesses and the general public has been slow. PKI systems have burdensome configuration and usage requirements [35]. High costs, the requisite user and administrator training in a complex technology, and the difficulties of trusted roots and cross-certification also makes PKI hard to adopt [32]. Ellison and Schneier [27] further illustrate the risks of PKI and "trusted" CAs. Overall, PKI-based systems are too heavy-weight for most web sites and their users.

Centralized single sign-on is seen by many as the panacea to the rising password woes. Systems like Liberty [54] and Shibboleth [74] use specialized identity servers that provide

authentication and the ability to assert additional attributes about their users. Though promising for future applications, the legal and business relationships required to implement these systems prohibit their widespread adoption. At this stage these systems are too heavy weight to replace passwords as the preferred authentication system. SAW does not require any legal/business agreements between the identity provider and the web site. Such agreements may improve the benefits of SAW, but are not necessary to deploy and use this system.

There are a variety of URL-based authentication systems. These systems, such as OpenID [62], Light-Weight Identity (LID) [55], and Simple eXtensible Identity Protocol (SXIP) [77], uniquely identify a user using a URL. Based on this URL, web sites communicate with a specialized identity provider to verify user identity. Although these systems provide a decentralized authentication mechanism, they require the creation of specialized identity providers and new identifiers (i.e., URLs) that are not as intuitive, recognizable, or as widely used as email addresses and other personal messaging identifiers. SAW is a simpler alternative to these systems when the authentication does not require any attribute information exchange. Also, SAW can serve as the authentication method for the identity provider, avoiding the need for yet another password.

Wordpress [85], a popular web log platform, has a third-party plug-in, Comment Authorization [23], that is designed to reduce comment spam by requiring self-moderation of comments. Once a comment is posted, an authorization link is sent to the author's purported email address. By clicking the link, the owner of the email address authorizes the posting of the comment. This method of self-authorization presents a novel idea for distributed authorization, but it is not a general purpose authentication mechanism like SAW.

Our primary research focus for the past decade has been trust negotiation [84], an approach to authentication in open systems that relies on third party attribute assertions contained in digital credentials. Trust negotiation has a significant deployment hurdle. Users typically do not possess digital credentials because most web sites do not accept them; most

web sites do not accept certificates because users do not possess them. It is unclear when, or if, this chicken and egg problem will be overcome. SAW grew out of our search for simpler approaches with the potential for an immediate impact on authentication challenges in open systems.

2.8 Conclusions

SAW is a simple concept. It builds on EBPR, which has already proved its utility for authenticating users, and improves it by thwarting passive attacks and significantly raising the bar for active attacks. These enhancements make SAW a viable alternative for passwords at a significant number of web sites. SAW has the potential to thrive because it does not require universal acceptance, modification of email providers, nor significant changes to existing web site infrastructure.

SAW is an important step towards simplifying authentication and making it more convenient. It relieves both web sites and users from having to establish and manage passwords by off-loading user authentication to email providers. Although SAW does not eliminate passwords completely (users still have to authenticate to their email providers), it should greatly reduce the number of passwords users need to remember. In addition, this system provides single sign-on to all SAW-enabled sites, exploits client-side automation to speed up the login process, and reduces the attack surface for phishing and social engineering attacks.

SAW addresses all the obstacles enumerated in Section 2.2.1 that would impede the adoption and utility of this new web site login approach. It also provides several advanced features not usually available to password-based systems. These include: 1) Leveraging email addresses and proof-of-address-ownership to facilitate sharing and collaboration; 2) Using email forwarding rules to allow intuitive delegation and revocation of authority; and 3) Exploiting the fact that all authentication attempts pass through a user's email account to provide client-side auditing.

SAW is a simple concept, but therein lies its strength. It is easy to understand and to implement. Its benefits are significant. The risks of its use are clear and have already proven to be manageable.

2.8.1 Future Work

As mentioned earlier, SAW provides personal messaging-based authentication and easily translates from email to other mediums, e.g., instant and text messaging. We are currently exploring other personal messaging systems and the possibilities of a hybrid combination of these mediums.

For example, a cell phone with email or text-messaging capabilities has the potential to mitigate the risks of using SAW from an untrusted machine, such as in a cyber café or public library. By retrieving AuthToken_{email} from the phone, or having the email provider forward it to the phone, the untrusted device will never be able to capture the email account password. To be effective, a means to conveniently transfer the token from the phone to the computer must be devised, e.g., transforming the token into a more human-friendly form or using a limited-range wireless protocol.

SAW primes the pump for attribute-based authentication because some email providers (e.g., universities and businesses) are authoritative sources for certain user attributes. In fact, the mere possession of an account at a specific provider is a desirable attribute for specifying access control. Email aliases are one technique to assert these attributes without modifying the email provider. For example, Alice has the account alice@cs.someschool.edu. This email address may be used to learn that Alice is a computer science student or faculty member at someschool. By creating the alias: alice@phd.grad.cs.someschool.edu, more attributes are added.

Alternatively, by replacing the delegation string described in Section 2.5.2 with an attribute(s) request, the email provider could be modified to deliver messages only if the recipient has specific attributes. In either of these approaches, the trustworthiness of the

attributes being asserted depends on the relationship between the email provider and the web site.

Since a variety of applications, not just web site logins, benefit from SAW, it would be convenient for users to have a centralized, client-side location to retrieve and distribute AuthToken_{email}. A local email client is an excellent location to host such a service. Through an extension or plug-in this email client (e.g., Mozilla Thunderbird and Outlook) would provide a single, intuitive location to manage local access to email accounts. It also provides a location to enforce fine-grained permissions on the programs that are allowed to access the authentication information sent from a particular web site.

A variety of interesting and exciting possibilities become a reality when email providers take an active role in the authentication process. Specifically, we are interested in making SAW feasible even when users cannot directly contact their email providers, e.g., logins for wireless internet access.

We are also currently preparing a large scale user study to evaluate the usability of SAW. This study will focus on authenticated blog access and file sharing.

Acknowledgments. This research was supported by funding from the National Science Foundation under grant no. CCR-0325951, prime cooperative agreement no. IIS-0331707, and The Regents of the University of California. We thank Andrew Harding for his work on the browser extensions, Daniel Wille for his work on the WordPress plug-in, and Reed Abbott for his work on the Apache authentication module.

Chapter 3

Wireless Authentication using Remote Passwords (WARP)

A. Harding, T. W. van der Horst, and K. E. Seamons. Wireless Authentication using Remote Passwords. *1st ACM Conference on Wireless Network Security (WiSec)*, Alexandria, VA, March 2008.

Additional Content: The published version is augmented here with Section 3.1.1 (Motivating Scenarios) and graphical versions of the figures. These were present in the original submission, but had to be removed/condensed due to length restrictions.

Abstract

Current wireless authentication mechanisms typically rely on inflexible shared secrets or a heavyweight public-key infrastructure with user-specific digital certificates and, as such, lack general support for environments with dynamic user bases where guest access is frequent. Simple Authentication for the Web (SAW) facilitates dynamic user bases in the context of web site logins by enabling users to authenticate to personal messaging identifiers (e.g., email addresses, IM handles, cell phone numbers). SAW, however, is ill-suited for wireless authentication because, in most cases, it is dependent on client-side Internet connectivity. Wireless Authentication using Remote Passwords (WARP) overcomes this constraint by building a hybrid protocol that combines the principles of SAW authentication with the Secure Remote Password (SRP) protocol.

3.1 Introduction

Wireless networks provide an attractive means for network access as they enable convenient roaming and device deployability without the burden of network cables and port accessibility. Although convenient, wireless communications are subject to passive eavesdropping, active injection, and modification attacks. For these reasons, it is important that wireless networks provide secure authentication to prevent unauthorized access to network resources and provide confidentiality and integrity to transmitted information.

Current wireless authentication mechanisms are usually based on user-specific certificates (PKI), global passphrases, or username/password pairs (see Section 3.7). These methods are either too heavy or inflexible and lack general support for environments with dynamic user bases, such as corporations or universities where guest access is frequent.

Wireless Authentication using Remote Passwords (WARP) augments the Secure Remote Password (SRP) [87] protocol using concepts from Simple Authentication for the Web (SAW) [81]. By proving ownership of an authorized personal messaging identifier (e.g., email address, IM handle, cell phone number), WARP enables users to authenticate without preestablished secrets or the heavy cost and inconvenience of user-specific certificates. The burden of wireless access control in dynamic user bases is dramatically reduced as authentication of these globally unique identifiers is performed by trusted third parties.

3.1.1 Motivating Scenarios

Home User Johnny, a security minded user, is anxious to set up a wireless network at home. Johnny is wary of current "home" solutions that use global passphrases. He is a member of a linux users group and often holds meetings in his residence. Johnny dislikes having to share his passphrase with his guests. He also does not have the resources or time to setup and maintain the infrastructure required for EAP-TLS[4] or MSCHAPv2[89]. He wants to provide secure wireless connectivity with the least amount of inconvenience for himself and his guests.

Corporate Environment Various employees from another company are visiting daily. Until this point, the company has tied the wireless authentication server to the user accounts used for regular computer and network access. Because of the tight integration, IT staff have had no extra burden managing wireless access; creation/deletion of the user accounts for a new/terminated employee automatically grants/revokes wireless access privileges. But now, without access to the collaborating company's user accounts, they have been forced to add temporary users every time someone visits.

Conference Committee A week long conference is being held at a university. It is desired that all participants have wireless connectivity during the conference. Obtaining user accounts for each participant through the IT department is time consuming and the task of distributing usernames and passwords is unwieldy. Certificate-based methods are also undesirable; the burden of obtaining a certificate is considered a waste of the attendees' time. Additionally, the conference staff have enough preparations without adding the hassle of configuring the wireless network. Forgotten username/password pairs or improperly configured certificates could spell disaster for the staff on the first day of the conference and they want to avoid wasting time that could otherwise be used for gainful participation in the conference. In short, they want to spend minimal time managing the wireless connectivity and reduce the number of steps participants need to take (and potentially do wrong) in order to authenticate.

3.2 Background

Simple Authentication for the Web (SAW) [81] enables decentralized authentication of globally unique personal messaging identifiers (e.g., email addresses, IM handles, cell phone numbers). This approach is ideal for systems with dynamic user bases because no shared secrets (e.g., passwords) are required between clients and relying parties (e.g., web sites). In-

stead, SAW leverages unmodified personal messaging providers (e.g., email, text and instant message services) to act as identity providers to relying parties.

SAW builds on the same basic technique employed by the "Forgot your password?" link common to many web sites; users must retrieve personal messages (e.g., email, text and instant messages) sent to them by the relying party through their messaging provider. Relying parties depend on providers to deliver messages only to authenticated recipients. As WARP represents a significant departure from SAW, a detailed protocol overview is omitted.

WARP builds on the following principles of a SAW authentication:

1. Reuse existing identifiers and authenticators.
2. Tightly couple identifiers and identity providers, e.g., email addresses specify the locations of their respective mail servers.
3. Authentication requires that users obtain two tokens known to the relying party. The first token is given to the initiator of an authentication, while the second is only obtained after a successful authentication to the identity provider.

3.2.1 The Chicken and the Egg

As many personal messaging providers (e.g., email, instant messaging) rely on client-side Internet connectivity for message retrieval, an interesting chicken and egg problem must be overcome to adapt SAW for wireless authentication: How do clients communicate with their personal messaging providers when the reason they are authenticating in the first place is to obtain network and Internet connectivity? Four potential solutions have been identified:

Temporary Connectivity A user has limited time to access the required personal messaging resources before his connectivity is terminated. This approach carries increased liability and is undesirable as it allows anyone to have temporary access to network resources; access that could be used to launch attacks on or from the local network.

Filtered Connectivity Attempt to allow clients to exchange traffic with only their personal messaging providers. For example, clients obtain IP-level connectivity to a restricted network with limited Internet access. The client is switched to the regular network after successfully authenticating.

Protection against abuse is complex and potentially impossible. Sophisticated filters could restrict traffic to authorized personal messaging protocols, but filtering is impossible if encryption is used. Data-limiting caps could be used but would be difficult to fine-tune. The complexity of the safeguards needed to decrease the liability of this approach make it largely unacceptable.

Out-of-band Message Delivery The chicken and egg problem does not exist when out-of-band channels are employed to deliver messages to users, e.g., text messages sent through a cellular providers' network (SMS). This approach does not work in locations without cellular coverage and requires SMS-capable devices.

Surrogate Authentication A surrogate approach requires the party to which users are authenticating to relay small messages to their personal messaging providers on their behalf. In this approach an IP-level of connectivity is not required; the Extensible Authentication Protocol (EAP) [3] carries the authentication traffic. Since the authenticator, not users, directly communicates with the personal messaging provider it has greater control and can limit unauthorized data transfer.

As users are unlikely to trust authenticators to log in as them to their personal messaging providers, this approach must provide assurances that user login credentials cannot be stolen or misused by the authenticator.

WARP is a surrogate solution that provides strong protections to user credentials.

3.3 WARP

In this paper, user (**U**) refers to the user and its wireless supplicant, relying party (**RP**) refers to the wireless access point (and potentially an associated authentication server, e.g., RADIUS server) and identity provider (**IDP**) refers to the personal messaging provider.

In general, WARP should be both convenient and secure. Specifically, it should:

- Preserve the three principles of SAW (see Section 3.2)
- Authenticate **U**
- Provide confidentiality and integrity to the session established between **U** and **RP**; and to the communications between **U** and **IDP**

To satisfy the first principle of SAW, WARP also reuses existing personal messaging identifiers and authenticators, e.g., email addresses and email account passwords.

With regard to the second principle, the location of **IDP** must be learned from/based on the user's identifier. For example, if an email address is used **IDP** would be hosted on a known port of the domain specified by the identifier, or specified in DNS similar to the `MX` (mail server) entry. Note that this requires **IDP**s to support this protocol; a significant departure from SAW, which leverages unmodified providers.

The third principle is the most challenging to fulfill. The purpose of the two token scheme is to allow **IDP** to assist **U** in the creation of an encrypted session with **RP** without **IDP** being able to compromise that session. It is simple for **RP** to deliver a token to each **U** and **IDP**, and relay small messages between the two. The difficultly arises by requiring **IDP** to confidentially deliver its token to only an authenticated **U**.

Conventional wisdom dictates that the simpliest way to accomplish this is to use a secret key known only to **U** and **IDP**. WARP's solution to this problem is to build on the Secure Remote Password (SRP) protocol, which is a password authenticated key agreement scheme that enables the establishment of a strong ephemeral session key from a potentially weak password.

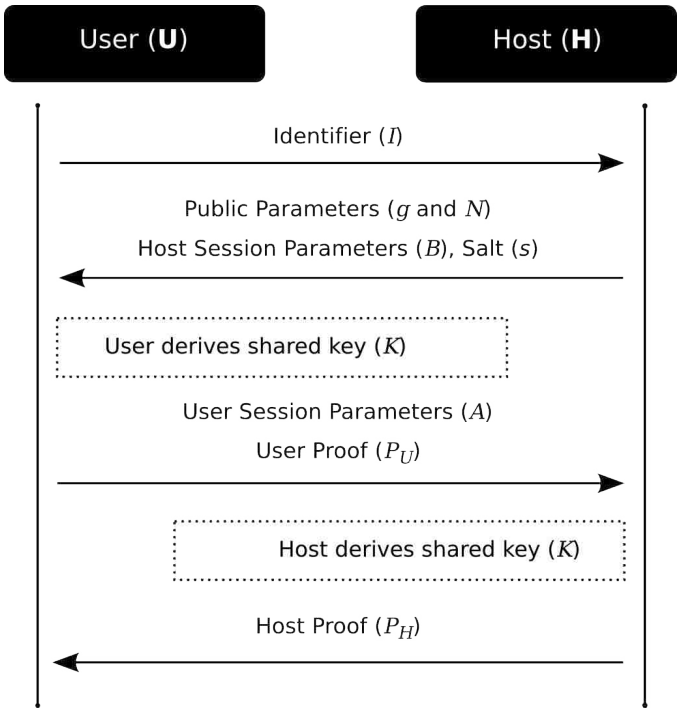


Figure 3.1: SRP protocol messages.

The remainder of this section is organized as follows. Section 3.3.1 gives an overview of SRP. Section 3.3.2 documents sSRP, which contains the SAW-based additions to SRP. Section 3.3.3 shows how sSRP is used for wireless authentication.

3.3.1 Secure Remote Password

Secure Remote Password (SRP) [87] is a protocol designed to provide password-based mutual authentication between a user (**U**) and a host (**H**), and establish an ephemeral session key. SRP reveals no information to eavesdroppers during authentication that can be used to mount an offline attack against the password. It is also resilient against well-known passive and active attacks. The host does not store passwords for each identifier in plaintext but instead a unique salt and verifier. The salt is used with the plaintext password to generate the verifier. The verifier is not password-equivalent and cannot be used to impersonate the user.

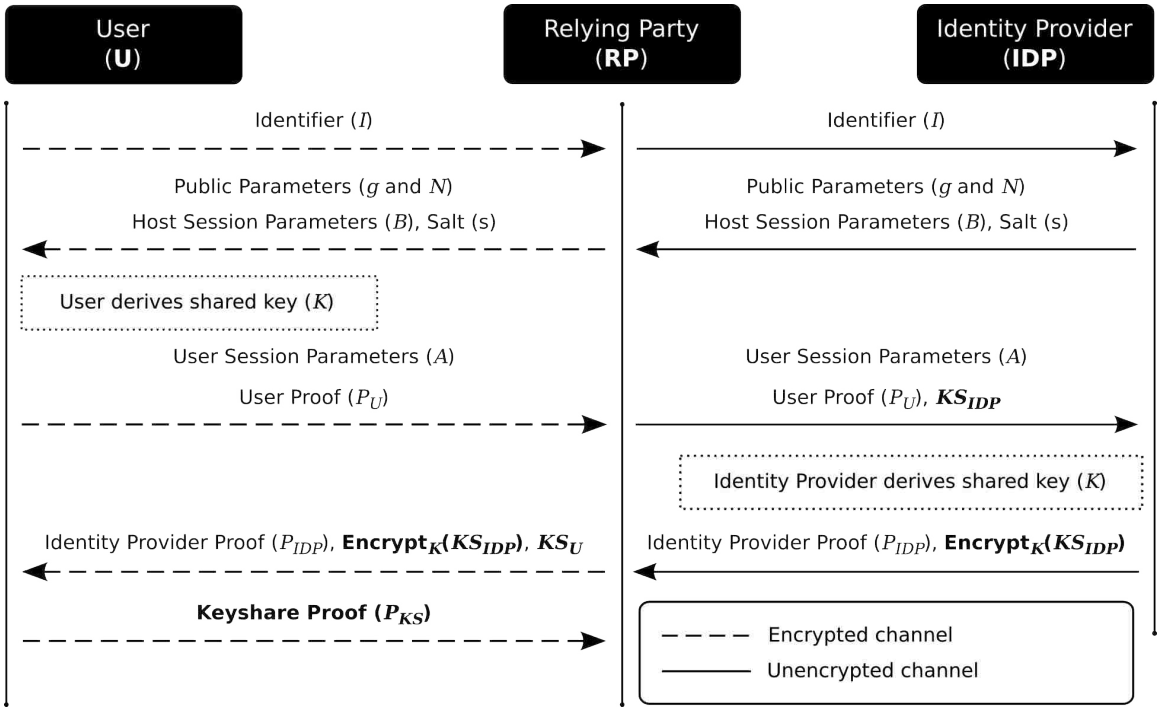


Figure 3.2: sSRP protocol outline (augmentations to SRP in bold).

As WARP treats SRP as a black box that results in a shared key between **U** and **H**, only a high level overview of SRP is given here. Figure 3.1 shows a condensed version of the SRP protocol currently being proposed as an extension [78] to TLS that operates as follows:

1. **U** sends its identifier (I) to **H**.
2. **H** looks up the public group parameters (g and N) for I as well as the user's salt and verifier (s and v). **H** computes B (its ephemeral session parameter) and returns it, along with g, N, and s to **U**.
3. **U** generates A (its ephemeral session parameter). Next, **U** computes the session key K using the values from **H**. **U** then sends A and a proof (P_U) of K to **H**.
4. **H** computes the session key K, verifies P_U, and sends its own proof (P_H) to **U**.

3.3.2 Surrogate SRP (sSRP)

WARP augments SRP with principles of SAW to create the Surrogate Secure Remote Password (sSRP). In sSRP, an identity provider (**IDP**) assumes the role of the SRP host. This protocol introduces a relying party (**RP**) through which messages between **U** and **IDP** are relayed. Like SRP, the protocol is initiated by the user (**U**).

sSRP treats the internals of SRP as a black box and only relies on SRP to create password-based ephemeral session keys between **U** and **IDP**. sSRP simply adds several message elements that are unrelated to the original, unmodified SRP messages. At a high level, we're piggybacking SAW token distribution over unmodified SRP. Figure 3.2 shows the sSRP protocol. Here is a description:

1. **U** submits its identifier (I) to **RP**, which authorizes the identifier and forwards it to **IDP**.
2. **IDP** looks up the public group parameters (g and N) for I as well as the user's salt and verifier (s and v). **IDP** computes B (its ephemeral session parameter) and returns it, along with g, N, and s to **U** via **RP**.
3. **U** generates its SRP session parameter (A), computes the session key K. and then sends A and a proof (P_U) of K to **RP**. **RP** generates a random value (KS) and splits it into two keyshares: KS_{IDP} and KS_U. **RP** appends KS_{IDP} to **U**'s message and sends it to **IDP**.
4. **IDP** computes the session key K, verifies P_U, and then encrypts KS_{IDP} using K. **IDP** then computes its own proof (P_H) and sends it, along with the encrypted KS_{IDP} to **RP**. **RP** appends KS_U before forwarding the message on to **U**.
5. **U** performs the following:
   (a) Decrypts KS_{IDP} using K.
   (b) Recreates KS using KS_{IDP} and KS_U

(c) Creates the keyshare proof (P_{KS}).

$P_{KS} = H(I \| g \| N \| s \| B \| A \| P_U \| P_{IDP} \| KS)$.

(d) Sends P_{KS} to **RP**.

sSRP leaves both **U** and **RP** with shared key KS, which is used as keying material to encrypt future transmissions. This key is different than the SRP shared key K.

In order to protect against eavesdropping and impersonation attacks, the link between **U** and **RP** must provide confidentiality, integrity, and authentication of **RP**. This protects the transmission of KS_U as it is sent to the user (see Section 3.5.1).

Upon first inspection, it may seem like KS_U provides no additional assurances. KS_U serves two purposes: 1) Prevents IDP from having the full keying material (IDP never sees KS_U); and 2) Makes KS_{IDP} by itself worthless, as an attacker needs both keyshares to either impersonate the user or decrypt post-authentication transmissions.

sSRP enables proof of personal messaging identifier ownership without employing the personal messaging medium itself. Ubiquitously deployed, this service provides a mechanism useful not only to WARP, but also to SAW and many other mechanisms that rely on proof of identifier ownership through password-based authentication.

Using other identity providers Although this description of sSRP used personal messaging identifiers, e.g., email addresses or instant messaging handles, sSRP can be used with any password-based identity provider where an sSRP service can be deployed. This means that users could authenticate to a wireless network using identifiers such as OpenID or Unix logins.

3.3.3 Employing sSRP in WARP

WARP is an incarnation of sSRP for wireless authentication. In WARP, the wireless supplicant **S** takes on the role of the user and the authentication server (**AS**) that of the relying party.

EAP-WARP, a new EAP method, has been created to support WARP. EAP-WARP encapsulates the sSRP protocol as it travels between the supplicant and the authentication server. EAP-WARP works as follows:

1. **S** and **AS** use EAP-TTLS [30] to authenticate **AS** and provide confidentiality and integrity for the link.
2. **S**, **AS**, and **IDP** perform sSRP. Upon submission of the sSRP keyshare proof P_{KS}, **S** has proven ownership of an authorized identifier to **AS**.
3. **AS** sends an EAP-Success message back to **S**.
4. **S** and **AS** use KS to derive the EAP Master Session Key (MSK), which provides confidentiality and integrity for the subsequent wireless session.

3.4 Implementation

Software to support wireless authentication using WARP has been developed and will soon be available.

`libssrp` is a general purpose library written in C that provides the functionality needed to conduct sSRP authentication. It is meant to be used by applications that supply their own transport functionality.

The current version of `libssrp` relies on `OpenSSL` [80] for its cryptographic primitives and arbitrary precision integers. `libssrp`, by default, relies on an SRP-compatible password file populated with salts and verifiers generated from plaintext passwords. An API can alternatively be used to allow flexible retrieval of salts and verifiers. An argument to stay with the default configuration is given in Section 3.6.

The `wpa_supplicant` [58] open-source package has been extended to support EAP-WARP. The extension consist of two files: 1) One C source file to provide EAP-WARP support; and 2) A patch file that modifies `wpa_supplicant` to include the extension. The extension is less than 400 lines of code.

This simple extension provides a layer that extracts sSRP packet data and provides it to `libssrp`. sSRP packet data returned from `libssrp` is inserted into an EAP packet that is returned to `wpa_supplicant`. The extension also exports the EAP MSK, derived from KS, to `wpa_supplicant`.

`FreeRADIUS` [79], an open-source RADIUS server, has also been extended with EAP-WARP support. The extension is around 800 lines of C code and comments, and like the extension for `wpa_supplicant`, provides extraction and insertion of sSRP messages to and from EAP packets, as well as exporting keying material. The `libssrp` library is again used to provide the bulk of the functionality.

An incarnation of the sSRP service has been written using the `libssrp` library and provides sSRP over TCP/IP. The service is written in C, can daemonize, and supports logging to syslog.

3.5 Threat Analysis

This section contains a threat analysis of WARP and the underlying sSRP protocol to enable proper risk evaluation by those deploying WARP.

SRP and SAW are the parent protocols of sSRP. SRP is already resilient to passive eavesdropping and active modification or impersonation attacks. sSRP purposefully inserts a middle party in between the user and identity provider in SRP. This creates two channels for attackers to target. Section 3.5.1 discusses threats to the channel in between **U** and **RP**. Threats on the channel between **RP** and **IDP** is provided in Section 3.5.2. Section 3.5.3 evaluates threats when both channels are available to an attacker.

A discussion regarding impersonation by **IDP** is provided in 3.5.4. Section 3.5.5 theorizes how WARP could be used to mount a denial-of-service attack. Section 3.5.6 concludes the threat analysis by discussing how sSRP could be abused as a covert channel and how such an attack is limited.

3.5.1 Channel between U and RP

In WARP, **U** communicates directly with only **RP**. Even if **U** connects to a malicious **RP**, this entity cannot leverage these communications to impersonate **U** at a legitimate **RP**.

The channel between **U** and **RP** must provide confidentiality, integrity, and authentication of **RP** in order to protect the transmission of KS_U. A man-in-the-middle attack is still possible over this channel depending on how authentication of the relying party is implemented. It is therefore necessary for sSRP to provide protection if the user connects to an attacker instead of the intended relying party.

For example, WARP uses EAP-TTLS to provide security on this channel and authenticate **AS**. In order to prevent man-in-the-middle attacks, the supplicant would need to verify **AS**'s certificate before accepting the connection. A careless supplicant choosing not to verify the certificate would allow a man in the middle to place himself between the supplicant and **AS**. The supplicant would establish a TLS session with the attacker, who would then establish a TLS session with **AS**. The supplicant would be oblivious to such an attack. All traffic would now flow through the attacker. The attacker can now observe KS_U.

KS_U, by itself, is useless since, without knowledge of KS_{IDP}, the attacker is unable to derive KS. KS_{IDP} is encrypted with the SRP session key K before travelling across this channel to prevent the attacker from obtaining it. The attacker, who does not know K, is unable to decrypt the keyshare and subsequently unable to impersonate the user.

A man-in-the-middle attack could allow an attacker to intercept P_{KS} before it arrived at the relying party. The proof could then be sent by the attacker to impersonate the user. This is fruitless, however, as knowledge of KS is required to further communicate with **RP**. In WARP terms, this means that the attacker would not have the correct EAP keying material to export and would therefore be incapable of communicating further with the access point.

3.5.2 Channel between RP and IDP

The user relies on **RP** to connect to the correct identity provider. Even if **RP** connects to a malicious identity provider, sSRP prevents that provider from learning anything about the user's password.

The channel that is established between **RP** and **IDP** is insecure. SRP parameters are protected by the built-in protections provided by SRP. KS_{IDP} is sent across this channel twice in this protocol: 1) In the clear on its way to the IDP; and 2) Encrypted with the shared SRP session key K as it is sent back to **U**. **AS** has knowledge of KS_{IDP} and could attempt to brute-force K by encrypting KS_{IDP} with all possible values for K. Even if **AS** is able to obtain K, K is not helpful in discovering the password [88].

3.5.3 Both Channels

If Mallory, who can observe the channel between **U** and **RP**, colludes with Eve, who can observe the channel between **RP** and **IDP**, then a one-time impersonation of the user is possible. To initiate this attack, Eve passively observes KS_{IDP} as it is sent from **RP** to **IDP**. Mallory likewise obtains KS_U as described in Section 3.5.1. If Eve is able to communicate KS_{IDP} to Mallory, then Mallory can construct KS and impersonate the user. This impersonation is limited to a single authentication because the keyshares are single-use and short-lived.

Although this one-time impersonation attack is complex and unlikely, it can be avoided. Encrypting KS_{IDP} with **IDP**'s public key before it is sent to **IDP** provides confidentiality, since only **IDP** is be able to decrypt it. An alternative approach would be to encrypt the entire channel between **RP** and **IDP**, but this creates unnecessary overhead as the remainder of the message elements are already protected.

Figure 3.3 shows a slight modification to the interaction between **RP** and **IDP** in sSRP, which prevents this one-time impersonation attack. The modifications are as follows (other interactions remain unchanged):

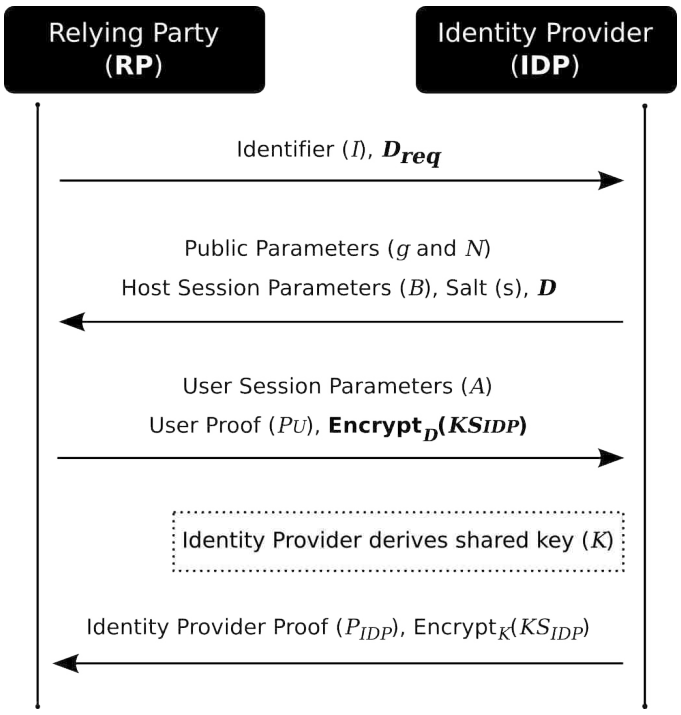


Figure 3.3: One-time impersonation resistant sSRP (augmentations shown in bold).

1. **RP** appends the optional public key request, D_{req}, to A as it travels to **IDP**.
2. **IDP** fills the request by sending his public key certificate along with B and s to **U** through **RP**. The certificate contains his public key (D).
3. **RP**: verifies and strips off the certificate.
4. **RP**: encrypts KS_{IDP} with D before it is sent to **IDP**.

The encrypted KS_{IDP} is only decryptable by **IDP** and is therefore useless to an eavesdropper.

3.5.4 Impersonation By IDP

Identity providers can impersonate any of their users. If an attacker or malicious insider takes control of the sSRP service hosted by the provider, they can impersonate users to gain wireless connectivity. For example, the attacker could replace the password verifier of

an authorized identifier with his own or alter the sSRP service to always grant successful authentication.

WARP relies on an SRP password file on the identity provider to provide verifiers, salts, and public parameters for authorized users. Because the password verifiers are not plaintext equivalent to the password, an attacker who steals the password file is unable to use the verifier to impersonate the user without first determining the user's password.

SAW, and therefore WARP, are built around existing trust given to identity providers. An organization adopting WARP must make judgments regarding the identity providers they choose to trust. Some organizations may alternatively require use of identifiers within its own systems (e.g., organizational email addresses) to satisfy a required trust level.

3.5.5 Denial-Of-Service (DoS)

SRP is resilient against attackers modifying the information being exchanged between **U** and **H**. sSRP does nothing to compromise this resilience. At most, an attacker could cause authentication to fail. This could be used to deny service to an otherwise authorized user. As simpler, jamming-based denial-of-service attacks (DoS) already exist, the potential for DoS attacks using WARP is negligible.

3.5.6 Covert Channels

A covert channel is a method of communication that uses another channel's bandwidth to transmit data without knowledge or consent. Covert channels take many different forms (e.g., steganography, timing between transmissions, text manipulation). WARP could be used to send data between **U** and **IDP**. Because WARP only opens up a channel of communication between the supplicant and the provider of an authorized identifier, it cannot be used as a covert channel to any arbitrary party.

Using WARP as a covert channel would not be an effective means of transferring large amounts of data; the number and size of WARP's messages are quite small. Similar

authentication request throttling used to limit DoS attacks could also be employed to greatly reduce the amount of information that could be exchanged over the covert channel.

Since the link between the AS and IDP is unencrypted, the supplicant could communicate information to a passive eavesdropper on that link.

3.6 Deployability

Deployability of WARP can be discussed from three points of view: wireless users, organizations providing wireless access, and identity providers.

Users WARP leverages familiar and convenient password-based interfaces while decreasing the total number of required user passwords by reusing existing login credentials.

Organizations WARP softens the burden of password and account management. Providing access to regular organizational staff could be automated by generating the access control list using existing knowledge of staff identifiers. Guest access could then be maintained through a second access control list.

WARP is unusable in situations where the authentication server is unable to communicate with identity providers (e.g., ad-hoc networks without Internet connectivity).

WARP assumes the same level of trust extended to identity providers by SAW and may be inappropriate for use in organizations where the existing trust extended to third-party identity providers is insufficient.

Identity Providers WARP requires identity providers to host an sSRP service. Until incentives outweigh the cost, it is unlikely that every identity provider will be willing to meet this requirement. Fortunately, sSRP is simple and easy to implement and therefore comes at a minimal cost.

As mentioned previously, sSRP relies on an SRP password file to provide verifiers, salts, and public parameters for authorized users. Migration of existing non-SRP password

files is tricky because these files generally contain non-plaintext forms of the passwords, e.g., password hashes, and entries in an SRP password files must be generated from plaintext user passwords.

Identity providers who currently use services that receive a plaintext password from the user, e.g., Unix logins, could modify the service to intercept the password and generate an SRP password file entry. This removes the need for a formal re-enrollment process; users log in once to enable sSRP. The modified service could be deployed well in advance of sSRP deployment to generate SRP password entries for active users. Since existing password files are not used to generate the verifiers, they can be safely maintained side-by-side the SRP password file; facilitating gradual deployment.

3.7 Related Work

Many authentication mechanisms rely on pre-established shared secrets. Global passphrases used in systems like WPA-PSK[86] can be difficult to distribute. If the shared secret is compromised a new one must be deployed to each device. Guest access can only be achieved by disclosure of the shared secret. It is also difficult to audit access because each user connects using the same passphrase.

Individual user accounts with passwords are employed by challenge/response-based systems such as MSCHAPv2 [89]. Account-level password-based authentication provides reasonable deployability for a static group of users. It also lends itself to auditing because authentication is tied to a set of credentials. However, these systems suffer from the same issues of other password-based authentication mechanisms: password re-use and difficulty in remembering strong passwords. It is also difficult to configure guest access where account creation incurs significant overhead or time.

Recent authentication mechanisms, e.g., EAP-TLS [4], rely on user-specific digital certificates to provide authentication. However, managing user-specific certificates and securing public/private key-pairs are challenging tasks for even savvy end-users. The sign-up

process can be resource intensive, adding additional overhead on administrators who must verify user identity and issue certificates according to stringent company policies. Often this process must be performed not only for each user, but for each of the user's devices, which increases complexity.

Greenpass [34] leverages EAP-TLS for authentication. EAP-TLS relies on PKI to provide the client and server certificates used in the TLS handshake. Greenpass also provides decentralized delegation of access by allowing delegators to sign a SPKI/SDSI certificate that a guest can then use as their EAP-TLS client-side certificate. Greenpass still involves a CA issuing user-specific certificates to regular users.

Network-in-a-Box (NiaB) [9] enrolls devices in the wireless network by employing location-limited communication channels (e.g., infrared or a USB key) to securely distribute the necessary PKI keys and certificates. Although key distribution is simplified, NiaB still requires an administrator be present to authorize each device during enrollment. This location-limited channel approach creates a physical bottleneck when many devices need to be enrolled within a short time period (i.e., conference wireless access where most delegates arrive within a short time period).

3.8 Conclusions and Future Work

WARP is a convenient and secure wireless authentication mechanism. By preserving the principles of SAW, WARP enables decentralized user authentication and facilitates dynamic user bases in the wireless realm without requiring client-side Internet or IP-connectivity. WARP's use of SRP enables users to authenticate using existing personal messaging account identifiers and passwords, without fear that relying parties or eavesdroppers can compromise their login credentials. Although identity providers assist users in creating authenticated, encrypted sessions with relying parties, the identity providers cannot compromise these sessions.

sSRP could replace the use of email or instant messages in the original SAW protocol for website logins. Using sSRP in this manner increases SAW's ability to thwart active impersonation attacks and eliminate latency issues associated with personal message delivery.

We are currently investigating how to integrate the intuitive delegation between personal messaging identifiers and natural client-side auditing capabilities of SAW into WARP.

Forthcoming revisions to sSRP eliminate the need to use PKI-based approaches, e.g., EAP-TTLS, to prevent passive observation of KS_U and KS_{IDP}. These revisions also enable the authentication of relying parties to identity providers.

Acknowledgments. This research was supported by funding from the National Science Foundation under grant no. CCR-0325951, prime cooperative agreement no. IIS-0331707, and The Regents of the University of California.

Chapter 4

Luau: Lightweight User Authentication

T. W. van der Horst and K. E. Seamons. Luau: Lightweight User Authentication, *Technical Report*.

Additional Content: The original submission is augmented here with Section 4.5.4, which includes a brief discussion of the potential for delegation using Luau. Also, as one form of delegation relies on session resumption, the discussion on future work (see Section 4.7) is expanded with one potential approach to securely resume a session.

Abstract

Proof of email address ownership is typically required to create an account and to reset a password when it is forgotten. Despite its shortcomings (e.g., latency, vulnerability to passive attack), this approach is a practical solution to the difficult problem of authenticating strangers on the Internet.

Lightweight User AUthentication (Luau) utilizes this emergent, lightweight relationship with email providers to offload primary user authentication from service providers; thus reducing the need for service provider-specific passwords. To facilitate this, Luau enables more efficient and secure proofs of email address ownership.

While Luau is password-based, user passwords are never disclosed to service providers. Luau thwarts user impersonation by eavesdroppers and can be tailored to the risks of active attack found in four common deployment scenarios. When

a scenario requires server authentication and an encrypted session, Luau relies on existing secure tunnel schemes (e.g., TLS), but also takes into account that human error often negates the assurances of these tunnels and, therefore, enables service providers to detect a variety of man-in-the-middle attacks. Together, these protections significantly reduce the risk of password phishing.

4.1 Introduction

People have too many passwords. Services on the Internet regularly require that users create *yet another* username and password. This affects both security and convenience due to problems associated with password reuse [44] and forgotten passwords. Lightweight User AUthentication (Luau) is a new password-based decentralized authentication approach that leverages a widely adopted trust model and enables service providers to offload user authentication to third parties they already trust. Luau is designed to be a practical replacement to service-specific passwords.

4.1.1 Emergent Trust in Email Providers

Proving ownership of an email address by retrieving a message sent to it has become a valuable approach to authenticating unknown entities on the Internet. Garfinkel [33] coined the term email-based identification and authentication (EBIA) to describe this imperfect, but extremely popular tool. This mechanism is typically relegated to a *secondary* means of authentication; used to verify ownership of an email address specified during account creation and when, almost inevitably, the account password is forgotten. Pragmatically speaking, EBIA more than adequately fills its current role, and will likely continue to do so, without any modification, within the foreseeable future.

The pervasive acceptance of EBIA demonstrates that a myriad of service providers are willing to rely on email providers to authenticate users on their behalf. What, then,

curtails the utility of this approach as a *primary* means of user authentication? We identify four factors:

1. Latency of email delivery hinders convenience
2. Unavailability of the email provider prevents logins
3. Eavesdropping (by an attacker or the email provider) enables impersonation
4. Requiring users to directly connect to their email provider is not practical in some scenarios

Addressing the first and second factors may require a significant increase in email infrastructure or specialized handling of EBIA messages. The third factor is trivially addressed by "turning on" TLS protection for email delivery (though readily available, TLS is typically not enabled between the service provider and email provider due to economic/performance concerns). The fourth factor is typically a non-issue for web logins, however, it makes EBIA infeasible in scenarios where this connectivity is unavailable, e.g., wireless network authentication. None of these factors overtly hinder EBIA in its current role as an ad hoc secondary authentication mechanism.

EBIA has given the world a taste of its latent potential as a primary user authentication mechanism; a *practical* approach to decentralized authentication. Through EBIA, email providers have become the *de facto* identity providers on the Internet. The primary goal of this work is to improve the security and convenience of proof of email address ownership in order to leverage this emergent phenomenon as a primary authentication mechanism. Note that such an authentication scheme is a suitable replacement to passwords only at services that accept this trust model. For services that do not accept this model, this approach still has utility as a more secure and efficient means to verify ownership of an email address.

Given this goal, how should this inchoate trust in email providers be leveraged? Client-side digital certificates are an appealing solution, however, for the reasons outlined below, Luau provides a decentralized authentication approach that is password-based.

4.1.2 Client-side Digital Certificates: An ideal solution?

At a high level, email provider-issued, client-side digital certificates are an attractive replacement for service-specific passwords, and a step towards the goal of single sign-on. The scalability and "offline" capabilities provided by public key cryptography are very desirable. This technology has also weathered well the test of time. The RSA algorithm is over 30 years old [71] and this year marks the twentieth anniversary of the X.509 certificate format [20]. Yet, despite the worthy success of PKI in server deployments (e.g., HTTPS), the proliferation of client-side certificates is remarkably underwhelming. The "imminent" arrival of client-side certificates seems always just out of reach. This delayed arrival is due, in part, to burdensome configuration and usage requirements as well as certificate distribution, management, and revocation issues [35]. High costs, the requisite user and administrator training in a complex technology, and the difficulties of trusted roots and cross-certification also make PKI hard to adopt [32].

If overcoming the problems barring their adoption was simple, PKI vendors would have crushed these obstacles to increased profits long ago. This is a difficult problem. Decades have passed and still client-side certificates remain the exception rather than the norm. This tepid acceptance leads one to question, given the current landscape, whether certificates are a *practical* replacement to service-specific passwords.

4.1.3 Our Approach

Password-based decentralized authentication resides in the middle-ground between service-specific passwords and client-side certificates. Users have passwords to their *identity providers*, but not with each and every *relying party*. Relying parties therefore "rely" on identity providers to authenticate users on their behalf. As with EBIA, this approach depends on the availability of the identity providers; a non-trivial benefit of this dependence is real-time verification/revocation.

Several schemes (see Section 4.6) have been proposed in this vein and are gaining popularity and acceptance (especially OpenID [62]). Nevertheless, none of these systems advocate, as we do, the specific use of email providers as identity providers. Also, these approaches are typically geared specifically for web-based logins and require users to connect directly to their identity providers *during* an authentication attempt; a process usually accomplished via browser redirects. This reliance on browser redirects introduces attractive avenues for password phishing. Also, this web-centric focus makes these systems unsuitable for use in scenarios where direct communication between users and their identity providers is infeasible, such as authentication to a wireless network.

Our approach leverages the trust that is currently placed in email providers, thus eliminating the need to establish a new trusted identity provider. Luau also addresses the drawbacks of current approaches to password-based decentralized authentication, namely: password phishing, their reliance on humans to ensure that a secure tunnel (e.g., TLS) is established with the correct entity, and their inability to be used for wireless authentication. In addition to addressing security and convenience, Luau is designed with an eye towards practical deployments (see Section 4.5).

Paper outline Section 4.2 lays the foundation for Luau. Section 4.3 presents the Luau protocol. Section 4.4 analyzes its security. Section 4.5 considers deployment and implementation issues. Section 4.6 examines related work. Section 4.7 contains conclusions and future work.

4.2 Foundation

In this paper, the relying party (RP) desires to authenticate the user (U) with the assistance of the user's identity provider (IDP). We assume that U authenticates to IDP using a password and that RP has no pre-established shared secrets with IDP.

4.2.1 Threat Model

This section specifies the threat model used to compare EBIA, and other decentralized authentication mechanisms, to Luau. This model defines the likely deployment scenarios, the common methods of attack, and the attackers.

Deployment Scenarios In this paper a scenario is defined by the properties of two different communications channels. As each of these channels may be secured independently by employing a server-authenticated encrypted tunnel (e.g., TLS), there are, from a global perspective, four potential scenarios:

Scenario:	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$
Secure Tunnel ($\mathsf{U} \rightleftarrows \mathsf{RP}$)	NO	YES	NO	YES
Secure Tunnel ($\mathsf{RP} \rightleftarrows \mathsf{IDP}$)	NO	NO	YES	YES

Scenarios $\mathcal{S}_1$ and $\mathcal{S}_2$ are the most common for deployments of EBIA and correspond to HTTP and HTTPS logins, respectively, with unsecured SMTP to deliver messages to the email providers. $\mathcal{S}_3$ and $\mathcal{S}_4$ correspond to HTTP and HTTPS logins, respectively, with TLS-protected SMTP. Although presented in the context of web logins, these scenarios have corresponding meanings in wireless setups (i.e., replace HTTP/HTTPS with EAP/EAP-TTLS [3, 30]).

As it is rare for email providers to support TLS-protected SMTP, $\mathcal{S}_1$ and $\mathcal{S}_2$ represent the most frequently used scenarios. Therefore, even though the authentication of IDP, which comes as part of the secure tunnel between RP and IDP, would be useful to a decentralized authentication system, EBIA is evidence that significant value can be achieved without requiring it. Also, the abundance of services that operate in $\mathcal{S}_1$ (e.g., low security web sites with HTTP login pages) indicates that there is value in not having to authenticate (via PKI) some relying parties.

Attacks The goal of login mechanisms is typically unilateral entity authentication (U to RP) as authentication of RP to U and any keys to secure the resulting session are han-

dled externally (e.g., with a server-authenticated encrypted tunnel). Therefore of primary concern is the ability for an attacker to impersonate U to RP, hereafter referred to as an $\mathcal{IMP}_{\mathsf{U}\to\mathsf{RP}}$ break. Directly related to this is an attacker's ability to impersonate IDP to RP, hereafter referred to as an $\mathcal{IMP}_{\mathsf{IDP}\to\mathsf{RP}}$ break. An $\mathcal{IMP}_{\mathsf{IDP}\to\mathsf{RP}}$ break implies an $\mathcal{IMP}_{\mathsf{U}\to\mathsf{RP}}$ break as the ability to impersonate IDP enables an attacker to mount a synchronized attack to impersonate any of IDP's users to RP.

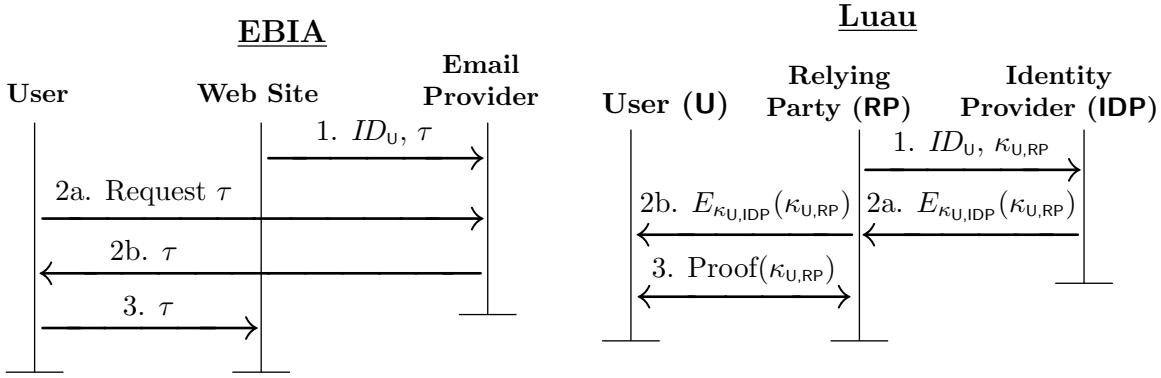
If the attacker can insert herself between U and RP she can become a *man-in-the-middle* (MITM) and if she is not detected this constitutes a $\mathcal{MITM}$ break. This break enables an attacker to eavesdrop or hijack the resulting session between U and RP. There are three common methods for an attacker to become a MITM: 1) Routing-MITM, an attacker controls a router between U and RP or has tricked traffic to route through her (e.g., ARP poisoning); 2) Pharming-MITM, DNS is poisoned so that lookups return a network address controlled by the attacker; and 3) Phishing-MITM, U is tricked into connecting to the attacker in lieu of the legitimate host (e.g., clicks a link in a phishing email).

One incarnation of a MITM attack is an RP that attempts to manipulate U's authentication so that it can impersonate U to a different RP. For example, U thinks she is logging in to comment on a blog, but the blog wants to log in as her to her online bank.

Attackers Two attackers are considered in this model. Eve ($\mathcal{E}$) is a passive eavesdropper whose goals are $\mathcal{IMP}_{\mathsf{U}\to\mathsf{RP}}$ and $\mathcal{IMP}_{\mathsf{IDP}\to\mathsf{RP}}$ breaks. Mallory ($\mathcal{M}$) is an active attacker whose goals are $\mathcal{IMP}_{\mathsf{U}\to\mathsf{RP}}$, $\mathcal{IMP}_{\mathsf{IDP}\to\mathsf{RP}}$, and $\mathcal{MITM}$ breaks. $\mathcal{E}$ is limited to observing the normal interactions between the three parties. $\mathcal{M}$ can observe, inject, modify, delay, destroy, and replay messages as well as create multiple concurrent sessions with any other party.

4.2.2 EBIA Trust Model

EBIA is founded on the premise that email providers deliver messages to their intended recipients and that they do not impersonate their users without authorization. The three



Stage 1: Web site gives email provider token τ. (It is understood that the email provider will only give τ to the legitimate user identified by ID_U.)

Stage 2: The user connects to her email provider, authenticates, and retrieves token τ.

Stage 3: The user gives τ to the web site.

Stage 1: Relying party and identity provider establish key $\kappa_{U,RP}$. (Again, the identity provider will only give $\kappa_{U,RP}$ to the legitimate user identified by ID_U.)

Stage 2: The identity provider encrypts $\kappa_{U,RP}$ such that only the legitimate user can decrypt it. The relying party forwards this package to the user.

Stage 3: The user decrypts $\kappa_{U,RP}$. Both the user and the relying party then prove knowledge of $\kappa_{U,RP}$ to each other.

Figure 4.1: High level approaches of EBIA and Luau.

stages for an authentication with EBIA are given in Figure 4.1. In Stage 1, the web site gives the email provider a message to deliver to one of its clients. This message is composed of the user identifier and a session-specific token. In Stage 2, the user connects to her email provider and retrieves this message. In Stage 3, the user returns the token to the web site, thus demonstrating ownership of the email address.

At a higher level of abstraction, EBIA relies on a two part approach. The first is a 3-party key distribution (Stages 1 and 2). The second is an entity authentication of the user based on that key (Stage 3).

EBIA assumes the existence of a discovery function which determines the identity provider from the user's identity; given an email address it is a simple process to discover the corresponding email provider.

Secure Tunnels EBIA relies on a secure tunnel (e.g., TLS) to authenticate RP to U (and, optionally, IDP to RP) as well as to secure the resulting channels. When the risk of attack is assumed to be negligible (e.g., $\mathcal{S}_1$), tunnels are typically not employed, however, if risks become greater, tunnels are enabled on a per channel basis to protect that link. Failure to establish, or to correctly verify, this tunnel completely invalidates its protections.

Negating Secure Tunnels Phishers routinely sidestep secure tunnels. The simplest method to circumvent a tunnel is to not create it and hope that the user does not notice. For example, a web page is made to appear exactly like the real site, but it is not sent over HTTPS[1] so the server is never authenticated to the client (as it usually would have been).

Another approach, hereafter referred to as the *certificate trick*, involves the user accepting the wrong certificate for the real server. There are two common avenues for this attack. First, the attacker sends the fake page over HTTPS[2] using a valid certificate issued to the phishing site, not the real site. As the certificate is valid for this site, no warnings are issued by the browser. Second, the attacker creates a self-signed certificate and assumes the user will ignore all browser warnings and accept this certificate[3].

The rampant success of phishing demonstrates the risk of relying solely on the user to ensure that a tunnel is established with the correct entity.

Attacking EBIA The vulnerabilities of EBIA are summarized as follows (recall that $\mathcal{IMP}_{\mathsf{IDP}\rightarrow\mathsf{RP}}$ also implies $\mathcal{IMP}_{\mathsf{U}\rightarrow\mathsf{RP}}$):

[1]The Anti-Phishing Working Group (APWG) reports that 99.23% of phishing occurs over plain HTTP [7].

[2]APWG estimates that 0.28% of phishing occurs over HTTPS [7].

[3]These attacks are not limited to phishers. Cain and Able [17] is a popular tool that uses ARP poisoning to force all local network traffic to flow through the tool. It also provides an automated TLS man-in-the-middle attack using a self-signed certificate.

	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$
$\mathcal{E}$	$\mathcal{IMP}_{\mathsf{IDP}\rightarrow\mathsf{RP}}$	$\mathcal{IMP}_{\mathsf{IDP}\rightarrow\mathsf{RP}}$	None	None
$\mathcal{M}$	$\mathcal{IMP}_{\mathsf{IDP}\rightarrow\mathsf{RP}}$, $\mathcal{MITM}$	$\mathcal{IMP}_{\mathsf{IDP}\rightarrow\mathsf{RP}}$, $\mathcal{MITM}$*	$\mathcal{MITM}$	$\mathcal{MITM}$*

* – requires certificate trick

In $\mathcal{S}_1$ and $\mathcal{S}_2$, EBIA is vulnerable to $\mathcal{IMP}_{\mathsf{IDP}\rightarrow\mathsf{RP}}$ breaks by both $\mathcal{E}$ and $\mathcal{M}$ since the token τ is not encrypted in Stage 1. When SMTP is tunneled through TLS (as it is in $\mathcal{S}_3$ and $\mathcal{S}_4$) then τ cannot be observed by either attacker. In $\mathcal{S}_1$ and $\mathcal{S}_3$ there is no protection (other than human observation) from $\mathcal{MITM}$ breaks. In $\mathcal{S}_2$ and $\mathcal{S}_4$ the server-authenticated tunnel provides additional methods for humans to detect MITM attacks, but it is still vulnerable to the certificate trick[4]. Although τ is submitted (in Stage 3) to RP in plaintext in $\mathcal{S}_3$, $\mathcal{E}$ cannot use this to impersonate U as τ is typically a short-lived, single-use token that is invalidated once U has presented it.

While EBIA provides better assurances in $\mathcal{S}_3$ and $\mathcal{S}_4$ than in $\mathcal{S}_1$ and $\mathcal{S}_2$, recall that these scenarios are rare in actual deployments.

4.3 Luau

Goals The primary goal of password-based authentication is to ensure that U's password is required to impersonate U. In addition to this aim and addressing the four factors that hinder the use of EBIA as a primary authentication mechanism, Luau is designed to:

1. Eliminate passive attacks.
2. Improve detection by U and RP of MITM attacks.
3. Reduce the risk of password phishing.

The first goal addresses a significant drawback of EBIA: the ability for $\mathcal{E}$ to impersonate U, or IDP, to RP. As EBIA has demonstrated the utility of unauthenticated identity

[4]In $\mathcal{S}_4$, $\mathcal{MITM}$ breaks are possible using routing/pharming-MITM, but not using phishing-MITM attacks as the message sent by RP typically contains a link to the legitimate server. $\mathcal{S}_2$, however, is vulnerable to a phishing-MITM attack as $\mathcal{M}$ can modify this link to point to herself.

providers, we require that protection from $\mathcal{E}$ not necessitate a secure tunnel between RP and IDP. Nevertheless, it should allow for the easy addition of such a tunnel, if authentication of IDP and protection from $\mathcal{M}$ is desired.

The second goal is based on the observation that, in all four scenarios, it is often possible to detect a MITM attack by using information (e.g., domain names, network addresses, digital certificates used in a tunnel's creation) that is readily accessible to each party. Even if a secure tunnel is used, Luau operates on the assumption that all communications are over an open channel until both sides are given the opportunity to detect MITM-attacks.

The third goal is motivated by the effectiveness of phishers and the certificate trick, even if RP employs server-authenticated, encrypted tunnels to strengthen its ability to authenticate itself to users. This idea is not reflected in existing tunnels (e.g., TLS) as they rely solely on users to detect MITM attacks. As users can be easily tricked into negating a tunnel's benefits, Luau uses a secure tunnel, if available, in the following capacities: 1) Improved identification of RP to detect MITM attacks; and 2) Protect, *after* a successful authentication, the resulting session.

High Level Approach Similar to EBIA, Luau follows a key distribution and entity authentication approach (see Figure 4.1). In Stage 1, RP establishes a shared secret $\kappa_{\mathsf{U,RP}}$ with IDP and supplies U's identifier ID_{U}. Rather than requiring U to directly contact IDP to retrieve $\kappa_{\mathsf{U,RP}}$, an in-band distribution mechanism is leveraged. In Stage 2a, IDP returns $\kappa_{\mathsf{U,RP}}$ encrypted with the key $\kappa_{\mathsf{U,IDP}}$, which is shared between U and IDP (the establishment of this key is covered in Section 4.3.1). RP then forwards the encrypted $\kappa_{\mathsf{U,RP}}$ to the initiator of the authentication (Stage 2b).

This in-band approach is more efficient than the out-of-band approach of EBIA because the authentication of U to IDP is implicit and it avoids the complexity of the interaction between U and IDP to retrieve the correct token. This in-band technique also eliminates the

vulnerability to phishing introduced by browser redirects and enables use of this approach in a wireless authentication scenario, when U cannot directly contact IDP.

In Stage 3, U and RP demonstrate to each other that they know $\kappa_{U,RP}$. A mutual proof is preferable to simply returning $\kappa_{U,RP}$ to RP because it helps ensure that the authentication is non-transferrable (i.e., it only has meaning to the RP that initiated it) and it enables RP to detect MITM attacks.

Unlike EBIA, Luau requires the modification, and the active participation, of all three parties. We believe that the benefits of improved security (i.e., protections from passive and active attacks, including phishing) and convenience (i.e., lower latency, removing need for site-specific passwords, and wireless authentication) are sufficient to merit this change. In both approaches, the dependence on the reliability of the email provider remains, though this drawback can be mitigated for the common case (see Section 4.5.3).

4.3.1 Building Blocks

Luau builds on existing approaches for: 1) Converting a password to a shared secret; 2) Three party key distribution; and 3) Mutual authentication using a shared secret.

Shared Secret between U and IDP ($\kappa_{U,IDP}$) It is convenient to model the relationship between U and IDP as a shared symmetric key $\kappa_{U,IDP}$, however, this is rarely the case. Typically, U and IDP have a password and a password verifier, respectively. We identify three methods to leverage a password to obtain a shared symmetric key:

Password-derived key [*Not Advised*] A straightforward password-derived key has the disadvantage of leaking material to verify an offline password guessing attack. One benefit of this approach is that U and IDP can independently derive this key without direct communication. If used, the key derivation function should be relatively time-intensive to limit the effectiveness of an offline attack.

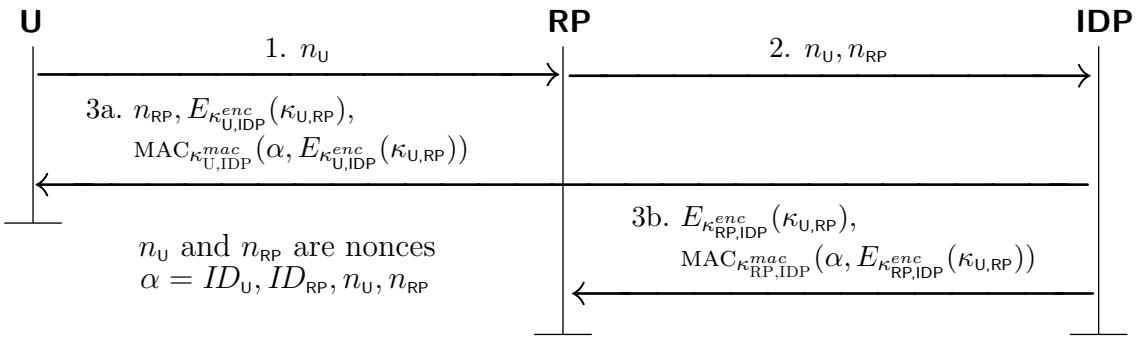


Figure 4.2: Three party key distribution [22]. The keys $\kappa^{enc}_{\mathsf{U,IDP}}$, $\kappa^{mac}_{\mathsf{U,IDP}}$, $\kappa^{enc}_{\mathsf{RP,IDP}}$, $\kappa^{mac}_{\mathsf{RP,IDP}}$ are pre-established, while $\kappa_{\mathsf{U,RP}}$ is generated by IDP. $E_\kappa(\cdot)$ is symmetric encryption. $\mathrm{MAC}_\kappa(\cdot)$ is a message authentication code.

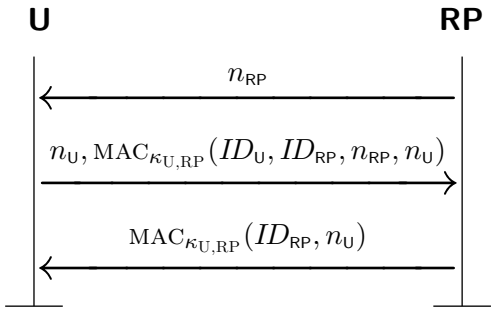
Password-independent key [*Acceptable*] U directly connects to IDP and, after authenticating, they agree on a value for the key (e.g., IDP selects a random key). A benefit of this approach is that this key cannot be used to guess U's password offline. While this approach is acceptable for typical web usage, it is not feasible in most wireless authentication scenarios, unless this key is established a priori, as it requires direct communication between U and IDP.

Password-authenticated key exchange [*Preferred*] In PAKE protocols (e.g., EKE [14], SPEKE [45], SRP [87]) a password is used to establish a key without leaking material for an offline password guessing attack. As these protocols are designed to operate securely over unprotected channels it is feasible for RP to relay the PAKE protocol messages between U and IDP, thus ensuring compatibility with the needs of wireless authentication (see Section 4.5.2 for further justification for why RP would do this).

Luau requires that U and IDP establish $\kappa_{\mathsf{U,IDP}}$ (which has two parts $\kappa^{enc}_{\mathsf{U,IDP}}$ and $\kappa^{mac}_{\mathsf{U,IDP}}$) before U attempts to authenticate to RP. This is hereafter referred to as *pre-authentication*. Once this key is obtained U should not have to directly communicate with IDP until this key expires. It is reasonable to assume that U and IDP may have multiple valid keys at any one time (e.g., multiple devices each with their own key), therefore, each key must have a unique identifier $ID_{\kappa_{\mathsf{U,IDP}}}$.

Three Party Key Distribution 3PKD [12] provides provably secure three party key distribution. It assumes that two parties (e.g., U and RP) each share a unique key with a trusted third party (e.g., IDP), but not with each other. Both parties rely on IDP to assist them in obtaining a shared secret with each other. Choo et al. [22] improved the original 3PKD algorithm. The improved protocol, altered to reflect notation used in this paper, is shown in Figure 4.2.

Mutual Authentication Mutual authentication protocol one (MAP1) enables two parties to authenticate each other in a provably secure fashion. It assumes two parties share the key $\kappa_{\mathsf{U,RP}}$ and uses message authentication via pseudo-random functions[5] (PRF) to accomplish entity authentication. The protocol, altered to reflect the notation used in this paper and with RP as the initiator, is as follows:



4.3.2 Constructing Luau

Luau modifies 3PKD to enable IDP to help U and RP to establish a key within the constraints and assumptions of Luau. It then uses MAP1 to authenticate U to RP.

Modifications to 3PKD In 3PKD, IDP unilaterally generates $\kappa_{\mathsf{U,RP}}$ and encrypts it such that only those entities with access to the keys associated with ID_{U} and ID_{RP} can decrypt it. As such, both sides can identify who else knows $\kappa_{\mathsf{U,RP}}$. 3PKD cannot be utilized "as is" since two of its assumptions do not hold in Luau:

[5] In practice, a PRF is often constructed using secure hash functions and message authentication codes (MAC) [11].

1. RP and IDP do not share a key.

2. ID_{RP} has no meaning to IDP.

The first assumption is not viable because it is not scalable for each RP to share a key with every IDP, nor are public keys a viable alternative as they violate the assumption of $\mathcal{S}_1$ and $\mathcal{S}_3$ that RP cannot be authenticated via PKI. The second assumption does not hold since, as demonstrated by EBIA, the actual identity of RP (ID_{RP}) is ultimately immaterial to IDP.

Luau relies on a Diffie-Hellman (DH) key exchange [25] to establish a seed key $\kappa_{\mathsf{RP,IDP}}$ between RP and IDP that eliminates the complexity of managing long-term keys at both RP and IDP. Luau assumes the presence of pre-established, well-known, well-tested generator/prime pairs (g, p) for a specified key size (e.g., [49, 52]) for use in the DH key exchange. This prevents $\mathcal{M}$ from injecting a pair for which she can compute discrete logs[6]. RP ultimately selects which pair to use for a specified run of the protocol.

This key exchange ensures that $\kappa_{\mathsf{U,RP}}$, which is derived from $\kappa_{\mathsf{RP,IDP}}$, cannot be observed by $\mathcal{E}$ and removes the need for $E_{\kappa^{enc}_{\mathsf{RP,IDP}}}(\kappa_{\mathsf{U,RP}})$ in message 3b. Protection from $\mathcal{M}$ is added by employing a secure tunnel for this channel. While these modifications do not affect IDP's ability to transfer $\kappa_{\mathsf{U,RP}}$ to U, it does remove the ability for RP to authenticate to U using $\kappa_{\mathsf{U,RP}}$. Luau addresses this limitation using MAP1.

Integrating MAP1 Using $\kappa_{\mathsf{U,RP}}$ and MAP1 it is a simple process to authenticate U to RP, however, as ID_{RP} is meaningless to IDP, an authentication of RP to U cannot rely on $\kappa_{\mathsf{U,RP}}$. Therefore, Luau relies on well-established techniques for identifying/authenticating servers on the Internet to specify ID_{RP}. In $\mathcal{S}_1$ and $\mathcal{S}_3$ it is sufficient to determine that U is connected to the legitimate RP based on network identifiers (e.g., domain names, network addresses). In $\mathcal{S}_2$ and $\mathcal{S}_4$ the server authentication of the encrypted tunnel (e.g., digital certificate) is

[6]For example, $p = 3$, $p - 1$ is composed of only small prime factors, and many cases of $p = 2^n$ [73].

leveraged in addition to available network identifiers. Using MAP1, Luau couples the claim that RP knows $\kappa_{\mathsf{U,RP}}$ with the scenario-dictated value for ID_{RP}.

Luau makes RP the initiator of the MAP1 exchange rather than U. This allows U to authenticate first and, thus, permits RP to use the value for ID_{RP} (as perceived by U) to detect MITM attacks. If RP detects such an attack, it communicates this to U by *not* sending the final MAP1 message, thus forcing the authentication attempt to abort.

As MAP1 may be contained within an encrypted tunnel, it is important to address the *compound authentication binding problem* [69]. This problem illustrates the need to couple an inner authentication protocol with its outer, server-authenticated tunnel to prevent MITM-attacks. The typical solution requires that the inner authentication protocol produce a session key and that this key be bound with the session key produced by the tunnel. Luau does not utilize this approach.

The main reason for not adhering to the traditional solution is that it is not practical to modify existing tunnel libraries (e.g., TLS) to accept external keying material. Instead, Luau couples the outer tunnel with the inner authentication protocol through the value used for ID_{RP}. Recall that when a tunnel is employed, the server's certificate is included in ID_{RP}. This binding also ensures the non-transferability of the authentication of U to RP to other relying parties.

As Luau does not require privileged access to secure tunnel libraries, it is feasible to implement Luau at an application layer and is easy to offload tunneling functionality to another machine.

4.3.3 The Luau Protocol

Luau leverages the lightweight relationship between service providers and email providers to authenticate U to RP. It eschews the inherent latency of EBIA in favor of an in-band approach that improves efficiency, avoids the phishing avenues introduced by browser redirects, and is compatible with the needs of wireless authentication.

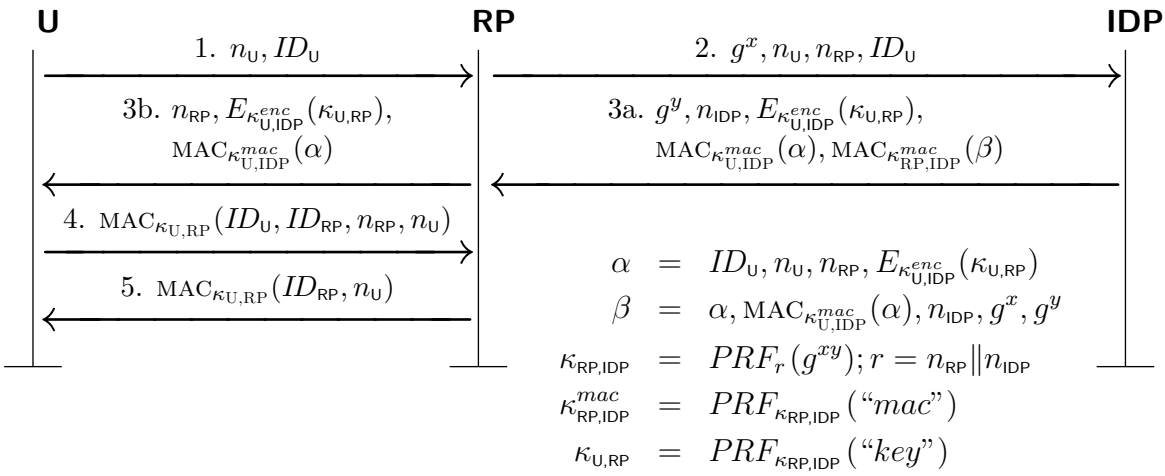


Figure 4.3: The Luau protocol. The keys $\kappa^{enc}_{\mathsf{U,IDP}}$ and $\kappa^{mac}_{\mathsf{U,IDP}}$ are pre-established (see Section 4.3.1).

While IDP can impersonate U, Luau seeks to eliminate, without requiring PKI, all passive impersonation attacks. Luau also seeks to mitigate the risk of users negating the secure tunnel and improve detection of MITM attacks by both U and RP. Recall that each deployment scenario dictates the use of secure tunnels on a per channel basis.

Protocol at a Glance As Luau assumes that RP and IDP do not have a pre-established secret, it creates $\kappa_{\mathsf{RP,IDP}}$ using a DH key exchange. If authentication of IDP to RP and protection from $\mathcal{M}$ is desired, a server-authenticated tunnel is used to protect the channel between RP and IDP.

A modified 3PKD transports the key $\kappa_{\mathsf{U,RP}}$ (derived from $\kappa_{\mathsf{RP,IDP}}$) to U through RP. U authenticates to RP via MAP1. If authentication of RP to U and protection from $\mathcal{M}$ is desired a secure tunnel is enabled between U and RP.

Protocol Messages Luau's protocol messages are illustrated in Figure 4.3. The first message of 3PKD and n_{U} from MAP1 are combined to form Luau's first message. Luau's second message adds RP's DH public param g^x to message 2 of 3PKD. Both the first and second messages of Luau explicitly specify ID_{U}, which also includes $ID_{\kappa_{\mathsf{U,IDP}}}$ for the desired

$\kappa_{\mathsf{U,IDP}}$. Note that the selected generator/prime pair for the DH key exchange is included with g^x.

Message 3a of Luau is a combination of messages 3a and 3b of 3PKD, and adds IDP's DH public param g^y and nonce n_{IDP}. Note that $E_{\kappa^{enc}_{\mathsf{RP,IDP}}}(\kappa_{\mathsf{U,RP}})$ is removed as RP uses the DH exchange to derive $\kappa_{\mathsf{U,RP}}$ and that both MACs have been updated to account for the new/removed elements of this message and message 2. n_{RP} is also removed since all messages pass through RP, which can easily append it in message 3b; the MAC in message 3b ensures that RP appends the correct value. Routing IDP's message to U through RP is more efficient (less communication in a global sense) and it ensures availability when U cannot directly contact IDP (e.g., wireless network authentication).

Messages 4 and 5 constitute the final messages of MAP1 with RP playing the role of the initiator rather than U.

4.4 Security Analysis

Though we believe that Luau lends itself well to further study in existing proof models (e.g., [11], [12], [10], [18]), due to lack of space, this section focuses on the arguments to construct such a proof.

At a high level, we argue that the properties of MAP1 and our modified 3PKD reduce $\mathcal{IMP}_{\mathsf{U}\rightarrow\mathsf{RP}}$ and $\mathcal{IMP}_{\mathsf{IDP}\rightarrow\mathsf{RP}}$ breaks down to the assurances of the DH key exchange between RP and IDP. It is also argued that ID_{RP} eliminates $\mathcal{MITM}$ breaks in $\mathcal{S}_4$ (even when the certificate trick is successfully employed) and increases the difficultly for an attacker to mount $\mathcal{MITM}$ breaks in $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$.

As all stored information is vulnerable to a "break-in" or some other form of leakage (e.g., cryptanalysis), this section also examines the impact of compromised session and long-term secrets. Lastly, this section explores phishing, privacy issues, and the potential for covert channels.

Assumptions Luau uses the same assumptions as MAP1 and 3PKD with respect to encryption schemes (e.g., AES) and pseudo-random functions (e.g., HMAC), namely, the strength of these primitives are determined by key size.

The best assurance a password protocol can provide is that an attacker has no advantage against the protocol greater than online guessing [36]. As Luau relies on existing password-based schemes to establish $\kappa_{\mathsf{U,IDP}}$, we make the following assumption:

Assumption 1 *Within the context of pre-authentication:*

- *If a password-derived key is used, then, at best, $\kappa_{U,IDP}$ cannot be obtained with a greater advantage than offline guessing since RP knows $\kappa_{U,RP}$ and it can use $E_{\kappa^{enc}_{U,IDP}}(\kappa_{U,RP})$ to verify an offline guessing attack.*
- *If a password-independent or PAKE protocol is used, then $\kappa_{U,IDP}$ cannot be obtained with a greater advantage than online guessing.*

4.4.1 Attacking Luau

As argued below, Luau thwarts passive attacks, foils the certificate trick in $\mathcal{S}_4$, and raises the bar for $\mathcal{MITM}$ breaks in $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$. Note that $\mathcal{IMP}_{\mathsf{IDP}\rightarrow\mathsf{RP}}$ breaks by $\mathcal{M}$ in $\mathcal{S}_1$ and $\mathcal{S}_2$ are still possible as IDP is not authenticated to RP.

Preventing $\mathcal{IMP}_{\mathsf{U}\rightarrow\mathsf{RP}}$ MAP1 is used to authenticate U to RP. This protocol carries the assurance that $\kappa_{\mathsf{U,RP}}$ is required to impersonate U to RP with non-negligible advantage over random guessing (the proof of the protocol shows that attacks on it can be reduced to breaking pseudorandom functions [11]). Therefore, in order to mount a successful $\mathcal{IMP}_{\mathsf{U}\rightarrow\mathsf{RP}}$ break an attacker must obtain $\kappa_{\mathsf{U,RP}}$.

As Luau assumes strong encryption, there are two methods for an attacker to procure $\kappa_{\mathsf{U,RP}}$: 1) Obtain $\kappa_{\mathsf{U,IDP}}$ and decrypt $E_{\kappa^{enc}_{\mathsf{U,IDP}}}(\kappa_{\mathsf{U,RP}})$; or 2) Compromise the DH key exchange that creates $\kappa_{\mathsf{RP,IDP}}$.

By Assumption 1, an attacker cannot obtain $\kappa_{\mathsf{U,IDP}}$ with greater advantage than online guessing (recall that this is the best a password-based protocol can achieve). With respect to compromising the DH key exchange, given sufficiently strong generator/prime pairs and exponent sizes (e.g., [49, 52]), it is reasonable to claim $\mathcal{E}$ cannot obtain $\kappa_{\mathsf{RP,IDP}}$ in any of the four deployment scenarios.

In $\mathcal{S}_1$ and $\mathcal{S}_2$, $\mathcal{M}$ can perform a DH-MITM attack and obtain the values that RP and IDP will derive for $\kappa_{\mathsf{RP,IDP}}$ (the nature of the DH-MITM attack ensures that they each generate different values). Using RP's value, $\mathcal{M}$ can derive $\kappa_{\mathsf{U,RP}}$ and perform an $\mathcal{IMP}_{\mathsf{U}\rightarrow\mathsf{RP}}$ break. Nevertheless, in $\mathcal{S}_3$ and $\mathcal{S}_4$, the server-authentication of the encrypted tunnel prevents $\mathcal{M}$ from compromising the DH key exchange in these scenarios.

Unlike other password-based decentralized authentication solutions (e.g., OpenID), Luau does not use cookies to maintain an authenticated state with IDP and, as such, problems with cross-site scripting and some browser vulnerabilities are avoided.

Preventing $\mathcal{IMP}_{\mathsf{IDP}\rightarrow\mathsf{RP}}$ With $\kappa_{\mathsf{RP,IDP}}$ an attacker can impersonate any of IDP's users to RP using a synchronized attack in which the attacker also acts as U. As discussed above, $\kappa_{\mathsf{RP,IDP}}$ can only be obtained by compromising the DH key exchange between RP and IDP, which is only possible by $\mathcal{M}$ in $\mathcal{S}_1$ and $\mathcal{S}_2$.

The purpose of $\text{MAC}_{\kappa^{mac}_{\text{RP,IDP}}}(\beta)$ is to provide a "proof of possession" of $\kappa_{\mathsf{RP,IDP}}$ as well as provide limited protection from active attack; it forces a DH-MITM attack before $\mathcal{M}$ can make undetected[7] modifications to the messages between RP and IDP. For example, without this protection, an attacker could replace ID_{U} with the identifier of a different valid user in order to enable that user to impersonate the victim. This protection adds minimal overhead as it piggybacks on an existing MAC in 3PKD.

Between RP and IDP, Luau is not bound to the external tunnel as it is between U and RP. The term MITM has no meaning to IDP since it only interacts with unauthenticated

[7] In order to remain undetected $\mathcal{M}$ must modify $\text{MAC}_{\kappa^{mac}_{\text{RP,IDP}}}(\beta)$ to reflect its (not IDP's) DH public parameters.

relying parties. As such it remains the sole responsibility of RP to detect MITM attacks on this channel. The key $\kappa_{\mathsf{U,IDP}}$ could be used by U to indicate a specific RP, however, without a method for IDP to authenticate RP it would be meaningless. Also, we believe it is reasonable to assume that RP is less likely to fall prey to MITM attacks since a human is not involved in the real-time verification of IDP's identity.

Addressing $\mathcal{MITM}$ breaks Recall that, by definition, only $\mathcal{M}$ can perform $\mathcal{MITM}$ breaks and that a successful break requires $\mathcal{M}$ to mount a MITM attack between U and RP while avoiding detection. The success of phishing demonstrates that human judgment (i.e., checking the readily observable network identifiers and, if present, digital certificate) provides only weak protection against MITM attacks.

Luau improves the detection of MITM attacks for both U and RP by requiring each party to commit (using $\kappa_{\mathsf{U,RP}}$) to its value for ID_{RP} (i.e., network identifiers and, if present, certificate) in messages 4 and 5. This additional protection requires very little overhead as Luau piggybacks this protection on the existing MACs of MAP1. The value of ID_{RP}, and $\mathcal{M}$'s ability to obtain $\kappa_{\mathsf{U,RP}}$ in $\mathcal{S}_1/\mathcal{S}_2$, make this protection scenario-dependent:

$\mathcal{S}_1$ Routing-MITM attacks cannot be detected in this scenario. Undetected pharming/phishing-MITM attacks are possible, but require a coordinated DH-MITM attack in which $\mathcal{M}$ uses IDP to generate a legitimate $E_{\kappa^{enc}_{\mathsf{U,IDP}}}(\kappa_{\mathsf{U,RP}})$ and $\text{MAC}_{\kappa^{mac}_{\text{U,IDP}}}(\alpha)$ and where $\mathcal{M}$ modifies messages 4 and 5 using $\kappa_{\mathsf{U,RP}}$ so that they contain the "expected" value for ID_{RP}.

$\mathcal{S}_2$ $\mathcal{MITM}$ breaks are possible on two conditions. First, the certificate trick must be employed since RP authenticates to U in this scenario. Second, $\mathcal{M}$ must also perform a DH-MITM attack between RP and IDP and use its values for $\kappa_{\mathsf{U,RP}}$ to modify messages 4 and 5.

$\mathcal{S}_3$ As in $\mathcal{S}_1$, routing-MITM attacks cannot be detected in this scenario as they effectively mask their presence in the network identifiers used for ID_{RP}. Other MITM attacks are

not possible because the secure tunnel between RP and IDP thwarts the DH-MITM attack that enables $\mathcal{M}$ to modify messages 4 and 5 in $\mathcal{S}_1$ and $\mathcal{S}_2$.

$\mathcal{S}_4$ As ID_{RP} includes RP's authenticated certificate, it is possible to detect routing-MITM attacks and the certificate trick because $\mathcal{M}$ cannot spoof ownership of RP's certificate without access to RP's private key. As in $\mathcal{S}_3$, $\mathcal{M}$ cannot modify messages 4 and 5 and therefore cannot perform pharming/phishing-MITM attacks.

Summary The vulnerabilities of Luau are summarized as follows (recall that $\mathcal{IMP}_{\mathsf{IDP}\rightarrow\mathsf{RP}}$ also implies $\mathcal{IMP}_{\mathsf{U}\rightarrow\mathsf{RP}}$):

	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$
$\mathcal{E}$	None	None	None	None
$\mathcal{M}$	$\mathcal{IMP}_{\mathsf{IDP}\rightarrow\mathsf{RP}}$, $\mathcal{MITM}$*	$\mathcal{IMP}_{\mathsf{IDP}\rightarrow\mathsf{RP}}$, $\mathcal{MITM}$†	$\mathcal{MITM}$‡	None

* – may require DH-MITM between RP and IDP

† – requires certificate trick and DH-MITM

‡ – only via routing-MITM

Though vulnerabilities to $\mathcal{M}$ exist in $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$, do not discount the potential utility of Luau in these scenarios. These weaknesses are rooted in the requirements of the deployment scenario (e.g., unauthenticated identity providers in $\mathcal{S}_1/\mathcal{S}_2$), which EBIA has demonstrated to be worthwhile, despite its risks.

4.4.2 Additional Considerations

Identifying RP and Preventing Phishing Luau ensures that U's password is never disclosed to relying parties (i.e., prevents password phishing) in all four scenarios. Nevertheless, thwarting phishing in general is limited by the ability for users to unambiguously specify the desired RP.

This is a problem with decentralized authentication systems in general, including client-side certificate approaches. As U can authenticate to any RP, how does she know if she is at the right one?

Luau's philosophy is to treat all RP's as untrusted and ensure that U's password is never disclosed and that an authentication attempt cannot be used to allow this RP to impersonate U at another RP. Nevertheless, it can only do so for the context of the authentication process. After the authentication is complete, it is outside Luau's influence to prevent the RP from attempting to elicit additional information from U under the guise of being someone else.

One potential solution is to add a continuity system that enables users to recognize if they have previously authenticated to a specific RP. This straightforward approach provides some utility, but does not preclude an RP that U has visited before from mounting attacks.

Compromise of Session Secrets The nonces (n_{RP}, n_{IDP}) in this protocol are used in the same manner as they are in IKEv2 [47] to create a seed key ($\kappa_{\mathsf{RP,IDP}}$) from the result of the DH key exchange (g^{xy}). These nonces also enable RP and IDP to reuse their DH parameters across multiple sessions (e.g., for performance reasons) while ensuring that each session is unique. These nonces also introduce session-specific randomness from each participant to ensure the freshness of the keys used in this session.

Compromising RP's or IDP's DH exponent (x or y, respectively) is not necessary to mount $\mathcal{IMP}_{\mathsf{IDP}\rightarrow\mathsf{RP}}$ breaks in $\mathcal{S}_1$ and $\mathcal{S}_2$. In the other two scenarios, the use of an encrypted tunnel between RP and IDP, ensures that an attacker cannot learn $\kappa_{\mathsf{RP,IDP}}$.

If $\kappa_{\mathsf{U,RP}}$ is compromised, then $\mathcal{M}$ can impersonate U to RP during the valid lifetime of $\kappa_{\mathsf{U,RP}}$ at RP. If $\kappa_{\mathsf{RP,IDP}}$ is compromised then $\mathcal{M}$ can derive a valid $\kappa_{\mathsf{U,RP}}$ for the user specified by RP when $\kappa_{\mathsf{RP,IDP}}$ was created. If $\kappa_{\mathsf{U,IDP}}$ is compromised then $\mathcal{M}$ can impersonate U to any RP during the valid lifetime of $\kappa_{\mathsf{U,IDP}}$ at IDP.

Compromise of Long-term Secrets If U's password is compromised, $\mathcal{IMP}_{\mathsf{U}\rightarrow\mathsf{RP}}$ and $\mathcal{MITM}$ breaks are possible until the password is changed. If IDP's password verifier is compromised (e.g., its verifier database is stolen) then an offline password guessing attack is possible. Depending on the method selected for pre-authentication, obtaining the password verifier may also allow an attacker to successfully negate Assumption 1 and obtain a valid $\kappa_{\mathsf{U,IDP}}$.

If RP's private key is compromised by $\mathcal{M}$, then she will be able to mount $\mathcal{MITM}$ breaks in $\mathcal{S}_2$ and $\mathcal{S}_4$, however, it does not improve her ability to directly learn U's password. In Luau, authenticating as a legitimate RP within the context of the tunnel is only valuable in that it more effectively convinces U to initiate an authentication.

IDP can impersonate any of its users to any RP. If IDP's private key is compromised by $\mathcal{M}$, then she will be able to mount $\mathcal{IMP}_{\mathsf{IDP}\rightarrow\mathsf{RP}}$ breaks in $\mathcal{S}_3$ and $\mathcal{S}_4$. Also, depending on the method selected for pre-authentication, $\mathcal{M}$ may be able to use this private key to successfully attack the pre-authentication phase.

A significant benefit of Luau is that RP no longer has to maintain a user password database. This eliminates RP as a jumping off point to attack other relying parties, thus mitigating, in part, the dangers of the domino effect [44] of password reuse.

As all authentications of U must pass through IDP it is possible (in $\mathcal{S}_3/\mathcal{S}_4$) to provide a global "lock-out" which prevents logins as U to any RP.

Privacy IDP never actually learns if U authenticates to a specific RP since anyone with a valid ID_{U} and $ID_{\kappa_{\mathsf{U,IDP}}}$ can interact with IDP and obtain an encrypted $\kappa_{\mathsf{U,RP}}$. As discussed further in Section 4.6, this provides plausible deniability, which does not exist in other decentralized authentication approaches like OpenID.

With respect to privacy and RP, U's email address is typically disclosed when accounts are created and, as such, tracking is always possible. Luau does not preclude the mapping

of a local username to an email account on the back end, which limits the information that $\mathcal{E}$ learns when eavesdropping between U and RP in $\mathcal{S}_1$ and $\mathcal{S}_3$.

Covert Channels A covert channel uses another channel's bandwidth to transmit data without knowledge or consent. Covert channels take many different forms (e.g., steganography, timing between transmissions, text manipulation). As RP relays information between U and IDP (e.g., ID_{U}, n_{U}, $E_{\kappa^{enc}_{\mathsf{U,IDP}}}(\kappa_{\mathsf{U,RP}})$, $\text{MAC}_{\kappa^{mac}_{\mathsf{U,IDP}}}(\alpha)$) this could be leveraged to send covert data. This is of particular concern when Luau is used for wireless network authentication since it can provide $\mathcal{M}$ with some limited communication with an Internet-based counterpart even though she has not yet been authorized for network access.

4.5 Deployment and Implementation Issues

This section examines participant-specific deployment considerations as well as our prototypes. To build these prototypes we created `libluau`, a general purpose library written in Java. This library creates and processes the Luau protocol messages (for each participant) in binary and human-readable formats, but relies on the parent application to transport these messages to their intended destination.

4.5.1 Users

Client-side Support Existing systems (e.g., browsers, SSH clients, wireless supplicants) must be updated in order to support Luau. This represents a significant deployment hurdle, however, recent initiatives for decentralized authentication (e.g., CardSpace [19], DigitalMe [26], Higgins [41]) demonstrate the willingness of major players to provide client-side software support for the user.

Web browsers present an attractive avenue for incremental deployment, since, without intervention by browser vendors, client-side support for Luau can be provided via browser extensions or zero-footprint clients (e.g., Java Applets). To prevent $\mathcal{M}$ from circumventing

the login process, U's password must be delivered directly to the client-side software. This is not a problem for most client software (e.g., SSH clients, wireless supplicants), however, it is a significant problem for web browsers, which typically rely on a server-supplied HTML-based login page.

Entering login credentials via the browser's chrome avoids problems with malicious login pages and provides a consistent authentication experience across all sites. Modern web browsers already provide rudimentary support for prompting users for login credentials when they recognize a server's request for HTTP Basic or Digest authentication [29]. We advocate improving this interface (see Section 4.7).

Implementation We created two client-side prototypes using `libluau`: an extension to the Firefox browser and a signed Java Applet. Both use the PAKE protocol SRP for pre-authentication and rely on HTTP headers to recognize that a web site supports Luau as well as to transport the protocol messages.

4.5.2 Relying Parties

Relaying Pre-authentication Messages If U cannot directly communicate with IDP, it is reasonable to allow RP to relay the messages of the PAKE protocol required for pre-authentication between U and IDP because:

1. Messages are forwarded to only known identity providers of authorized users (ID_{U} dictates the IDP and RP dictates acceptable identifiers).
2. PAKE protocol messages are small in size and of a known composition.
3. For wireless authentication this is a lightweight alternative to allowing temporary network access to unauthenticated users.

Wireless Authentication Extensible Authentication Protocol (EAP) [3] provides a framework for deploying Luau as a wireless network authentication scheme. Although EAP

provides a mechanism to directly bind an authentication protocol to a secure tunnel, this binding does not provide any guarantees if the host is not properly authenticated (e.g., phishing, certificate trick). Luau's ability to protect against users negating the benefits of the tunnel is therefore desirable in this realm.

Due to the difficultly of adding new EAP methods to wireless networks, industry support is required to significantly impact this realm. While Luau takes into account the needs of wireless authentication, we advocate that initial deployments be to web-based relying parties as they represent an incremental, low-cost testbed for vetting this approach.

Implementation Relying party support is realized as a Java Servlet Filter that prevents access-restricted pages from being retrieved by unauthorized users. This filter uses HTTP headers to convey the messages created by `libluau` and to indicate that Luau logins are available.

4.5.3 Identity Providers

Mitigating the Availability/Reliability of IDP If IDP is unavailable, U cannot authenticate to RP. One approach to address this problem is to have a password backup or secret question/answer with RP. This method switches the traditional roles of emails and passwords as primary and secondary authentication mechanisms. If used, this would negate a key benefit of Luau as it reintroduces passwords.

A secondary email address, preferably in a completely different domain than the primary one, registered with RP during account creation, represents an elegant solution for the vast majority of users. This approach mitigates the problem of relying on a single third party at authentication time by providing a "backup" that can be used if the primary email provider is currently unavailable.

On Identifiers and Identity Providers Identifiers can come in many forms (e.g., email addresses, IM Handles, phone numbers, URLs, XRIs [61]). Luau assumes that these identi-

fiers are directly or indirectly coupled to identity providers. A direct coupling implies that the location of IDP can be derived from the identifier (e.g., an email address). An indirect coupling implies that a lookup service must be involved to determine the identity provider. For example, an XRI lookup service can be used to determine the current identity provider for a given XRI. A reliable coupling process is necessary to ensure that the correct identity providers are involved in the authentication process.

Becoming an identity provider may represent a significant increase in liability. Google, Yahoo!, and Microsoft (which have massive user bases to which they provide free email services) have already signaled their desire to support password-based decentralized authentication [63].

Implementation Our prototype identity provider is a Java Servlet that communicates via HTTP headers. As Luau is a simple request and response with respect to the identity provider, it does not have to maintain any relying party-specific state.

4.5.4 Delegation using Luau

Like other decentralized authentication systems, Luau is well suited to *server-based delegation*. Although this type of delegation has the potential to provide powerful, feature-rich delegation options, support for it is rare since it typically requires extensive modifications to existing applications.

Recall that delegation in password-based systems (see Section 2.5.2) is often accomplished by sharing the password and that the key benefit is that service providers do not need to be aware of, or support, delegation. Four significant problems with this approach were also identified:

1. Once a password is shared, it cannot be revoked without changing the password.
2. Using the delegator's password, the delegate can change the account password and, therefore, revoke access from the delegator.

3. A delegate can share the password with others without the approval of the delegator.

4. It is often hard to remember to whom the password has been delegated.

There are two approaches to provide client-based delegation in Luau. The first is similar to one-time delegation in SAW. By supplying a delegate with a valid $\kappa_{\mathsf{U,RP}}$ of the delegator to a specific relying party, the delegate can authenticate as the delegator to that relying party. The second approach provides a broader delegation and requires that the delegate be given a valid $\kappa_{\mathsf{U,IDP}}$ of the delegator; thus enabling the delegate to authenticate as the delegator to *any* relying party.

Unlike SAW, the process of obtaining a key from the delegator is easily automated and does not require direct interaction with the delegator at authentication time. A delegate's keys can be obtained by interacting with the delegator's identity provider. This approach leverages the ability, in Luau, for any entity to be a relying party. Specifically, the delegator's identity provider acts as a relying party in order to authenticate a delegate. Once a delegate has successfully authenticated it obtains a valid key that enables it to authenticate as the delegator[8].

Example 1: Obtaining $\kappa_{\mathsf{U,IDP}}$ Assuming Alice delegates access to Bob, the process by which Bob can authenticate as Alice to any relying party is as follows:

1. Bob authenticates to his identity provider and obtains $\kappa^{Bob}_{\mathsf{U,IDP}}$.

2. Using $\kappa^{Bob}_{\mathsf{U,IDP}}$ and Luau, Bob authenticates to Alice's identity provider and then obtains $\kappa^{Alice}_{\mathsf{U,IDP}}$.

3. Using $\kappa^{Alice}_{\mathsf{U,IDP}}$ and Luau, Bob authenticates to the relying party as Alice.

Example 2: Obtaining $\kappa_{\mathsf{U,RP}}$ Assuming Alice delegates access to Bob to Site X, the process by which Bob can authenticate as Alice to Site X is as follows:

[8]This approach also enables a group of users to share a single identifier. For example, a group member authenticates to the identity provider that controls the group identifier, the identity provider verifies group membership and then issues a new key.

1. Bob authenticates to his identity provider and obtains $\kappa_{\mathsf{U,IDP}}^{Bob}$.
2. Using $\kappa_{\mathsf{U,IDP}}^{Bob}$ and Luau, Bob authenticates to Alice's identity provider and specifies that he wants to authenticate as Alice to Site X.
3. Alice's identity provider authenticates as Alice to Site X. It then hands off $\kappa_{\mathsf{U,RP}}$ to Bob.
4. Using $\kappa_{\mathsf{U,RP}}$ Bob "resumes" the authenticated session[9] with Site X as Alice.

Addressing the Problems with Delegation in Password-based Systems The first problem is mitigated by the ability to immediately revoke existing $\kappa_{\mathsf{U,IDP}}$ keys (at the identity provider) given to delegates and to reject future authentications by that delegate to the delegator's identity provider.

With respect to problem 2, a solution akin to SAW is possible in Luau. In SAW, attempts to change the primary/secondary email address registered at a provider require an additional authentication of the user, which can be clearly distinguished from a normal login attempt. This enables users to configure their email forwarding rules to forward attempts by delegates to authenticate but retain attempts by delegates to change the registered email address at a specific provider, unless explicitly allowed by the delegator. In Luau, a special flag set by the relying party in its request to the identity provider could specify whether or not this authentication allows the user to change account login information at the relying party. The identity provider allows or disallows this authentication attempt based on the properties stored with a specific $\kappa_{\mathsf{U,IDP}}$.

The third problem with delegation in password-based systems is more difficult and is not addressed by SAW. If identity providers can be trusted to honestly disclose the delegation chain for the specified $\kappa_{\mathsf{U,IDP}}$ at each authentication attempt, then it is straightforward to address this problem in Luau. Nevertheless, the decentralized and independent nature of identity providers makes this a difficult assumption to enforce. Further study of this issue is left as future work.

[9]Session resumption in Luau is left as future work (see 4.7).

Problem 4 is mitigated since this approach does not require the delegator to share her password with a delegate and it also enables the delegator's identity provider to maintain an up-to-date record of delegations.

Delegation in Luau is more efficient than SAW since it avoids the latency inherent in email delivery and forwarding. This approach can easily accommodate lengthy delegation chains and, although the burden for obtaining the keys rests on the delegate, this process is easily automated. Luau also has the potential to be more secure and offer more features as the identity provider can take an active role in the delegation process. Interesting future work in this area involves the auditing and fine-grained polices that are possible if authentication of relying parties to identity providers was added to Luau.

4.6 Related Work

Authenticated key exchange for three parties began with Needham and Schroeder [59], which directly influenced Kerberos [50]. Bellare and Rogaway [12] provide a formal security model and the first provably secure symmetric-key based system (3PKD).

Abdalla et al. [2] is the first provably secure password-based protocol in a 3-party setting, where two parties each share a password with a trusted server. Recent research [53, 42, 21, 43] in this vein has examined the use of these protocols as a decentralized authentication system. To the best of our knowledge, all password-based 3-party key exchanges require that relying parties share a secret with the user's identity provider. In contrast, the loose coupling between relying parties and identity providers in Luau does not require them to pre-establish shared secrets.

There are a variety of PKI-based systems that provide decentralized user authentication. Certificate-based client authentication in systems like TLS [24] and EAP-TLS [4] enable clients, certified by trusted third parties, to demonstrate ownership of a private key to authenticate to servers. PKI systems, in general, have burdensome configuration and usage requirements [35].

OpenID [62] is a decentralized authentication system that uniquely identifies users via URLs. Brands [16] has complied a thorough collection of drawbacks, which we summarize here[10]:

Security Reliance on browser redirects invites phishing. Browser vulnerabilities and cross-site scripting can hijack a user's authenticated state with an identity provider. The identity provider can impersonate all of its users. Compromising the OpenID account gives access to all the user's accounts.

Privacy Identity providers can track all websites you use.

Trust Why should an identity provider be trusted to vouch for an identity?

Usability URIs should identify websites, not people. Redirection creates a disjointed user experience. If the identity provider is down, you cannot log in.

Adoption Many organizations are becoming identity providers, but their services do not accept OpenIDs.

As mentioned previously, Luau eschews redirects in favor of an in-band approach and does not maintain an authenticated state with identity providers using cookies. While impersonation by the identity provider and the reuse of the same login credentials across multiple domains can be seen as putting all of one's eggs in one basket, in many cases, the eggs are already in that basket. With control of a user's email account, or simply the ability to eavesdrop incoming email messages, current attackers can obtain access to all the other accounts whose logins credentials can be reset using that email account. Recall that Luau is only a suitable primary authentication scheme at relying parties that trust email providers to authenticate users on their behalf. Luau plugs many security leaks in EBIA, while maintaining its convenience.

Luau provides a "plausible deniability" approach to privacy as identity providers never actually learn if a user successfully authenticates. With respect to trust and usability,

[10]This summary also includes potential patent issues not discussed here.

Luau leverages the existing trust placed in email providers as well as existing, ubiquitous identifiers. Lastly, with respect to adoption, Luau's potential to thwart password phishing and limit the ability for users to negate secure tunnels are attractive features for relying parties in addition to the traditional benefits of decentralized authentication.

Liberty [54] and Shibboleth [74] are alternative approaches to password-based decentralized authentication and can also assert additional attributes about their users. They resolve many of the issues faced by OpenID through legal/business agreements, but still rely on browser redirects, and therefore are still susceptible to password phishing, and are not well suited for wireless authentication.

CardSpace [19] is an OS-based authentication interface that provides a consistent, secure UI across all authentications. It is built on a framework in which the relying party communicates with the identity provider through the user. Unlike Luau, this precludes this approach from taking full advantage of unauthenticated third party identity providers (which EBIA has demonstrated to be useful) and complicates its compatibility with wireless authentication.

Simple Authentication for the Web (SAW) [81] leverages EBIA as a primary login mechanism. It addresses the vulnerability of EBIA to passive attack through a token splitting scheme and relies on unmodified email providers. Its key drawbacks are the usability impacts incurred by the latency of email delivery and its vulnerability to active attack when TLS-protected SMTP is not employed.

4.7 Conclusions and Future Work

Email providers are *the* de facto identity providers on the Internet. They have already demonstrated their worth and have the potential to become even more valuable in this respect to both their clients and to relying parties. Luau enhances these providers' ability to conveniently and securely authenticate their users to relying parties by providing a

lightweight, scalable, and adaptable protocol that unifies authentication across the application and network layers and dramatically reduces the risk of password phishing.

EBIA has demonstrated the convenience and the utility of its trust model, even if email providers are unauthenticated. Luau leverages this lightweight relationship between service providers and email providers to reduce the need for site-specific passwords. As relying parties typically require an email address during account creation, Luau represents a practical reuse of existing login credentials while maintaining the portability and convenience of passwords.

In addition to convenience, Luau provides assurances tailored to the deployment scenario. Luau does not disclose a user's password during the authentication process, even when interacting with a malicious relying party. Luau also thwarts several MITM attacks without requiring the relying party to authenticate to the user. Before phishers/pharmers can impersonate users to a relying party, Luau requires that they perform a synchronized attack in which they first impersonate that user's identity provider to the relying party.

Future work includes extending Luau to support attribute assertions from the identify provider and conducting usability studies with real-world deployments. As Luau is designed to operate where secure tunnels are not feasible, integration with other technologies designed for these scenarios is interesting. For example, after a secure login, SessionLock [6] prevents session hijacking when using unsecured channels. Also, using Luau in a peer-to-peer setting (e.g., a peer is a relying party) may enable new interactions and services.

Enabling session resumption between users and relying parties using $\kappa_{\mathsf{U,RP}}$ also merits further research. Ideally, a special flag could be used to indicate that a session (uniquely identified via the n_{U} and n_{RP} used in the initial authentication) should be resumed. The modified 3PKD process would then be skipped and the MAP1 portion of Luau, using the existing $\kappa_{\mathsf{U,RP}}$ and new nonces, would then be used.

Acknowledgments. This research was supported by funding from the National Science

Foundation under grant no. CCR-0325951, prime cooperative agreement no. IIS-0331707, and The Regents of the University of California.

Chapter 5

pwdArmor:
Protecting Conventional Password-based Authentications

T. W. van der Horst and K. E. Seamons. pwdArmor: Protecting Conventional Password-based Authentications. *24th Annual Computer Security Applications Conference (ACSAC)*, Anaheim, CA, December 2008.

Additional Content: Figure 5.1 has been restored to the larger image submitted in the original submission. Also, Section 5.4 has been expanded with both text and Table 5.1 to improve the summary of the security analyses of pwdArmor and conventional password protocols.

Abstract

pwdArmor is a framework for fortifying conventional password-based authentications. Many password protocols are performed within an encrypted tunnel (e.g., TLS) to prevent the exposure of the password itself, or of material for an offline password guessing attack. Failure to establish, or to correctly verify, this tunnel completely invalidates its protections. The rampant success of phishing demonstrates the risk of relying solely on the user to ensure that a tunnel is established with the correct entity.

pwdArmor wraps around existing password protocols. It thwarts passive attacks and improves detection, by both users and servers, of man-in-the-middle attacks. If a user is tricked into authenticating to an attacker, instead of the

real server, the user's password is never disclosed. Although pwdArmor does not require an encrypted tunnel, it gains added protection from active attack if one is employed; even if the tunnel is established with an attacker and not the real server. These assurances significantly reduce the effectiveness of password phishing. Wrapping a protocol with pwdArmor requires no modification to the underlying protocol or to its existing database of password verifiers.

5.1 Introduction

In the typical password-based login (e.g., HTML form, SSH "keyboard-interactive") the user's plaintext password is sent to the server, which uses it to compute a verifier. This verifier is then compared to the copy of the verifier stored by the server and, if they match, the authentication succeeds. It is customary to establish a server-authenticated, encrypted tunnel (e.g., TLS with a server certificate, SSH transport layer) to prevent passive observation of the plaintext password as it is sent to the server and to protect the resulting session from eavesdroppers and hijackers.

There are several methods that attackers use to circumvent this encrypted tunnel. The simplest is to not create it and hope that the user does not notice. This is the bread-and-butter of phishers around the globe[1]. A web page is made to appear exactly like the "real" server, but it is not sent over HTTPS so the server is never authenticated to the client (as it usually would have been). An attempt to "login" to the fake site discloses the user's password to the phisher.

A variation on this attack, hereafter referred to as the *certificate trick*, involves tricking a user into accepting the wrong certificate for the real server. There are two main avenues to mount this attack. First, the attacker sends the phishing page over HTTPS[2] using a valid certificate issued to the phishing site, not the real site. As the certificate is valid for this site,

[1]The Anti-Phishing Working Group (APWG) reports that 99.23% of phishing occurs over plain HTTP [7].

[2]APWG estimates that 0.28% of phishing occurs over HTTPS [7].

no warnings are issued by the browser. Second, the attacker creates a self-signed certificate and assumes the user will ignore all browser warnings and accept this certificate as if it were from the real server[3].

SSH demonstrates the useful practice of reusing existing password verifiers; it can rely on user account information that is created and managed externally. As SSH relies on a server-authenticated, encrypted tunnel, the attacker must trick the user into accepting her public key in lieu of the real server's key in order to perform the certificate trick. If successful, the attacker learns the user's password. SSH uses a key continuity approach that informs users if the server's public key has changed since their last login.

In the context of their verifiers, password authentication mechanisms fall into two categories: password-equivalent and password-dependent. In a password-equivalent protocol (e.g., HTTP Digest authentication [29], Kerberos, EKE [14]) the server's password verifier is, for all intents and purposes, equivalent to the user's password. In a password-dependent protocol (e.g., HTTP Basic authentication [29], S/Key [37], SRP [87]) the verifier cannot be used directly to impersonate the client, as it is dependent on the plaintext password, but it can be useful to an offline guessing attack. We believe that password-equivalent verifiers have an inherent risk (especially from malicious insiders) that can and should be avoided, particularly when many of these verifiers can readily be used in a password-dependent fashion.

Password-authenticated key exchange (PAKE) protocols (e.g., EKE, SRP), do not require encrypted channels to protect the password and have the added benefit of establishing a mutually-authenticated session key that can be used to protect a subsequent session. Unlike conventional protocols within encrypted tunnels, these protocols cannot readily use existing password verifiers in a password-dependent manner (see Section 5.2.3). Also, current PAKE protocols do not provide privacy protection to the user's identity.

[3]These attacks are not limited to phishers. Cain and Able [17] is a popular tool that uses ARP poisoning to force all local network traffic to flow through the tool. It also provides an automated TLS man-in-the-middle attack using a self-signed certificate.

Our Contributions pwdArmor is a framework for leveraging conventional password protocols, and existing password verifier databases, to create PAKE protocols. Unlike other PAKE protocols, pwdArmor is neither password-equivalent nor password-dependent, rather it preserves this characteristic from its underlying password protocol. Also, pwdArmor can provide privacy protection to the user's identity, with or without an external encrypted tunnel.

pwdArmor treats the server authentication of an encrypted tunnel as an added bonus rather than a critical hinge of its security. Even if a user is tricked into performing the protocol directly with an attacker, the user's password is never exposed.

As a proof of concept, we used pwdArmor to wrap HTTP Basic authentication and the One-Time Password (OTP) protocol [38] (a derivative of S/Key). The client-side is realized as an extension to Firefox and also as a signed applet. The server-side is implemented as Servlet Filter in Tomcat.

Although some verifier databases contain essentially a password-dependent verifier (e.g., hash of the password), the selected conventional password protocol may use it as a password-equivalent verifier (e.g., HTTP Digest authentication, MS-CHAPv2 [89]). Using pwdArmor, and a different conventional password protocol, these verifier databases can be leveraged in a more secure, password-dependent manner, potentially eliminating the need for the original password-equivalent protocol. We demonstrate this by replacing HTTP Digest authentication with pwdArmor and HTTP Basic authentication.

Paper Outline Section 5.2 lays the foundation for pwdArmor. Section 5.3 presents the pwdArmor framework. Section 5.4 analyzes its security. Section 5.5 considers deployment issues. Section 5.6 discusses the prototype implementation. Section 5.7 examines related work. Section 5.8 contains conclusions and future work.

5.2 Foundation

In this paper, user (U) and host (H) desire to mutually authenticate and optionally establish a key that will provide forward secrecy. We assume that U has a password pwd_{U} and that H stores a verifier pwd_{U}^{ver} and associated information α, such that $pwd_{\mathsf{U}}^{ver} = Verifier(pwd_{\mathsf{U}}, \alpha)$. α contains the additional information (e.g., salt, realm, index), if any, required to create the verifier from the password.

5.2.1 Threat Model

This section specifies the threat model used to compare existing conventional password protocols and pwdArmor. This model defines the likely deployment scenarios, the common methods of attack, and the attackers.

Target Scenarios We target two common scenarios, which, based on the properties of their communications channels, are categorized as follows:

$\mathcal{S}_{clear}$ An unsecured channel (e.g., HTTP) is used for all communications.

$\mathcal{S}_{tunnel}$ A server-authenticated, encrypted tunnel (e.g., HTTPS, SSH) is used for all communications.

A third scenario, which adds server authentication to $\mathcal{S}_{clear}$, is the least likely to be used in practice and will not be specifically addressed in this paper due to lack of space.

Attacks Password protocols are designed such that pwd_{U} is required to impersonate U to H. Obtaining pwd_{U} constitutes a $\mathcal{PWD}$ break. As the number of potential passwords is relatively small (especially when compared to the size of keys typically used in cryptographic protocols), an attacker's ability to correctly guess the password is of particular concern. There are two approaches to password guessing: online and offline.

Online guessing attacks repeatedly invoke the protocol with H while varying the password. The best protection an online password protocol can provide is to ensure that attackers cannot obtain an advantage against the protocol greater than online guessing.

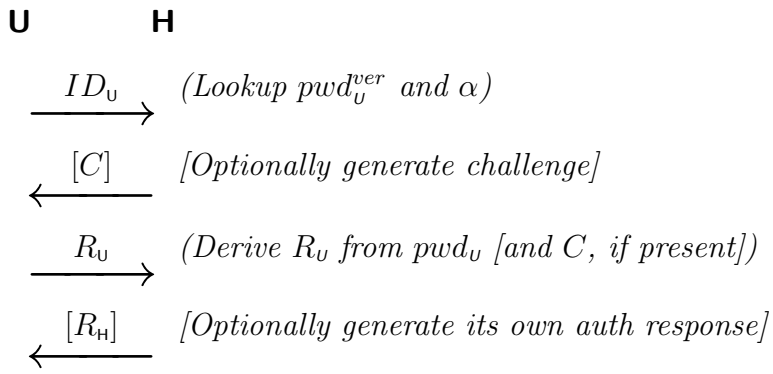
Offline guessing attacks are more efficient than online attacks, but they require verification material (i.e., the result of a known deterministic function of the password) to determine if a guess is correct. Obtaining this material constitutes a $\mathcal{LEAK}$ break, which is a step towards a $\mathcal{PWD}$ break. In practice, many organizations consider it acceptable that an attacker cannot compromise a protocol (e.g., Kerberos) with an advantage greater than offline-guessing. In these situations, services often dictate the minimum strength of user passwords.

If the attacker can insert herself between U and H she can become a *man-in-the-middle* (MITM). By simply relaying the authentication protocol's messages between the two unsuspecting parties, she can, after the authentication is complete, hijack U's session with H. This constitutes a $\mathcal{MITM}$ break, which enables the attacker to impersonate U without a $\mathcal{PWD}$ break. In practice, there are three common methods to perform this attack: 1) Routing-MITM, an attacker controls a router between U and H or has tricked traffic to route through her (e.g., ARP poisoning); 2) Pharming-MITM, DNS is poisoned so that lookups return a network address controlled by the attacker; and 3) Phishing-MITM, U is tricked into connecting to the attacker in lieu of the legitimate host (e.g., clicks a link in a phishing email).

Attackers Two attackers are considered in this model. Eve ($\mathcal{E}$) is a passive eavesdropper whose goals are $\mathcal{PWD}$ and $\mathcal{LEAK}$ breaks. Mallory ($\mathcal{M}$) is an active attacker whose goals are $\mathcal{PWD}$, $\mathcal{LEAK}$, and $\mathcal{MITM}$ breaks. $\mathcal{E}$ is limited to observing the normal interactions between U and H. $\mathcal{M}$ can observe, inject, modify, delay, destroy, and replay messages as well as create multiple concurrent sessions with any other party.

5.2.2 Conventional Password Protocols

Conventional password protocols consist of these logical message elements:



In the first round, U discloses her identifier ID_U and H responds with an optional challenge C, which may be dependent on ID_U. U then sends her authentication response R_U, to which H can optionally respond with its own authentication response R_H. If C and R_H are not required, then the protocol can be condensed into a single message with two logical message elements: $U \rightarrow H : ID_U, R_U$.

R_U enables U to demonstrate knowledge of pwd_U to H. Based on the characteristics of R_U (and assuming the absence of an encrypted tunnel to protect it) we classify password protocols as follows:

Type-0 R_U is always the same and therefore replayable. A challenge by the server is typically not required. Examples include responses that contain the password itself (e.g., HTML forms, SSH "keyboard-interactive").

Type-1 R_U is *not* replayable, but it can be used to mount an offline password guessing attack. A challenge by the server is required to construct R_U. Examples include conventional challenge/response protocols (e.g., HTTP Digest authentication) and some one-time password protocols (e.g., OTP [38]).

Type-2 R_{U} is *not* replayable and it does *not* contain material for an offline password guessing attack. Specifically, although R_{U} may contain some form of the password it also involves a large, unobservable, session-specific secret that significantly complicates an offline attack as password guesses must also correctly guess the value of the session secret. Examples include PAKE protocols like SRP and SPEKE [45].

An orthogonal characteristic to this classification is whether or not pwd_{U}^{ver} is equivalent to pwd_{U} (i.e., password-dependent vs. password-equivalent)[4]. Another orthogonal characteristic is whether or not ID_{U} is required to generate C. For example, in HTTP Digest authentication the challenge (a server-generated nonce) is independent of ID_{U} whereas the challenge in OTP is identity-dependent because it contains the user's seed and hash count.

In our target deployment scenarios both Type-0 and Type-1 protocols rely on the following assumption:

Assumption 1 *U and H assume that messages passed between them will not be observed, or modified, by an attacker.*

The strength of this assumption is dictated by the scenario. In $\mathcal{S}_{clear}$, this assumption equates to "security by obscurity." In practice, this is an acceptable risk for many low-value sites and services. In $\mathcal{S}_{tunnel}$, this assumption is only valid when the tunnel is correctly established with the legitimate host. Recall that phishers regularly trick users into negating this assumption.

Attacking Type-0/1 Protocols If Assumption 1 holds then $\mathcal{PWD}$, $\mathcal{LEAK}$, and $\mathcal{MITM}$ breaks cannot occur in Type-0/1 protocols. If Assumption 1 does not hold, then the following attacks are possible:

[4]In PAKE protocols password-equivalent and password-dependent are called balanced and augmented, respectively.

	Type-0	Type-1
$\mathcal{E}$	$\mathcal{PWD}$, $\mathcal{LEAK}$	$\mathcal{LEAK}$
$\mathcal{M}$	$\mathcal{PWD}$, $\mathcal{LEAK}$, $\mathcal{MITM}$	$\mathcal{LEAK}$, $\mathcal{MITM}$

5.2.3 Why not just use Type-2?

pwdArmor augments Type-0/1 protocols with the properties of a Type-2 protocol. As both pwdArmor and Type-2 protocols require new client-side software, what advantages does pwdArmor offer over a Type-2 protocol? The primary advantage is that pwdArmor can reuse existing (legacy) password verifier databases while maintaining their password-dependence.

Current password-equivalent Type-2 protocols (e.g., EKE) can reuse existing verifier databases, however, doing so eliminates any password-dependent benefits these verifiers may have enjoyed with their original password protocol. Such a transition negates a significant benefit of the original protocol, since the verifier, if it is stolen, can now immediately be used to impersonate the user.

Current password-dependent Type-2 protocols (e.g., SRP) require databases of their own specialized password verifiers in order to preserve password-dependence. Relying on existing verifiers makes these protocols password-equivalent, thus negating their password-dependent benefits. Requiring new verifiers introduces a significant deployment overhead and potentially breaks compatibility with legacy systems that require the old verifier databases.

A second advantage is that pwdArmor provides optional privacy protection to the identity of the user.

5.3 pwdArmor

Goals The primary mission of a password protocol is to ensure that pwd_{U} is required to impersonate U to H. In addition to this aim, pwdArmor is designed to meet the following goals:

1. Eliminate $\mathcal{PWD}$ breaks.

2. Limit the damage when Assumption 1 does not hold.

3. Detect MITM attacks.

4. Allow H to dictate the difficulty of $\mathcal{MITM}$ breaks.

The first goal is to limit the avenues for users to inadvertently disclose their passwords to phishers, eavesdroppers, and other attackers.

The second goal is motivated by the effectiveness of phishers in negating Assumption 1, even if H employs server-authenticated, encrypted tunnels to strengthen this assumption. As users can be easily tricked into negating a tunnel's benefits, pwdArmor uses a secure tunnel, if available, in the following capacities: 1) Improved identification of H to detect potential MITM attacks before they happen and to prevent $\mathcal{LEAK}$ breaks by $\mathcal{M}$; 2) Protect, after a successful authentication, the resulting session; and 3) Provide privacy protection to ID_{U} during the authentication.

The third goal is based in the observation that, in both $\mathcal{S}_{clear}$ and $\mathcal{S}_{tunnel}$, it is possible to detect a MITM attack by using information (e.g., domain names, network addresses, digital certificates used in a tunnel's creation) that is readily accessible to each party. Adding, or strengthening as the case may be, server authentication also helps assure users that they are communicating with the desired host and not just a phisher that accepts any password and then tries to elicit additional information from their victims.

The fourth goal captures the intuitive idea that H's choice of $\mathcal{S}_{tunnel}$ instead of $\mathcal{S}_{clear}$ should complicate $\mathcal{M}$'s ability to successfully achieve a $\mathcal{MITM}$ break. This idea is not reflected in existing tunnels (e.g., TLS, SSH) as they rely solely on users to detect MITM attacks and, as such, hosts have no say in the difficultly of $\mathcal{MITM}$ breaks.

For the purposes of evaluating pwdArmor, $\mathcal{E}$ is considered successful if she obtains $\mathcal{LEAK}$ or $\mathcal{PWD}$ breaks with a non-negligible advantage over online guessing. With respect to $\mathcal{M}$, $\mathcal{LEAK}$ breaks are acceptable and therefore she is considered successful only if she obtains a non-negligible advantage over offline guessing. Additionally, $\mathcal{MITM}$ breaks are

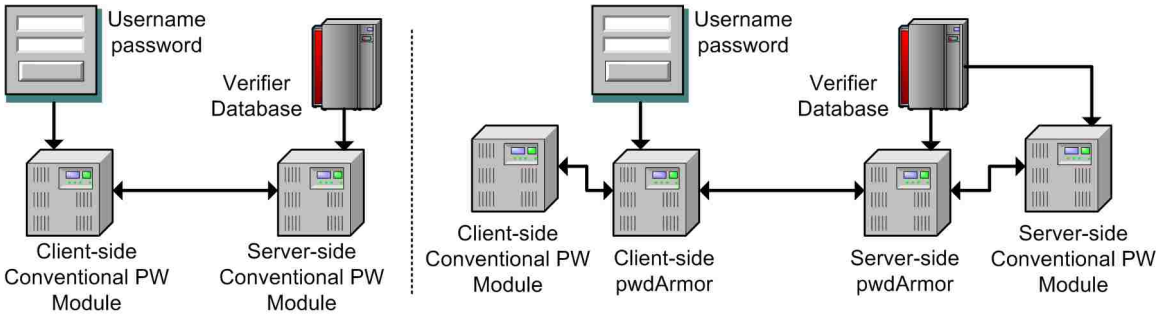


Figure 5.1: In conventional password protocols (left) the client- and server-side modules directly interact. pwdArmor uses middleware to augment conventional password protocols (right). Both the server-side pwdArmor and conventional password modules require access to the verifier database. pwdArmor uses the verifier to secure the user's authentication response, while the conventional password protocol uses it to verify the password.

unacceptable, with the exception of the routing-MITM attack when H uses $\mathcal{S}_{clear}$ as this attack in this scenario is virtually undetectable.

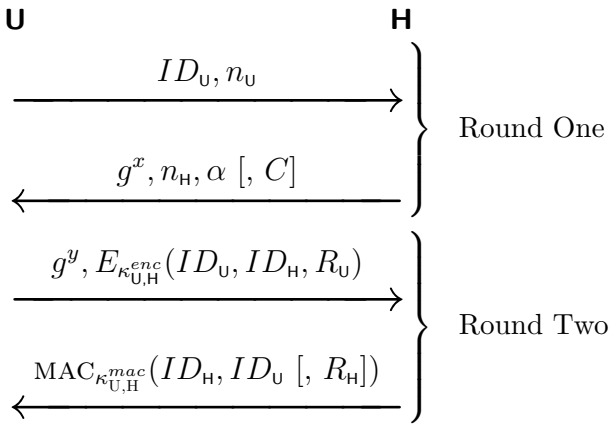
High Level Approach pwdArmor leverages middleware (see Figure 5.1) to wrap unmodified Type-0/1 protocols and bind them to an external secure tunnel, if present. pwdArmor encrypts U's authentication response R_{U} to ensure its confidentiality and integrity. This encrypted message also contains the identifying host information (as observed by U), hereafter referred to as ID_{H}, in order to facilitate the detection, by H, of MITM attacks. As this information is scenario-dependent, H can also detect if U is using pwdArmor in a different scenario than expected (e.g., H requires logins in $\mathcal{S}_{tunnel}$, but $\mathcal{M}$ has tricked U into using $\mathcal{S}_{clear}$).

The key ($\kappa^{enc}_{\mathsf{U},\mathsf{H}}$) used to encrypt this message is composed of two components: 1) $\kappa_{\mathsf{U},\mathsf{H}}$, which is created using a Diffie-Hellman (DH) key exchange [25]; and 2) pwd^{ver}_{U}. The use of $\kappa_{\mathsf{U},\mathsf{H}}$ ensures that $\mathcal{E}$ cannot be successful without compromising the assumptions of DH. As DH is subject to a MITM attack, $\mathcal{M}$ can force U and H to derive different values for $\kappa_{\mathsf{U},\mathsf{H}}$, each known to $\mathcal{M}$. In this event, $\mathcal{LEAK}$ breaks are possible, however, the pwd^{ver}_{U} component of $\kappa^{enc}_{\mathsf{U},\mathsf{H}}$ ensures that she cannot decrypt the message and obtain R_{U} or modify ID_{H}. Without modifying ID_{H}, it is unlikely a $\mathcal{MITM}$ break will be successful.

Once the authentication succeeds, then a separate key derived from $\kappa_{\mathsf{U,H}}$ is suitable for use as a mutually-authenticated key which provides forward secrecy.

5.3.1 Framework

The pwdArmor messages are illustrated below:



Round One U begins by submitting her identifier and a nonce n_{U}. H responds with its DH public parameter g^x and its own nonce n_{H}. H's response also includes α (how to derive pwd^{ver}_{U} from pwd_{U}) as well as the challenge, if any, supplied by the underlying password protocol.

Round Two Using the elements from H's message, U derives $\kappa^{enc}_{\mathsf{U,H}}$. Specifically:

$$\begin{aligned} \kappa_{\mathsf{U,H}} &= PRF_r(g^{xy}); r = n_{\mathsf{U}} \| n_{\mathsf{H}} \\ \kappa^{enc}_{\mathsf{U,H}} &= PRF_{\kappa_{\mathsf{U,H}}}(pwd^{ver}_{\mathsf{U}}, \text{“}enc\text{”}) \end{aligned}$$

where PRF is a family of pseudorandom functions (e.g., HMAC [60] is commonly used as a PRF). U then returns its encrypted response, along with its own DH public parameter g^y, to H. Including ID_{U} in $E_{\kappa^{enc}_{\mathsf{U,H}}}(\cdot)$ allows U to securely assert her identifier to H. As the

original transmission of ID_{U} is sent without any cryptographic protection, $\mathcal{M}$ could modify it so that the user logs into a different account (this assumes that these accounts share the same password).

In the final message H demonstrates to U that it knows both pwd_{U}^{ver} and $\kappa_{\mathsf{U},\mathsf{H}}^{mac} = PRF_{\kappa_{\mathsf{U},\mathsf{H}}}(pwd_{\mathsf{U}}^{ver}, \text{“}mac\text{”})$ through the use of a message authentication code (MAC) function (e.g., HMAC). This message must be sent only if the user authentication succeeds, otherwise, it constitutes a $\mathcal{LEAK}$ break as the initiator of an authentication knows $\kappa_{\mathsf{U},\mathsf{H}}$. After U verifies H's message, the key $\kappa_{\mathsf{U},\mathsf{H}}^{other} = PRF_{\kappa_{\mathsf{U},\mathsf{H}}}(\text{“}other\text{”})$ can be used by other applications as a mutually authenticated key that provides forward secrecy.

Identifying H ($\mathbf{ID_H}$) The scenario determines ID_{H}:

$\mathcal{S}_{clear}$ ID_{H} = Network identifiers (e.g., domain name, network address).

$\mathcal{S}_{tunnel}$ ID_{H} = H's certificate and network identifiers.

Privacy Protection If $\mathcal{S}_{tunnel}$ cannot be used by H, but ID_{U} must be kept secret from $\mathcal{E}$, then ID_{U} can be optionally encrypted using $\kappa_{\mathsf{U},\mathsf{H}}^{priv} = PRF_{\kappa_{\mathsf{U},\mathsf{H}}}(\text{“}priv\text{”})$ before it is sent to H (this is a well known approach to providing privacy when DH is involved). There are two different situations to consider:

Challenge and α are *not* Identity-dependent ID_{U} is removed from the first message, encrypted using the key $\kappa_{\mathsf{U},\mathsf{H}}^{priv}$, and then added to the third message. The encrypted ID_{U} must be external to the contents of $E_{\kappa_{\mathsf{U},\mathsf{H}}^{enc}}(\cdot)$ as the key $\kappa_{\mathsf{U},\mathsf{H}}^{enc}$ is dependent on pwd_{U}^{ver}, which cannot be retrieved by H before ID_{U} is known.

Challenge or α is Identity-dependent ID_{U} is again encrypted with $\kappa_{\mathsf{U},\mathsf{H}}^{priv}$, and as either C or α is identity-dependent and *may* leak identifying information, both of these values are also encrypted with $\kappa_{\mathsf{U},\mathsf{H}}^{priv}$ before they are sent to U. Since H cannot know the correct values for C and α before it learns ID_{U}, an extra round of messages is required since

H's first response cannot include these values and U cannot send $E_{\kappa^{enc}_{\mathsf{U},\mathsf{H}}}(\cdot)$ until these values are received.

5.4 Security Analysis

Attacking pwdArmor As with conventional password protocols, if Assumption 1 holds the $\mathcal{PWD}$, $\mathcal{LEAK}$, and $\mathcal{MITM}$ breaks cannot occur in pwdArmor. If Assumption 1 does not hold, then the following attacks are possible (note that both Type-0 and Type-1 protocols have the same assurances when used with pwdArmor):

	$\mathcal{S}_{clear}$	$\mathcal{S}_{tunnel}$
$\mathcal{E}$	None	None
$\mathcal{M}$	$\mathcal{LEAK}$, $\mathcal{MITM}^*$	$\mathcal{LEAK}$

* Only if $\mathcal{M}$ uses a routing-MITM attack

The remainder of this section provides arguments which justify the claims made above and explores the impact of compromised session and long-term secrets.

Thwarting $\mathcal{PWD}$ breaks Neither $\mathcal{E}$ nor $\mathcal{M}$ can mount successful $\mathcal{PWD}$ breaks as the conventional password protocol authentication response R_{U} is encrypted using $\kappa^{enc}_{\mathsf{U},\mathsf{H}}$, which is based, in part, on pwd^{ver}_{U}. In effect, an attacker must know the password verifier before she can attack this protocol to obtain pwd_{U}. Note that while obtaining pwd^{ver}_{U} constitutes a $\mathcal{LEAK}$ break, a $\mathcal{LEAK}$ break does not necessarily mean that the attacker has obtained pwd^{ver}_{U}.

Limiting $\mathcal{LEAK}$ breaks pwdArmor relies on a DH key exchange to generate $\kappa_{\mathsf{U},\mathsf{H}}$ and to prevent its passive observation. We assume the presence of pre-established, well-known, well-tested generator/prime pairs (g, p) for a specified key size (e.g., [49, 52]). This prevents $\mathcal{M}$ from injecting a pair for which she can compute discrete logs[5]. H ultimately decides on

[5]For example, $p = 3$, $p - 1$ is composed of only small prime factors, and many cases of $p = 2^n$ [73].

which pair to use for a specified run of the framework. As R_{U} is encrypted using $\kappa^{enc}_{\mathsf{U,H}}$, a key that is based, in part, on $\kappa_{\mathsf{U,H}}$, $\mathcal{E}$, who is limited to eavesdropping, cannot mount successful $\mathcal{LEAK}$ breaks as each password guess must also correctly guess the value of $\kappa_{\mathsf{U,H}}$.

The deployment scenario dictates the difficultly for $\mathcal{M}$ to mount $\mathcal{LEAK}$ breaks. In $\mathcal{S}_{clear}$, $\mathcal{M}$ can perform a DH-MITM attack by substituting the value of g^x sent by H with its own value $g^{x'}$. When $\mathcal{M}$ receives g^y from U she can compute $\kappa_{\mathsf{U,H}}$. Since she knows the DH component of $\kappa^{enc}_{\mathsf{U,H}}$ she can verify offline password guesses by first using the guess to compute $pwd^{ver'}_{\mathsf{U}}$, deriving $\kappa^{enc}_{\mathsf{U,H}}$, and then checking to see if the decryption of $E_{\kappa^{enc}_{\mathsf{U,H}}}(\cdot)$ contains the password guess. Note that a DH-MITM attack destroys the ability for U and H to establish the same value for $\kappa_{\mathsf{U,H}}$ and therefore relaying the encrypted R_{U} will be detected by H as it cannot successfully decrypt $E_{\kappa^{enc}_{\mathsf{U,H}}}(\cdot)$.

In $\mathcal{S}_{tunnel}$ a $\mathcal{LEAK}$ break is only possible if U creates the tunnel with $\mathcal{M}$ instead of H (i.e., succumbs to the certificate trick). In this case, $\mathcal{M}$ will be able to perform the $\mathcal{LEAK}$ break in the same manner described above for $\mathcal{S}_{clear}$.

Preventing $\mathcal{MITM}$ breaks Although a DH-MITM attack enables $\mathcal{LEAK}$ breaks, it destroys $\mathcal{M}$'s ability to mount $\mathcal{MITM}$ breaks as both U and H learn that something is amiss. H knows this because it is unable to decrypt R_{U} since its value for $\kappa_{\mathsf{U,H}}$ is different from U's value. U learns of this as $\mathcal{M}$ cannot produce a valid $\text{MAC}_{\kappa^{mac}_{\mathsf{U,H}}}(\cdot)$, due to her lack of knowledge of pwd^{ver}_{U}. Due to these factors, given a single run of the protocol to attack, $\mathcal{M}$ can choose to attempt $\mathcal{LEAK}$ breaks or $\mathcal{MITM}$ breaks, but not both.

Recall that server-authenticated tunnels (e.g., TLS, SSH) rely on users to detect MITM attacks. pwdArmor adds the ability for H to detect MITM attacks, which were missed by U, by including ID_{H} in $E_{\kappa^{enc}_{\mathsf{U,H}}}(\cdot)$. Therefore, in order to achieve $\mathcal{MITM}$ breaks, $\mathcal{M}$ must conceal its presence from both U and H. Again, the difficultly of this masquerade depends on the scenario.

In $\mathcal{S}_{clear}$ conventional phishing/pharming-MITM attacks, missed by U, should be detected by H since these attacks have difficultly masking all of the network identifiers that compose ID_{H}. A routing-MITM attack in this scenario does mask its presence effectively and therefore cannot currently be detected by pwdArmor. Note that $\kappa_{\mathsf{U},\mathsf{H}}^{other}$ may be used to establish a secure tunnel after the authentication completes, and this tunnel would prevent even a routing-MITM attack from achieving a $\mathcal{MITM}$ break.

In $\mathcal{S}_{tunnel}$ the entity with which U is connected is identified by the certificate used in the server-authentication of the tunnel. In order for H not to detect a MITM attack, $\mathcal{M}$ would have had to authenticate to U as if it was the legitimate H (e.g., in TLS/SSH this requires a proof of ownership of H's private key).

Compromise of Session Secrets If U's or H's DH exponent (x or y, respectively) is compromised, then $\mathcal{M}$ would be able to compute $\kappa_{\mathsf{U},\mathsf{H}}$ from a recorded session and therefore perform a successful $\mathcal{LEAK}$ break.

The nonces (n_{U}, n_{H}) in this protocol are used in the same manner as they are in IKEv2 [47] to create a seed key ($\kappa_{\mathsf{U},\mathsf{H}}$) from the result of the DH key exchange (g^{xy}). These nonces also enable U and H to reuse their DH parameters across multiple sessions (e.g., for performance reasons) while ensuring that each session is unique. Note that the forward secrecy of $\kappa_{\mathsf{U},\mathsf{H}}$ is maintained only between sessions where the DH values are not reused.

These nonces also introduce session-specific randomness from each participant to ensure the freshness of the keys used in this session. Also, deriving session-specific, purpose-specific symmetric keys using $\kappa_{\mathsf{U},\mathsf{H}}$ helps ensure that if one of these keys is ever compromised, the effects are limited to the scope of that key.

It is important to observe that since $\kappa_{\mathsf{U},\mathsf{H}}^{other}$ is not used as keying material in the resulting session, forward secrecy of that session is dependent on whether the underlying $\mathcal{S}_{tunnel}$ method provides forward secrecy.

<table>
<tr><th></th><th colspan="2">Conventional Password Protocols</th><th>pwdArmor</th></tr>
<tr><td>Tunnel
(correctly established)</td><td colspan="2">None</td><td>None</td></tr>
<tr><td>No
Tunnel</td><td colspan="2"><table><tr><th></th><th>Type-0</th><th>Type-1</th><th>Type-2</th></tr><tr><td>$\mathcal{E}$</td><td>$\mathcal{PWD}$,
$\mathcal{LEAK}$</td><td>$\mathcal{LEAK}$</td><td>None</td></tr><tr><td>$\mathcal{M}$</td><td>$\mathcal{PWD}$,
$\mathcal{LEAK}$,
$\mathcal{MITM}$</td><td>$\mathcal{LEAK}$,
$\mathcal{MITM}$</td><td>None</td></tr></table></td><td><table><tr><th></th><th>Type-0/1</th></tr><tr><td>$\mathcal{E}$</td><td>None</td></tr><tr><td>$\mathcal{M}$</td><td>$\mathcal{LEAK}$, $\mathcal{MITM}$*</td></tr></table>* Only if $\mathcal{M}$ uses a routing-MITM attack</td></tr>
<tr><td>Tunnel
(established with $\mathcal{M}$)</td><td colspan="2"></td><td><table><tr><th></th><th>Type-0/1</th></tr><tr><td>$\mathcal{E}$</td><td>None</td></tr><tr><td>$\mathcal{M}$</td><td>$\mathcal{LEAK}$</td></tr></table></td></tr>
</table>

Table 5.1: Summary and comparison of the assurances provided by conventional password protocols and pwdArmor.

Compromise of Long-term Secrets If pwd_{U} is compromised by $\mathcal{M}$, she can impersonate U until the password is changed. In $\mathcal{S}_{tunnel}$, if H's private key is stolen by $\mathcal{M}$, then she can perform $\mathcal{MITM}$ breaks, however, it does not improve her ability to directly learn pwd_{U} and obtain a $\mathcal{PWD}$ break. In pwdArmor, the ability to authenticate as the legitimate host within the context of the tunnel is only valuable in that it more effectively convinces U to initiate an authentication with $\mathcal{M}$.

If pwd_{U}^{ver} is compromised (e.g., H's verifier database is stolen), then two attacks are possible. First, $\mathcal{M}$'s possession of pwd_{U}^{ver}constitutes a $\mathcal{LEAK}$ break. Second, $\mathcal{M}$ can perform a DH-MITM attack and obtain an unencrypted R_{U}. This attack is possible since $\mathcal{M}$ now knows both private components of $\kappa_{\mathsf{U},\mathsf{H}}^{enc}$ and can decrypt $E_{\kappa_{\mathsf{U},\mathsf{H}}^{enc}}(\cdot)$. For Type-0 protocols this constitutes a $\mathcal{PWD}$ break. For Type-1 protocols a $\mathcal{MITM}$ break is possible since $\mathcal{M}$ can re-encrypt R_{U} (and the correct value for ID_{H}) using the appropriate key as well as construct a valid value for $\text{MAC}_{\kappa_{\mathrm{U,H}}^{mac}}(\cdot)$.

pwdArmor vs. Conventional Password Protocols Table 5.1 summarizes the security analyses of pwdArmor and conventional password protocols. When an encrypted tunnel is

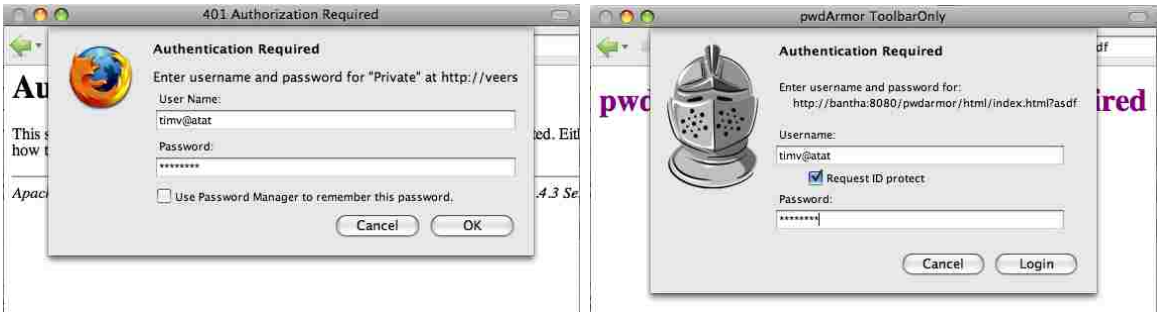


Figure 5.2: Modern web browsers can already prompt users for login information. The traditional HTTP Authentication dialog (left) is mimicked by our integration (right) of pwdArmor into Firefox.

used and correctly established (i.e., Assumption 1 holds) neither approach is vulnerable to the attacks of this model. The more interesting comparison comes when no tunnel is used, or the tunnel was created with the attacker. In both these cases Type-0/1 protocols perform equally poorly, while pwdArmor provides significant improvements.

5.5 Deployment Considerations

Client-side Support Due to the computational requirements of pwdArmor, client-side software support is essential. Existing systems (e.g., browsers, SSH clients, wireless supplicants) must be updated in order to support pwdArmor. Web browsers present an attractive avenue for incremental deployment, since, without intervention by browser vendors, client-side support for pwdArmor can be provided via browser extensions or zero-footprint clients (see Section 5.6). To prevent $\mathcal{M}$ from circumventing the login process, pwd_U must be delivered directly to the client-side software and not the server. This is not a problem for most client software (e.g., SSH clients, wireless supplicants), however, it is a significant problem for web browsers, which typically rely on a server-supplied HTML-based login page.

Entering login credentials through the browser's chrome is an attractive alternative to logins embedded in web pages because it provides a consistent authentication experience across all domains and avoids problems with malicious login pages. Modern web browsers are

already capable of prompting users for login credentials (see Figure 5.2) when they recognize a server's request for HTTP Basic or Digest authentication. This practice, however, is not widely adopted by web sites.

Wireless Authentication Extensible Authentication Protocol (EAP) [3] provides a valuable framework for implementing and deploying pwdArmor for wireless (and also wired) network authentication in both $\mathcal{S}_{clear}$ and $\mathcal{S}_{tunnel}$ scenarios. Although EAP provides mechanisms to directly bind authentication protocols to a secure tunnel (see Section 5.7), this binding does not provide any guarantees if the host is not properly authenticated (i.e., the user has been tricked by a phisher). Therefore, pwdArmor is still useful as it protects against users negating the benefits of the tunnel. Support for a new pwdArmor EAP method must be added to the user's wireless supplicant software and to the network authentication server. For home users, it is preferable for pwdArmor to be supported by the access points themselves as opposed to requiring them to host a specialized authentication server attached to the access points (as it is typically done in enterprise environments).

5.6 Implementation

We have developed `libpwdarmor`, a general purpose library written in Java, that provides the functionality needed to build client/server pwdArmor modules. This library creates and processes pwdArmor messages in binary and human-readable formats (see Section 5.6.1), however, the parent application is responsible for transporting these messages to their intended destination.

Currently, `libpwdarmor` supports HTTP Basic and OTP authentication and the following verifiers:

- MD5-based BSD password algorithm (essentially, but a bit more complicated than, the password hashed 1000 times; used by most Linux distributions to create the password verifiers stored in /etc/shadow)

- Apache variant[6] of the MD5-based BSD password algorithm (used by the Apache web server to create verifier databases for use with HTTP Basic authentication)
- HTTP Digest $H(A1)$ verifier (the MD5 hash of "username:realm:password")
- MD5-based OTP verifiers (the n^{th} truncated hash of the seed and the password; these non-static verifiers are updated after each successful authentication to the $(n-1)^{th}$ hash)
- SHA1-based OTP verifiers (same as above except the result of the SHA-1 hash function is converted to little endian before it is truncated).

HTTP Basic can be adapted to use any of these verifiers, however, it only makes sense to use OTP authentication with the OTP verifiers.

`libpwdarmor` adds a user options element (O_{U}) to inform H of U's supported DH groups and password protocols as well as to request privacy protection for ID_{U}. It also adds a host options element (O_{H}) to enable H to notify U of the selected DH group and password protocol. This element also includes a session identifier that allows interaction with H over a stateless transport mechanism.

As `libpwdarmor` supports a variety of underlying protocols, $\mathcal{M}$ could manipulate O_{H} so that U will use the weakest password protocol or verifier format she supports. This has the potential to increase the efficiency of an offline guessing attack (e.g., one hash with HTTP Digest verifier vs. 1000 hashes with MD5-based BSD password verifier). A relatively time-intensive PRF could be used to limit the effectiveness of this attack.

Client-side Support We used `libpwdarmor` to create a browser extension for Firefox. This extension recognizes that a web site supports pwdArmor and transports the pwdArmor messages via HTTP headers. The extension uses a similar modal dialog box as browsers currently use to prompt users for login information (see Figure 5.2).

[6]Which only differs from the original algorithm by changing the "magic" string from 1 to apr1.

	HTTP Basic (MD5-based BSD Verifiers)	OTP	HTTP Basic (HTTP Digest Verifiers)
ID_U	"timv@atat"		
α	alg=apr1, salt=CGyXh...	alg=otp-sha1, seed=pongo, cnt=100	alg=http-digest, realm=Hoth
C	n/a	otp-sha1 99 pongo	n/a
R_U	password	AND FULL FAN GAFF BURT HOLM	password
R_H	n/a		

Table 5.2: The pwdArmor message contents for specific conventional password protocols (nonces, DH key exchange values, and encrypted values are omitted).

We also used `libpwdarmor` to create a signed Java Applet. As zero-footprint clients, like Java Applets, involve browsers running server-supplied code they cannot provide the same assurances as a pure client-based approach, but are attractive due to their portability. This applet is intended to be loaded from a trusted source and then used to login to any pwdArmor-enabled web site. After the applet is loaded the user enters the web site she wants to authenticate to, or selects from a previously established list, and enters her login information. After the authentication is complete the applet opens a new browser tab with the cookies it received from the host and, thus, transfers the authenticated session from the applet to the browser.

Server-side Support Server-side support is realized as a Servlet Filter in Tomcat. This filter prevents access-restricted pages from being retrieved by unauthorized users and relies on an HTTP header to indicate that pwdArmor logins are available. It also uses HTTP headers to exchange the pwdArmor messages created by `libpwdarmor`. Once authentication is successful it uses session cookies to maintain an authenticated state with the client.

5.6.1 Integrating with Existing Protocols

Table 5.2 gives examples of the specific contents of the pwdArmor messages for the following password protocols:

HTTP Basic with the Apache Variant of MD5-based BSD Verifiers In this protocol α specifies the method of verifier creation and contains the salt that is required to create the same verifier stored by H. HTTP Basic has no host-supplied challenge and expects pwd_{U}, which is sent in R_{U}.

If $\mathcal{M}$ steals pwd_{U}^{ver} from H and then tricks U into attempting an authentication to her then she will be able to use $\kappa_{\mathsf{U},\mathsf{H}}$ and pwd_{U}^{ver} to decrypt R_{U}. Since HTTP Basic is a Type-0 protocol, the value for R_{U}, in this case pwd_{U}, allows $\mathcal{M}$ to repeatedly impersonate U until pwd_{U} is reset.

Outfitting HTTP Basic with pwdArmor provides it with the same protections it would have normally received if it were contained within an encrypted tunnel, and adds the benefit of protecting R_{U} if that tunnel is circumvented.

OTP The value of α is equivalent, in content, to the challenge from the previous successful authentication. The challenge contains the information to use pwd_{U} to derive current one-time password, which is sent to H in R_{U}.

If $\mathcal{M}$ obtains a copy of pwd_{U}^{ver} and tricks U into attempting an authentication to her, she will be able to decrypt R_{U}. As OTP is a Type-1 protocol, the value for R_{U}, in this case the current one-time password, allows $\mathcal{M}$ to impersonate U a single time. It is interesting to note that when OTP is used without pwdArmor it is vulnerable to a "small n" attack in which $\mathcal{M}$ reduces the hash index of the real host's challenge. If $\mathcal{M}$ manipulates the hash index contained in α, she will, at worst, obtain $\mathcal{LEAK}$ break since pwd_{U}^{ver} is needed to decrypt R_{U} and obtain the "small n" value generated by U. If $\mathcal{M}$ obtains a copy of pwd_{U}^{ver}, the small n attack is still not feasible as the value for pwd_{U}^{ver} that U will generate will not correspond to the stolen pwd_{U}^{ver} as their respective hash indices are not equal.

HTTP Basic with HTTP Digest Verifiers The algorithm and realm specified in α enables U to generate the $H(A1)$ HTTP Digest password verifier. Again, HTTP Basic has no host-supplied challenge and expects pwd_{U}.

pwdArmor provides HTTP Basic with the same protection from $\mathcal{PWD}$ breaks as HTTP Digest, without requiring password-equivalent verifiers. In this approach pwd_{U} is hashed and then compared to $H(A1)$, allowing what were password-equivalent verifiers to become password-dependent. If HTTP Digest is still employed elsewhere using these same verifiers, then they remain password-equivalent for those HTTP Digest authentications.

5.7 Related Work

As is evidenced by the success of password phishing, the assurances provided by encasing conventional password protocols within encrypted tunnels (e.g., TLS) can be easily circumvented by tricking the user. The key problems with encrypted tunnels are the difficultly for users to: 1) Determine if the host authentication ever occurred; and 2) Correctly verify the identity of the server. Several solutions have been proposed to improve the authentication of H by U. Visualization techniques [67] help improve verification of public keys. Additional human perceptible server authenticators (e.g., pictures, voice) [31] may also help users. Other systems, like BeamAuth [5], use bookmarks to ensure that logins only occur with legitimate sites. These solutions do not address the lack of creation of the tunnel. pwdArmor provides the same passive/active protections as encrypted tunnels, along with the additional benefit that if the host authentication is circumvented then pwd_{U} is not leaked.

Delayed password disclosure (DPD) [46] authenticates H without a secure tunnel by requiring U to verify the correctness of a host-supplied image after each character of pwd_{U} is entered. A *distinct* oblivious transfer for *each* password character ensures that actual password character is not disclosed. U then authenticates via a traditional PAKE protocol. Unlike pwdArmor, DPD is designed to thwart static phishing sites and does not protect against MITM attacks. Also, pwdArmor maintains the current practice of allowing users to quickly submit their usernames and passwords.

The need to couple an inner authentication protocol with its outer tunnel has been previously examined as the *compound authentication binding problem* [8, 69]. The solution

proposed (and adopted by EAP [3]) requires that the resulting session key be derived from: 1) The tunnel key; and 2) A key created by the inner authentication protocol or from pwd_{U}. Although this binding prevents MITM attacks, the tunnel must be modified and the password protocol is not protected if the tunnel is incorrectly established.

IKEv2 [47] does not directly address the tunneling of Type-0/1 protocols, which it calls "legacy authentication" mechanisms, but does supports the optional binding of itself to EAP, provided the underlying EAP method produces a key. As with EAP, IKEv2 does not provide any protections against phishers if the tunnel is incorrectly established.

Oppliger et al. [64] couples a TLS session to a specific authentication through a hardware/software token. Client-side certificates (from the token) are used in the TLS handshake to prevent MITM attacks, not for client authentication. pwdArmor accomplishes the same goals by using nonces to ensure unique sessions and by using ID_{H} to ensure a tight coupling with the encrypted tunnel.

Halevi and Krawczyk [36] specifies a generic, password-based, encrypted challenge-response protocol and an instantiation that provides mutual authentication and key exchange. Even though this protocol encrypts pwd_{U} with H's public key, if the phisher successfully performs the certificate trick then the benefits of this protocol are completely negated. Also, the requirement that H have a public key pair may be unreasonable for the low-value services that typically operate in $\mathcal{S}_{clear}$ scenarios.

PAKE protocols are the strongest answer, to date, for the troubles with password-based authentication through encrypted tunnels. There are a variety of protocols: SRP [87], SPEKE [45], EKE [14], PDM [48], SNAPI [57], PAK [56], AuthA [13], AMP [51]. If executed correctly, these protocols are not vulnerable to $\mathcal{PWD}$, $\mathcal{LEAK}$, or $\mathcal{MITM}$ breaks. As pwdArmor only provides all of these assurances if U does not fall victim to a certificate trick, PAKE protocols provide superior benefits to pwdArmor, with two exceptions. First, and most significantly, is their inability to reuse existing password verifier databases as detailed in Section 5.2.3. Second, is their lack of privacy protection for ID_{U}. With the

exception of SNAPI, this protection cannot be easily added to these protocols as their DH-based key exchanges are tightly coupled to ID_{U} (and as such cannot be used to privately transmit ID_{U}) and they do not require H to have a public key pair.

5.8 Conclusions and Future Work

pwdArmor is a more secure alternative to the current practice of encasing conventional password-based authentication protocols within a server-authenticated, encrypted tunnel. The key benefit of pwdArmor is evident when users think they are authenticating to one of their legitimate service providers, but are, in fact, being phished. In this scenario, all the benefits of using an encrypted tunnel can be negated by users, whereas in pwdArmor, users' passwords are never disclosed. This advantage comes from pwdArmor's treatment of server authentication as an added bonus, rather than the linchpin of its assurances.

Unlike other PAKE protocols, pwdArmor can reuse existing verifier databases while maintaining the password-dependent nature of those verifiers. Reuse of existing verifiers is valuable as it allows simplified deployment, avoids the overhead of creating new verifiers, and preserves compatibility with legacy systems. pwdArmor also, unlike other PAKE protocols, offers optional privacy protection to the user's identity during the authentication.

Operating system utilities (e.g., CardSpace [19]) for managing and using login credentials have the potential to unify user authentication across a variety of mediums (e.g., web site, wireless network, local application logins). As such, they represents an attractive avenue for deploying the client-side functionality required by pwdArmor.

Assuming the presence of HTTPS only for login pages, SessionLock [6] secures resulting web sessions from passive eavesdropping without TLS. A combination of pwdArmor and SessionLock, could eliminate the need for HTTPS for a large number of low-security web sites and therefore remove the additional costs associated with the performance overhead and caching behavior of TLS.

Acknowledgments. This research was supported by funding from the National Science Foundation under grant no. CCR-0325951, prime cooperative agreement no. IIS-0331707, and The Regents of the University of California.

Chapter 6

Derivative Applications

This research is designed to remove the need for service-specific passwords by providing convenient and secure approaches to decentralized authentication. This chapter presents three systems that take advantage of our new authentication protocols and demonstrate the utility and potential of our new technologies. While SAW is used as the authentication mechanism for these systems, Luau is also suitable if its deployment requirements are met. pwdArmor may be used in conjunction with Luau or as a replacement to an existing password protocol at an identity provider.

6.1 Extensible Pre-Authentication in Kerberos (EPAK)

P. L. Hellewell, T. W. van der Horst, and K. E. Seamons. Extensible Pre-Authentication in Kerberos. *23th Annual Computer Security Applications Conference (ACSAC)*, Miami, FL, December 2007.

6.1.1 Overview

Kerberos [50] provides distributed identity-based authentication. It is a time-tested and widely adopted approach, used by business, government, military, and educational institutions. It enables a user to authenticate a single time and then use the result of that authentication to log in to the application servers within the Kerberos realm for a predetermined

time period. Kerberos is a closed-system: every user must be known to the authentication server *a priori*.

As an organization's access control systems and applications are often built around the Kerberos infrastructure (e.g., Microsoft Active Directory), replacing Kerberos entirely is prohibitive. Extending Kerberos itself to support additional authentication schemes is therefore an attractive solution as it allows systems like Active Directory to remain intact. Nevertheless, modifying Kerberos represents significant challenges due to a lack of source code availability for some implementations and a lengthy standardization process.

Extensible Pre-Authentication in Kerberos (EPAK) extends Kerberos to facilitate the integration of future authentication mechanisms. EPAK introduces a loose coupling between Kerberos and its authentication scheme that enables new schemes to be added without further modification to Kerberos. To demonstrate the power and flexibility of the EPAK framework, we integrated two new authentication methods: trust negotiation and SAW. Both of these authentication approaches enable open systems: users do not have a direct pre-established relationship with the authentication server.

6.1.2 How EPAK works (High Level)

Normal Kerberos Operation The Kerberos protocol requires four parties: a Client, an Authentication Server (AS), a Ticket-Granting Server (TGS), and an Application Server. The AS is responsible for authenticating the Client. The TGS issues tickets that allow a Client to access an application server.

The Kerberos authentication process consists of three phases (see Figure 6.1). In Phase 1, the Client requests a ticket-granting ticket (TGT) from the AS. The response from the AS can only be decrypted by a Client with the valid user's password.

The dependence on a password to successfully complete Phase 1 tightly couples the password-based authentication to the process of obtaining a TGT. This tight coupling complicates the switch to non-password-based authentication mechanisms.

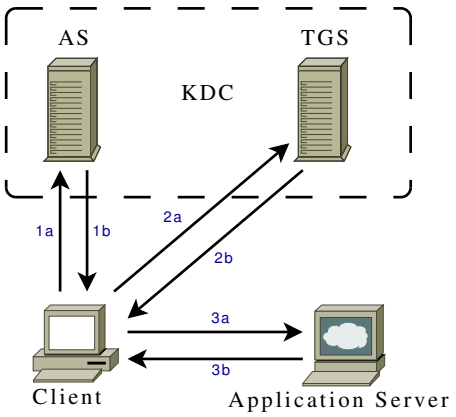


Figure 6.1: The Kerberos protocol has three phases.

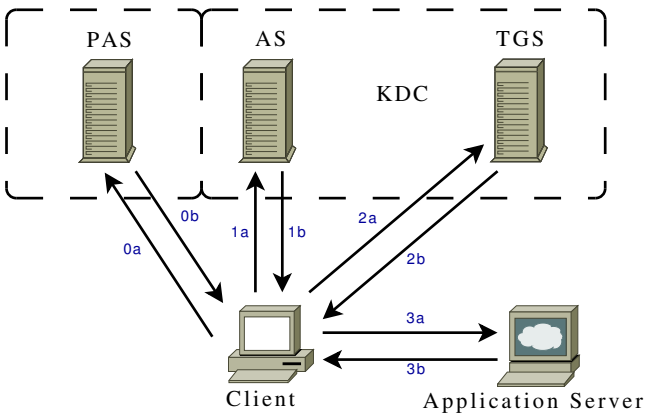


Figure 6.2: EPAK adds Phase 0 to the original Kerberos protocol.

In Phase 2, the Client uses the TGT to obtain a service-granting ticket (SGT). In Phase 3, the SGT is used to access the application server. Phase 1 only needs to be performed once per time period, while Phases 2 and 3 can be repeated (e.g., to access other application servers in the domain) as needed during that time period.

Kerberos with EPAK EPAK adds a single phase to Kerberos that is similar to the three existing phases (see Figure 6.2). This phase, between the Client and the Pre-Authentication Server (PAS), is used to authenticate the Client using the desired authentication mechanism. Once an authentication succeeds, the Client receives an authentication-granting ticket (AGT) from the PAS. This AGT is used to obtain a TGT from the AS. This straightforward modification removes the tight coupling (i.e., the password) between the Client and the AS and enables a variety of authentication approaches to be leveraged without further modification to Phase 1.

A significant benefit of EPAK is that authentication systems with slow performance (e.g., SAW with high-latency email delivery) can leverage Kerberos' SSO capability.

6.2 CPG: Closed Pseudonymous Groups

R. S. Abbott, T. W. van der Horst, and K. E. Seamons. CPG: Closed Pseudonymous Groups. *Workshop on Privacy in the Electronic Society (WPES)*, Alexandria, VA, October 2008.

6.2.1 Overview

Anonymity is useful when eliciting honest and forthright feedback, especially if the comments are negative or controversial. Closed Pseudonymous Groups (CPG) is a system for pseudonymous communication within a closed group. In CPG, each legitimate group member obtains a pseudonym (nym) that cannot be linked to the user's identity, but can be used to link together a user's actions.

Although linkability is typically frowned upon in many anonymous systems, the ability to connect an anonymous user's actions can be valuable in many scenarios as it enables administrators to accurately track a user's activity over time. If users abuse their anonymity, they can be blacklisted without compromising their true identity. CPG is designed to be easy to understand, to implement using existing techniques, and to use.

Although CPG can be implemented with a variety of authentication schemes, our prototype implementation uses SAW for the authenticate initial authentication of group members, which are uniquely identified via their email addresses.

The account provisioning inherent in SAW makes this initial authentication a simple, straightforward process, especially when there are no pre-established secrets (e.g., passwords) between group members and the anonymous system. Nyms in our prototype implementation are realized as separate (anonymous) email addresses obtained independently by group members. As the anonymous systems uses SAW to authenticate these nyms, it does not need to manage user-specific passwords. This also helps ensure anonymity if a user forgets their nym account password as an external email provider handles the reset process, not the

Stage 1 Membership Token Acquisition	**Stage 2** Wait	**Stage 3** Nym Registration	**Stage 4** Service Access
• Server authenticates client • Server signs blinded membership token (if one has not already been issued to this client) • Client unblinds token	...	• Server verifies unblinded token • Server authenticates nym • Server records token as used • Server authorizes nym for service access	• Server authenticates nym • Client uses service

Table 6.1: Summary of CPG stages

anonymous system. The prototype implementation extends the SAW toolbar for Firefox to automate the client-side computation required for CPG.

6.2.2 How CPG Works (High Level)

CPG requires three parties: membership server, service provider, and client. The membership server verifies a user's group membership and enables pseudonym registration with a service provider. CPG is resilient to collusion between the membership server and the service provider to discover a user's true identity.

CPG has four stages (see Table 6.1). In Stage 1, the membership server authenticates the client and signs a client-generated, blinded membership token. Stage 2 is a mandatory waiting period to thwart timing attacks. In Stage 3, the client register a pseudonym with the service provider using the unblinded membership token. Lastly, in Stage 4, the client is able to use the pseudonym to log in to the service provider. The first three stages are used to setup a nym with the service provider and the final stage is used during the lifetime of the nym to access the provider.

6.3 EASEmail

T. W. van der Horst, R. Segeberg, and K. E. Seamons. EASEmail: Easy Accessible Secure Email. *(Work In Progress)*, Computer Science Department, Brigham Young University, 2009.

6.3.1 Overview

The majority of email messages are sent without any cryptographic protection; thus, their contents are clearly visible to any eavesdropper. In this respect, emails are like postcards, which have two major drawbacks. First is their lack of message privacy. Second is their vulnerability to forgery or modification. We refer to this level of protection as *postcard-level* security.

Given these drawbacks, users, on occasion, desire to protect the content of their messages from passive observation. This is especially true for web-based email (e.g., Gmail, Hotmail, Yahoo) where all email is stored on the server. In the physical world, users protect a message by using an envelope, which protects its contents from passive observers. We refer to this level of protection as *envelope-level* security.

What do users have in the email world? In rare cases, senders and recipients share a previously established secret(s). More common is the small minority that uses public key-based approaches for encrypted email. A traditional PKI-based approach (e.g., PGP[68], S/MIME [75]) requires senders and recipients to generate key pairs, reliably distribute their public keys, and vigilantly protect (and backup) their private keys. Users are also required to obtain and manage the public keys of others. These are intimidating, burdensome, and error prone tasks [83] for many users. We refer to this level of protection as *armored car-level* security.

EASEmail is a new encrypted email approach that provides *at least* envelope-level security. By thwarting passive observation of message contents and raising the bar for

message modification/forgery it provides an attractive middle ground between traditional email delivery and the armored-car protection of traditional PKI-based approaches.

EASEmail requires no sender/recipient interaction on top of traditional email and eliminates the need for users to store/manage private keys or manage/distribute public keys, a major impediment to the proliferation of encrypted email. Using EASEmail, people send encrypted emails without directly establishing or exchanging keys with recipients. Although EASEmail can be implemented with a variety of authentication schemes, the prototype implementation used SAW to authenticate ownership of these email addresses.

6.3.2 How EASEmail Works (High Level)

Traditional email encryption methods require senders to obtain recipients public keys. EASEmail solves this issue by introducing a lightweight symmetric key distribution center (KDC). In EASEmail, once a sender authenticates and obtains an encryption key from the KDC, existing standards for email message encryption are used to secure the message and send it to the recipient. The recipient can likewise authenticate to the KDC and obtain the message decryption key.

The KDC uses SAW to authenticate users before releasing message encryption keys. The keys furnished by the KDC are derived from a single master key using an identity-based key derivation function. As such, the KDC does not have to store any user- or message-specific information, making it lightweight and highly scalable.

It is possible to involve more than one KDC in the creation of an EASEmail. Involving more than one key server distributes trust ordinarily placed in a single key server. Consequently, the risk to an encrypted email from a compromised KDC is greatly reduced. All involved key servers must be compromised or collude to decrypt a message. This is a valuable property, especially when the sender and/or recipient controls at least one of the key servers. Both senders and recipients are able to interact with multiple key servers in

parallel because these interactions are not dependent on one another. Threshold schemes [73] can also be used to require m of n KDCs.

Chapter 7

Summary and Conclusions

The goal of this research is to reduce the need for service-specific passwords, while improving the security and convenience of user authentication. This chapter summarizes our contributions, outlines two adoption strategies, and identifies future work.

7.1 Our Contributions

Identify email providers as the de facto identity providers on the Internet: This research addresses the old quandary of "Who do we trust, and for what?" [27] by leveraging the emergent trust in email providers that has made them the *de facto* identity providers on the Internet. Although a personal messaging medium is valuable for communication, it is the relationship between the personal messaging provider and user, rather than the medium itself, that is ultimately useful for identification and authentication. The ability to leverage this relationship independently of the messaging medium is valuable as the potential latency of these mediums is high.

Provide a unilaterally deployable solution: SAW enables web sites to unilaterally adopt email-based decentralized authentication. SAW requires no specialized third party, no client-side software, reuses existing identifiers and authenticators, and represents an acceptable risk to web sites that already employ email-based password resets.

Reduce the latency and improve the security of SAW: WARP and Luau demonstrate that the latency and client-side connectivity restraints of SAW can be overcome if the

identity provider supports the decentralized authentication protocol. They can also thwart the one-session active impersonation attack present in SAW.

Unify authentication in application and network layers: Without the restraint of client-side connectivity, WARP and its successor, Luau, can provide decentralized authentication in both web and wireless scenarios

Improve security by moving logins off web pages: All four of the systems created in this research enable user authentication to be moved off server-controlled web pages. This dramatically reduces the attack-surface for password phishing and enables improved detection of man-in-the-middle attacks by both clients and servers.

Extend the useful lifetime of passwords: Passwords have not outlived their usefulness; rather, the current manner in which they are used is stretching the bounds of their utility. The crux of the problem is that the limitations of human memory and patience curb the scalability of this traditional 1:1 relationship between users and service providers. This research presents decentralized authentication mechanisms designed to shift this relationship to a 1:n, in which a single relationship with an identity provider allows a user to authenticate herself to multiple relying parties.

Bootstrap more secure alternatives: We argue that these new approaches not only extend the useful lifetime of passwords, but also provide a robust platform for improving privacy, auditing, and delegation of authority, while maintaining the simplicity, the convenience and the portability of passwords. They also pave the way towards attribute authentication, the idea that a user authenticates based on "what" she is rather than "who" she is.

7.2 Adoption Strategies

The power to catalyze the adoption of decentralized authentication lies in the hands of service providers and personal messaging providers, but what comes first: relying parties or identity providers? As it is impossible to foresee which party will take the first step, we identify two likely adoption strategies for enabling secure and convenient decentralized authentication using passwords.

7.2.1 A Bottom-up Approach

In this strategy web sites take the initiative to bootstrap the adoption of decentralized authentication. They use SAW to become relying parties that leverage unmodified email providers as identity providers. SAW enables users and relying parties to experience the power and flexibility of decentralized authentication without having to obtain a new identifier, password, or additional client-side software.

As web sites can unilaterally deploy SAW, it offers them an opportunity to "dip their toes in the water" and test out decentralized authentication. If desired, a roll-out of SAW can be done in parallel to an existing password system. The benefits to web sites of adopting SAW include not having to maintain a database of user authentication information, which is an attractive target for attackers and malicious insiders.

Email providers can take several steps to foster this process. For example, as the convenience of SAW hinges on the latency of email delivery, giving higher priority to the delivery of authentication email messages helps minimize login delays. Also, enabling TLS for email delivery (at least for authentication messages) removes SAW's vulnerability to a one-session, active impersonation attack. Lastly, client-side automation software significantly improves the convenience of SAW; email providers could promote, develop, or distribute these tools.

The final milestone in this adoption strategy is the creation of a dedicated identity provider service using Luau: a tailorable framework that provides a password-based de-

centralized authentication solution that unifies authentication for the application and the network layers. Providing a dedicated identity provider service, like Luau, alleviates the additional pressure on personal messaging mediums created by SAW and removes the achilles heel of message delivery latency.

pwdArmor can ease the transition of entities into identity providers by facilitating the secure reuse of existing password verifiers. The client-side software for SAW that provides a uniform and consistent login experience across disparate domains (and provides improved protections against phishing) also facilitates an easier transition to more advanced approaches like Luau by abstracting away the underlying decentralized authentication mechanism.

Though an important milestone, Luau is itself a jumping-off point for attribute-based authentication (see Section 7.3).

7.2.2 Top-down: "The open hand"

This strategy does not involve SAW and begins with the final milestone of the bottom-up strategy. In this approach, personal messaging providers adopt Luau and then invite and encourage web sites to become relying parties and accept this form of authentication. Again, the incentive for personal messaging providers is to add value to their accounts. The incentives for web sites is the ability to offload user authentication and the potential ability for the large number of users of an identity provider to able to instantly make use of the web sites' services. Again, pwdArmor eases the creation of identity providers by facilitating the secure reuse of existing password verifiers.

Though similar to the deployment strategy employed by Microsoft Passport [65], which ultimately failed, this approach has the advantage of a looser coupling between relying parties and identity providers.

7.3 Future Work

The systems presented in this dissertation do not remove password-based authentication from the equation, they simply enable users to conveniently, and securely, leverage their existing login credentials to their personal messaging providers to authenticate to relying parties in open systems.

Although these novel authentication technologies are dependent on passwords, they also provide a springboard to more secure authentications mechanisms. After the paradigm shifts from service-specific passwords to a reliance on identity providers, it is much easier to exploit the benefits of other authentication technologies (e.g., smart cards, biometrics, etc.) as changes only have to be made at the identity provider, not every site that the user frequents.

While personal messaging providers have significant potential as identity providers, there are also many organizations (e.g., universities, employers, financial institutions, governments) that are trusted to vouch for not only a user's identity but also a selected set of attributes (e.g., enrollment status, memberships, credit status, age, citizenship). The ability of these entities to authenticate their users and the attribute information they maintain are valuable resources that should be migrated from the background to the forefront. Enabling external parties to leverage these relatively untapped information represents a reuse of valuable existing resources.

Past efforts at decentralized authentication have required significant business/legal relationships between identity and relying parties. Once attributes enter the equation these issues will have to be addressed. Once they are, then this research can be used to provide a springboard to valuable systems, like trust negotiation, which exploits the ability for identity providers to assert other attributes about their users, besides their identity.

References

[1] R. S. Abbott, T. W. van der Horst, and K. E. Seamons. CPG: Closed Pseudonymous Groups. In *Workshop on Privacy in the Electronic Society (WPES)*, October 2008.

[2] M. Abdalla, P. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three party setting. In *IEE Proceedings*, March 2006.

[3] B. Aboba, L. Blunk, J. Vollbrecht, and J. Carlson. RFC 3748: Extensible Authentication Protocol.

[4] B. Aboba and D. Simon. RFC 2716: PPP EAP TLS Authentication Protocol, 1999.

[5] B. Adida. Beamauth: Two-factor web authentication with a bookmark. In *ACM Conference on Computer and Communications Security*, October 2007.

[6] B. Adida. Sessionlock: securing web sessions against eavesdropping. In *International Conference on World Wide Web*, April 2008.

[7] APWG. Phishing Trends Reports, January 2008. Available from http://anti-phishing.org.

[8] N. Asokan, V. Niemi, and K. Nyberg. Man-in-the-Middle in Tunnelled Authentication Protocols. In *Security Protocols Workshop*, 2003.

[9] D. Balfanz, G. Durfee, R. Grinter, and D. Smetters. Network-in-a-Box. In *USENIX Security Symposium*, 2004.

[10] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *Advances in Cryptology – Eurocrypt*, 2000.

[11] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *Advances in Cryptology*, 1994.

[12] M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *ACM Symposium on Theory of Computing*, 1995.

[13] M. Bellare and P. Rogaway. The AuthA Protocol for Password-Based Authenticated Key Exchange. Submission to IEEE P1363, February 2000.

[14] S. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In *IEEE Symposium on Security and Privacy*, 1992.

[15] R. Bradshaw, J. Holt, and K. E. Seamons. Concealing complex policies with hidden credentials. In *ACM Conference on Computer and Communications Security*, October 2004.

[16] S. Brands. The Identity Corner: The problem(s) with OpenID. http://idcorner.org/2007/08/22/the-problems-with-openid/, August 2007.

[17] Cain and Able. Available from http://www.oxid.it/cain.html.

[18] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Theory and Application of Cryptographic Techniques*, 2001.

[19] CardSpace. http://msdn.microsoft.com/CardSpace.

[20] C.C.I.T.T. Recommendation X.509, The Directory - Authentication Framework, December 1988.

[21] C. Chang and J. Lee. An Efficient and Secure Multi-Server Password Authentication Scheme using Smart Cards. In *Conference on Cyberworlds*, 2004.

[22] K.-K. R. Choo, C. Boyd, Y. Hitchcock, and G. Maitland. On Session Identifiers in Provably Secure Protocols: The Bellare-Rogaway Three-Party Key Distribution Protocol Revisited. In *Security in Communication Networks*, 2004.

[23] Comment Authorization. http://wp-plugins.net/plugin/commentauth/.

[24] T. Dierks and E. Rescorla. RFC 4346: The Transport Layer Security (TLS) Protocol Version 1.1, April 2006.

[25] W. Diffie and M. E. Hellman. New directions in cryptography. In *IEEE Trans. on Information Theory*, Nov 1976.

[26] DigitalMe. http://code.bandit-project.org/trac/wiki/ DigitalMe.

[27] C. Ellison and B. Schneier. Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure. *Computer Security Journal*, 16(1):1–7, 2000.

[28] Firefox Password Manager. http://wiki.mozilla.org/Firefox:Password_Manager.

[29] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. RFC 2617: HTTP Authentication: Basic and Digest Access Authentication, June 1999.

[30] P. Funk and S. Blake-Wilson. Internet draft: Eap tunneled tls authentication protocol version 0. draft-funk-eap-ttls-v0-02, November 2007.

[31] S. Gajek, M. Manulis, A. Sadeghi, and J. Schwenk. Provably secure browser-based user-aware mutual authentication over TLS. In *ACM Symposium on Information, computer and communications security*, 2008.

[32] US General Accounting Office, Advances and Remaining Challenges to Adoption of Public Key Infrastructure Technology, GAO-01-277, 2001; http://www.gao.gov/new.items/d01277.pdf.

[33] S. L. Garfinkel. Email-based Identification and Authentication: An Alternative to PKI? *IEEE Security & Privacy*, pages 20–26, 2003.

[34] N. Goffee, S. Kim, S. Smith, P. Taylor, M. Zhao, and J. Marchesini. Greenpass. In *PKI Research and Development Workshop*, pages 26–41, 2004.

[35] P. Gutmann. PKI: It's not dead, just resting. *IEEE Computer*, 35(8):41–49, August 2002.

[36] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. *ACM Transactions on Information and System Security*, 1999.

[37] N. Haller. The S/KEY one-time password system. In *Symposium on Network and Distributed System Security*, 1994.

[38] N. Haller, C. Metz, P. Nesser, and M. Straw. RFC 2289: A One-Time Password System, February 1998.

[39] A. Harding, T. W. van der Horst, and K. E. Seamons. Wireless Authentication using Remote Passwords. In *1st ACM Conference on Wireless Network Security (WiSec)*, March 2008.

[40] P. L. Hellewell, T. W. van der Horst, and K. E. Seamons. Extensible Pre-Authentication in Kerberos. In *23th Annual Computer Security Applications Conference (ACSAC)*, December 2007.

[41] Higgins: Open Source Identity Framework. http://www.eclipse.org/higgins/.

[42] L. Hu, X. Niu, and Y. Yang. An Efficient Multi-server Password Authenticated Key Agreement Scheme Using Smart Cards. In *Conference on Cyberworlds*, 2004.

[43] R. Hwang and S. Shiau. Password authenticated key agreement protocol for multi-servers architecture. In *Conference on Wireless Networks, Communications and Mobile Computing*, June 2005.

[44] B. Ives, K. R. Walsh, and H. Schneider. The Domino Effect of Password Reuse. *Communications of the ACM*, 47(4):75–78, 2004.

[45] D. P. Jablon. Strong password-only authenticated key exchange. *SIGCOMM Comput. Commun. Rev.*, 1996.

[46] M. Jakobsson and S. Myers. Delayed Password Disclosure. *SIGACT News*, 38(3):56–75, 2007.

[47] C. Kaufman. RFC 4306: Internet Key Exchange (IKEv2) Protocol, December 2005.

[48] C. Kaufman and R. Perlman. PDM: A New Strong Password-based Protocol. In *USENIX Security Symposium*, 2001.

[49] T. Kivinen and M. Kojo. RFC 3526: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE), May 2003.

[50] J. T. Kohl, B. C. Neuman, and T. Y. Ts'o. The Evolution of the Kerberos Authentication Service. In *Spring EurOpen Conference*, 1991.

[51] T. Kwon. Authentication via Memorable Password. Submission to IEEE P1363, May 2000.

[52] M. Lepinski and S. Kent. RFC 5114: Additional Diffie-Hellman Groups for Use with IETF Standards, Jan 2008.

[53] L. Li, I. Lin, and M. Hwang. A remote password authentication scheme for multi-server architecture using neural networks. *IEEE Transactions on Neural Networks*, 2001.

[54] Liberty Alliance Project. http://projectliberty.org/.

[55] Light-Weight Identity (LID). http://lid.netmesh.org/.

[56] P. MacKenzie. The PAK suite. Submission to IEEE P1363, May 2002.

[57] P. MacKenzie and R. Swaminathen. Secure Network Authentication with Password Identification. Submission to IEEE P1363, August 1999.

[58] J. Malinen. wpa_supplicant. Available from http://hostap.epitest.fi/wpa_supplicant/.

[59] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 1978.

[60] NIST. The Keyed-Hash Message Authentication Code (HMAC). FIPS Pub 198a.

[61] OASIS. Extensible Resource Identifier (XRI) TC. http://www.oasis-open.org/committees/tc_home.php?wg_ abbrev=xri, April 2008.

[62] OpenID. http://openid.net/.

[63] OpenID Foundation. http://openid.net/foundation/.

[64] R. Oppliger, R. Hauser, and D. Basin. SSL/TLS session-aware user authentication – Or how to effectively thwart the man-in-the-middle. *Computer Communications*, 29(12):2238–2246, August 2006.

[65] Microsoft .NET Passport Review Guide. http://www.microsoft.com/net/services/passport/review_guide.mspx.

[66] Password Safe. http://schneier.com/passsafe.html.

[67] A. Perrig and D. Song. Hash Visualization: A New Technique to Improve Real World Security. In *Intl. Workshop on Cryptographic Techniques and E-commerce*, 1999.

[68] OpenPGP. http://tools.ietf.org/wg/openpgp/.

[69] J. Puthenkulam, V. Lortz, A. Palekar, and D. Simon. Internet Draft: The Compound Authentication Binding Problem. draft-puthenkulam-eap-binding-04.txt, October 2003.

[70] P. Resnick. RFC 2822: Internet Message Format, 2001.

[71] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 1978.

[72] P. Saint-Andre. RFC 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence, 2004.

[73] B. Schneier. *Applied Cryptography, 2nd Edition*. John Wiley & Sons, Inc., New York, NY, pages 70-71, 262-263, 1996.

[74] Shibboleth. http://shibboleth.internet2.edu/.

[75] S/MIME. http://tools.ietf.org/wg/smime/.

[76] M. Steiner, G. Tsudik, and M. Waidner. Refinement and extension of encrypted key exchange. *SIGOPS Oper. Syst. Rev.*, pages 22–30, 1995.

[77] SXIP 2.0 Protocol Specification. http://sxip.net/.

[78] D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin. Internet Draft: Using SRP for TLS Authentication, draft-ietf-tls-srp-14, 2007.

[79] The FreeRADIUS Server Project. FreeRADIUS. Available from http://www.freeradius.org/.

[80] The OpenSSL Project. OpenSSL. Available from http://www.openssl.org/.

[81] T. van der Horst and K. Seamons. Simple Authentication for the Web. In *Security and Privacy in Communication Networks*, September 2007.

[82] T. W. van der Horst and K. E. Seamons. pwdArmor: Protecting Conventional Password-based Authentications. In *24th Annual Computer Security Applications Conference (ACSAC)*, December 2008.

[83] A. Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *8th USENIX Security Symposium*, 1999.

[84] W. H. Winsborough, K. E. Seamons, and V. E. Jones. Automated Trust Negotiation. *DARPA Information Survivability Conference and Exposition*, January 2000.

[85] Wordpress. http://wordpress.org/.

[86] Wi-fi protected access. http://wifialliance.com.

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

[88] T. Wu. SRP-6. Technical white paper, Stanford University, October 2002.

[89] G. Zorn. RFC 2759: Microsoft PPP CHAP Extensions, Version 2, 2000.