

User-centric PKI

Radia Perlman
Sun Microsystems
16 Network Circle
Menlo Park, CA 94025
1-425-651-4094

radia.perlman@sun.com

Charlie Kaufman
Microsoft
1 Microsoft Way
Redmond, WA 98052
1-425-707-3335

charliek@microsoft.com

ABSTRACT

The goal of supporting Single Sign-On to the Web has proven elusive. A number of solutions have been proposed – and some have even been deployed – but the capability remains unavailable to most users and the solutions deployed raise concerns for both convenience and security. In this paper, we enumerate desirable attributes in a scheme for authenticating from an Internet browser to a web site and the authorization that follows. We categorize the currently deployed or advocated approaches, describing their benefits and issues, and we suggest incremental improvements to such schemes. We then outline a design for public-key based authentication particularly suited to what we believe to be the common case: users, acting on their own behalf (as opposed to as an employee of an organization), performing actions on the web such as making a purchase or maintaining an account at a service provider. We contrast the usability/privacy/security properties of our design with other identity management/authentication schemes deployed or being proposed today. Our design is truly user-centric, in the sense that the user acts as his own CA, and as a decision point for authorizing release of user information to web sites, rather than having an Identity Provider be the center of trust.

Categories and Subject Descriptors

K.6.5 [Management of Computing and Information Systems]: Security and Protection – *Authentication, Unauthorized access (e.g., hacking, phreaking)*.

General Terms

Design, Security, Human Factors.

Keywords

Web services, PKI, authentication, single sign-on

1. INTRODUCTION

This paper starts with a description of various problems that it would be desirable to address when using the Internet, particularly in the context of a user doing business with a variety of services on the web. Then we describe a variety of existing approaches, and review their strengths and weaknesses compared to the set of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IDtrust '08, March 4–6, 2008, Gaithersburg, MD

Copyright 2008 ACM 978-1-60558-066-1...\$5.00.

problems as well as to each other. We do not focus on particular syntax (such as SAML or X.509), or the details of particular deployed systems, but rather the properties of families of schemes that may or may not use the same standardized syntax. For example, we use the term “certificate” to refer to any digitally signed statement, which would include X.509 identity certificates[12], “attribute certificates” [7], or SAML assertions [3].

After describing the properties of currently deployed or advocated solutions, we describe an approach which is truly “user-centric”, in the sense that not only does the user decide which attributes to divulge to which sites, but mutual authentication to web sites with which the user has an existing relationship is directly performed between the user (the user’s machine) and a server, rather than, for instance, having the user authenticate to a third party (an “Identity Provider”), and have that identity provider vouch for the user. Also, the user’s private attributes, although perhaps stored (encrypted) at some service on the web to facilitate user roaming, are not available to anyone except the user, the user’s local machine, and on servers to which the user chooses to reveal them.

2. DESIGN GOALS

Today, users typically have independently configured and maintained user names and passwords at lots of different web sites, and must enter the appropriate ones when visiting particular sites. This is both inconvenient and insecure. Ideally, users would like to be able to authenticate to a machine once when they log in (possibly using the evolving “best practice” technologies like smart cards and fingerprint readers) and then have the machine authenticate to all visited web sites securely, efficiently, and transparently. We first list some desirable attributes (as a means for comparing various approaches). There are inevitable trade-offs; no solution meets all of the goals.

2.1 Single Sign-on

As with the slogan “the network is the computer”, the user experience should be of a single authentication step, for instance, activating a smart card and inserting it into a machine, or typing a username and password. After that, the user should be able to access any service on the net with which the user has an ongoing relationship, and be logged into the user’s account without the user consciously doing another authentication.

2.2 Protection from Phishing

Safety: If a user is tricked into authenticating to an evil site (e.g., phishing), the site should gain no credentials with which to impersonate the user, either at the site the user intended to visit, or at other sites to which the user has access.

2.3 Minimal Dependence on Third Parties

Efficiency: A user should be able to communicate directly with a web site without having to first communicate with other machines.

Robustness: A user should be able to communicate with a web site even if other sites are not reachable at the time.

Off-line Trusted Entities: It is desirable to have the system operate in a way that the most trusted components are not connected to the network. The reason for this is that a system that requires a trusted server to be on-line in order for authentication between other systems to work will be less secure because:

- An on-line server is an attractive target for network-based attacks. It is likely to have security flaws, and thus likely to be compromised. Even if not compromised, it is subject to DDOS attacks.
- Such a server must be replicated for availability and performance, and each replica must be physically secured.

2.4 Flexible use of accounts

Multiple accounts with the same service provider: A user may have multiple accounts with the same service provider. For instance,

- one account for purchasing work-related items, and another for personal items
- entering chat rooms with different pseudonyms
- multiple email accounts with a single email provider

It is desirable for authentication to be automatic, while still allowing the user to choose which account to access.

Multiple users with the same account: A family might share an account at a site such as a DVD-by-mail service. A user would like to be able to access all his private accounts through a single sign-on, but also be automatically logged into accounts on sites in which he is one of several authorized users.

Easy and Secure Account Creation: It should be easy to create a new account and have it automatically use, for future authentication, the credentials with which the user is currently logged in. If the user has several potential credentials (e.g., password, set of client certificates), the user should be able to select which should be used for this new account.

2.5 Convenient Roaming

Convenient Roaming: A user would like to be able to access accounts from a variety of different platforms (home desktop machine, work machine, laptop, cellphone, hotel lobby shared desktop machine).

Least Privilege: Machines (even a user's own machine) are not necessarily secure. Logging in with "full credentials" to perform a single operation such as printing a boarding pass, or checking to see what conference room the next meeting is in, might be taking an unwarranted risk.

Credential Agility: Although a smart card might be the most secure choice for user authentication, the desktop machine in the hotel lobby from which the user would like to print his boarding pass might not have a smart card reader. Or the user might have left his smart card at home, and still want to do work. It is

desirable for the user to be able to access the web through whatever authentication methods are practical for the situation.

Authorization based on authentication method: If there is the ability to log into different machines using different credentials as described above, some logins will be more secure than others. Therefore, it might be desirable for authorization decisions to depend on where the user is authenticating from, as well as how he authenticated. For instance, the user might decide that making trades in his brokerage account only be possible when he has authenticated with his smart card, or be working on his home desktop machine, whereas checking his available balance might be authorized from any location and using any authentication method.

Or a server holding a company's confidential files might allow access to those files only to users that have authenticated from an onsite machine using a smart card.

2.6 Additional user attributes

Convenient access to user information: There is some information that the user always knows, for instance, home address. It might be convenient for this information to be automatically filled in when required, but it is less necessary than other information that the user might need and not easily remember, such as passport number and credit card numbers.

Authenticated Attributes: There are some attributes that a user cannot simply claim by filling information into a field, such as whether they are over 18 (e.g., for purchasing alcohol over the web), or whether they are not a citizen of one of the countries currently disapproved of by some government (e.g., for export control decisions), or whether they are employed by a company that has corporate membership in some organization (e.g., a service that allows all employees of member corporations to have access to on-line search).

2.7 Credential loss or theft

One-step revocation: The user might want to change the credential with which he performs his single sign-on. This might be because, as is good security practice, he changes his password periodically, or because his smart card has been stolen. When this happens, accounts at all servers must (within a reasonable amount of time) know not to accept the old credential.

Protection against loss of credentials: In many schemes (including the one we advocate) a lot of important information is stored, encrypted with the user's private key or password. If the user forgets this password or loses the smart card with the private key, it should be possible to salvage all the information.

One-step rollover to a new credential: Once the old credential is revoked, it should be convenient to start using a new credential, without the requiring the user, for instance, to individually reinstalling his account at each service provider.

2.8 Minimal platform changes

Deployability of an authentication scheme depends critically on how many different components must be updated before it can be used. Virtually all of the schemes being proposed or deployed fall into one of two categories: schemes that can be deployed on client devices with supporting infrastructure without modifying the existing web sites; and schemes that can be deployed on web

servers with supporting infrastructure without modifying the deployed base of clients. Schemes that require changes to both clients and servers face a chicken and egg problem, since there is substantial cost and no benefit to deploying the first half of the solution.

Note that some client changes are easier than others. All of the major browsers are highly extensible (though incompatibly), allowing add-ons to be installed that do additional processing. There are a wide range of such add-ons aimed at specialized processing of web site logons. Users should be leery of installing add-ons from unfamiliar web sites, though most are not. Two problems with such add-ons are that they may not be available to users who are logging on to kiosk or otherwise unfamiliar machines, and they may not be available for the browsers on specialized devices like cell phones.

2.9 Access Privacy

A user should be able to access sites without any server being able to know which other services a user is using. Where possible, it is desirable for a user to have a degree of anonymity where the server knows that the user is authorized to access something without knowing the identity of the user. If the user is accessing two different sites, the user may wish that the two sites not be able to determine that the requests to them originate with the same carbon based life form.

3. EXISTING SCHEMES

In this section we review existing schemes. By “existing”, we mean either deployed or advocated. We are intentionally grouping families of schemes that have similar properties for the purposes of our discussion, and hope their developers won’t be offended by our glossing over design aspects they might consider to be crucial. This is also a rapidly growing field, and we are not attempting to make our list complete.

3.1 Classic Username/Password

While within an enterprise or a university environment, more sophisticated authentication schemes based on centrally registered “accounts” are used for file, database, and sometimes http access, the dominant authentication means on the World Wide Web remains username and password. On a good day, those credentials are protected from eavesdropping by SSL. They are used for everything from accessing banks and brokerage accounts to email and social networking sites, and by and large users have to choose separate usernames and passwords for each site they use. Payment schemes are predominantly based on users typing in credit card numbers and expiration dates to merchant sites. The inconvenience of these mechanisms is well known and the security weaknesses are well publicized. Users cannot remember high quality secrets (even in small quantities, much less in large ones), smart cards are not widely deployed (especially outside of an enterprise), and there is no coordination among Internet services that would allow any centrally managed form of authentication to be accepted everywhere.

Passwords present many security issues: some obvious and some more subtle. Users often choose passwords that are easy to guess, users write them down, the passwords are exposed to keyboard loggers and shoulder surfers, and users tend to use the same

password on multiple systems, allowing servers to impersonate the user to other servers.

Furthermore, prompts for usernames and passwords take place on ordinary web pages that can be easily forged, tricking users into giving away that information to impostor sites. While the protocols for authenticating web servers to browsers are cryptographically strong, the visual cues by which browsers identify web sites to users have been shown to be ineffective[27]. Users should not be expected to understand the arcana of URLs, and know, e.g., that <http://84.212.151.26/www.creditunion.com> is not actually the web site “creditunion.com”.

Although it might be desirable for a user to use a different password at every site, there is no way for a user to remember all the different usernames and passwords at all the sites at which the user has an account. Neither can the user pick a strong password and use it everywhere because sites have incompatible restrictions on what kinds of characters must and must not be contained in passwords.

The user can’t even use the same username at all places for reasons such as:

- The username the user has been using is already in use at some new site the user would like to register with.
- The site might have its own syntax rules about usernames that the user’s customary username does not fit (such as minimum or maximum length, or legal characters in the name).
- The user might not want anyone to be able to correlate his accounts at different sites.
- Some sites use the user’s email address as their username. Email addresses tend to change, since they are often provider-based or employer-based.
- The user might have several accounts at a site, and wish to select which one to use for the session.

To prevent passwords used at multiple sites from being exposed if one of the sites is compromised, sites should store a hash of the user’s password for verification rather than the actual password. Unfortunately, many sites store the cleartext password. This is obviously done at sites which email the actual password to the user when the user has forgotten the password.

When forgotten password recovery is to be based on the (dubious) security of email, there are two possibilities:

- A site keeps the user’s actual password, and emails the password to the email address the user has registered with, when the user claims to have forgotten her password.
- A site keeps a hash of the user’s password, and resets the password to some single-use password. When the user logs in (or clicks on the link in the email), the user is asked to set the password to something else.

The first system has the advantage that someone cannot maliciously claim to have forgotten someone else’s password, and cause that person to go through the inconvenience of resetting

their password. This would likely be only a minor inconvenience unless the user does not have access to email at the time.

The second system has the advantage that someone who steals the password database at one merchant cannot easily impersonate users at other systems. With the first system (cleartext passwords stored), even if a user were to customize passwords at different sites, a thief who notices the user's password is "amazonPwD89351" might guess that the user's password at ebay might be "ebayPwD89351".

Evaluating this "classic" username/password authentication against our design goals, it fails at Single Sign-on (the user must separately authenticate to each site) but provides some protection against phishing: it is easy for the attacker to trick the user into revealing the password to a target site, but at least the password is (hopefully) only useful in impersonating the user at the target site. It would be much worse if phishers could obtain a password that would allow access to all of the user's resources. It actually fares fairly well against the other criteria. Roaming is easy – passwords can be typed into any machine. Credential loss recovery is different for every site and there is no coordinated recovery if the user loses many at once (e.g., they were all stored on a sticky note or in a file on a stolen laptop). It has no dependence on third parties, allows maximum flexibility to users with multiple accounts, and by definition requires no platform changes.

Another merchant practice that causes inconvenience for users, as well as making them more vulnerable to security and privacy problems, is for merchants to maintain customer information beyond anything that adds value to the user. An occasional purchase of merchandise from a web site does not require a long term relationship with the user -- it is largely for the merchant's benefit that the merchant maintains personal information about the user after the transaction has completed.

Worse yet, in the case of the occasional purchase from an on-line merchant, it is often more onerous to be a returning customer than a new customer. If it has been several years since the user last interacted with the merchant, it is very likely the user has forgotten his username and password on the site. A new user need only type in information that he knows (such as his home address) and credit card (which he copies from a credit card he selects).

A common experience for a user that has purchased something from the site several years ago is that once she types her email address (needed to obtain a receipt), she is informed by the site that she is indeed a returning user, and must now log in, furnishing her username and password. Given that she has almost certainly forgotten both, she now has to go through a "username recovery procedure" that may or may not succeed (because she doesn't remember which telephone number she gave them when she bought from them 4 years ago), or what she made up for answers to the few amazingly inappropriate security questions the web site makes her choose from. Personally, I have no pet, no favorite sports team, I don't remember my 2nd grade teacher's name, and my father does not have a middle name.

How many things are wrong with this currently widely deployed strategy?

- *It should be no less convenient for someone who has previously purchased something at the merchant to*

make a purchase, than someone who is making a first time purchase.

- *If an account is to be maintained, and the user must find and authenticate with a username/password at the site, security questions for retrieving this forgotten information must be chosen by the user.*

Furthermore, in the case of an occasional purchase at a site, not only is the experience typically more onerous for a returning user than a first time user, but the ongoing relationship (the merchant keeping information about that user) is of extreme *negative* value to the customer. Not only is it less convenient for the user to make a future purchase (because of having to first go through the procedure for recovering the old username and password, which even if succeeds is likely to take several minutes), but the user's personal information is stored in a database that is likely to be broken into at some point. Most information the web site needs, such as the user's address, is reliably maintained in the user's head, thank you, and not a burden to type in. Other information, such as the credit card that the user used last time the user purchased something, is not only dangerous to store, but is likely to be out of date when a customer only purchases something from that merchant every few years,. Also, users tend to have many credit cards, and have reasons (perhaps based on the current balances on each of their cards, or current promotional rebates) for selecting a particular card for a particular purchase. Storing credit card information is thus not very useful to a customer and very dangerous.

There should be a law:

After a merchant has been paid, a customer must be allowed to request that any subset of the information about that user stored at the merchant (including all of it) must be deleted by the merchant. And of course, the merchant should be required to comply with this request.

But there are some web sites that do need to maintain an ongoing relationship with a user -- DVD subscription services in which a user maintains a movie queue, brokerage services where a user trades online, airline accounts where a user purchases tickets and accesses their frequent flier points, tax packages where users prepare their taxes and where their records are maintained year after year, to name a few.

3.2 Enhanced Username/Pwd Schemes

Browsers usually have a feature where they will remember, for various sites, what the user's username and password are, and fill them in for the user. The problem with this is that then it becomes even more likely the user will not remember his username and password when he winds up needing to access the site from a different machine, or discovers all that state irretrievably gone when his computer dies and he needs to replace it. *The less frequently a user needs to type his username and password to a site, the more likely it will be that the user will forget the username and/or password when the user actually needs to remember and type it.*

The Local Password store approach requires no changes to servers. Browsers typically have this feature "built-in" for storing this information when *HTTP authentication* [8] is used, but there

are a growing collection of add-ons that support it for the more common case where passwords are entered on a web page.

Examples include PwdHash[19], TrustBar[10], PasswordVault[20], Pvault[5], Skipper[23], and Passpet[29]. The idea is that plug-in software installed on client machines notices password prompts from servers and automatically satisfies them (usually invisibly to the user except for a small delay). PwdHash is a little different from the others. It does not keep a database of passwords, and does not change the user experience at all (users have to type in as many usernames and passwords as they ever did). PwdHash improves security in the common case where a user uses the same password at lots of sites. It hashes the password typed by the user with the site name to get a unique password for each site. This means that breaking into one site will not enable impersonation of the user at other sites and breaks phishing for passwords.

These systems are relatively straightforward conceptually, but in practice involve a lot of fragile trickery dealing with the plug-in architectures of the various browsers.

There is no standard password prompt that web pages display, though they do have a lot in common. The password field usually is configured to echo some special character rather than what the user types in the field, and it is often called "password". Figuring out exactly how to recognize a password prompt page and respond correctly for the great variety of existing web sites is a continuing challenge for people who maintain this sort of software.

This approach often has as a design goal that the user never sees the actual password, and it is long and randomly chosen to make it difficult to guess. That means that the software must also recognize the account creation page and know what various site's rules are for acceptable passwords so it can fill in appropriate ones. For sites that require periodic password changes, the software can automatically change the password while the user is doing something else.

A challenge with this approach is keeping the various password databases synchronized if the user accesses the web from multiple machines. Another is that the software (and database) must be installed on any machine from which the user wants to access the web, which may not be allowed in the case of kiosk or borrowed machines.

Most of the add-on schemes either have some sort of roaming built in or have some sort of export to a file capability to allow such roaming to take place manually. They also generally allow the roaming of other personal data like addresses, phone numbers, and favorite web sites.

Measuring these add-on schemes against our design goals, they do a good job at providing single sign-on. They typically depend on third parties only to support roaming. Their support of roaming, carrying additional user information, access privacy, and failure recovery is potentially very good, though the details vary by scheme. Dealing with shared accounts and users with multiple accounts is a user interface challenge, since it conflicts with seamless single sign-on, but these schemes have the potential to do as well as is possible. The most interesting issue regards how these schemes deal with phishing. They generally cite protection from phishing as a major feature, since users don't know the site passwords and therefore can't be tricked into revealing them.

While users don't check the identity of a web page before entering a password, the automation software will. The flip side is that the credentials for the one single sign-on are much more valuable than those of any particular web site, so if some phishing attack (or getting the user to run hostile software on his machine) can acquire that secret the damage is greater.

3.3 Identity Provider Federated Solutions

A variety of web-based schemes have been proposed (Liberty[25], web services[26], Shibboleth[15][21]) in which a user authenticates to an "identity provider", which provides the user with a credential that identifies the user to a service provider. In such systems, the service providers must be affiliated with the identity provider in a trust relationship (in a previously arranged "federation" arranged between the service provider and the identity provider), and the user must "link" the identity at the service provider with the user's identity at the identity provider.

As originally conceived, browsers were designed so that web sites could not interact with one another except in specific limited ways. A web site can store state associated with a user session – or even a long term memory associated with the site – in the form of a cookie on the user's machine. The short term memory enables users to log into a web site once rather than for each page viewed, but it is forgotten when the user logs out (of either the site or the browser), and not shared with other sites. The long term memory survives reboots and allows web sites to know what user last accessed the web site from a particular machine, but it's generally considered identification rather than authentication because machines are frequently shared. And like the short term memory, this is designed not to be shared with other web sites.

The cookie scheme works well for sessions within the pages of a single web site, but an SSO scheme can't create a cookie visible to web sites whose DNS names (contained in the URL) are not sufficiently similar to those of the sign-on site. The common workaround for this involves HTTP/URL redirection. If I enter the URL of a site into my browser, my browser will go to the site whose DNS name is embedded in the URL, and try to retrieve the data associated with the URL. A site can – if it chooses – provide a different URL for my browser to fetch instead. The URL to which I am "redirected" need not have any similarity to the original URL, so by generating a URL that contains data, this provides a mechanism for independent sites to interact. The second site can – if it chooses – give my browser a third URL to retrieve, with the chain extending indefinitely.

Most web SSO schemes work with some variant on the following:

- 1) When a web site that is part of a federation receives an http request, it looks for authentication information included as a session cookie or part of the URL. If there is none (and there won't be the first time through), it redirects the browser to the IDP (identity provider) site with a URL that begins with the name of the IDP site but includes (as data) the URL in the request.
- 2) The IDP site looks for a cookie indicating that the user is already "logged in". If so, it redirects the user's browser again, pointing it back to the original URL, this time adding the user's authentication information.
- 3) Now back at the original web site, there is still no cookie, but there is additional information in the URL authenticating the user, so the web site creates a session

cookie and returns it along with the page associated with the original request. In normal use, the two redirects take place so quickly that the user doesn't notice that anything extra is going on. The authentication takes place seamlessly.

If when the user reached step 2, he had not yet logged into the identity provider (i.e. no cookie), then instead of redirecting the user back to the original web site the identity provider authenticates the user using any of a number of techniques that could include username and password, but could also involve smart cards or X.509 certificates. After the user authenticates, the IDP sends the user's browser a web sign-on cookie and then redirects him back to the original web site.

This design requires no changes to the browser. Functionality put in the browser for other purposes is exploited to do the authentication. Web sites that want to join a federation and the identity providers must carefully coordinate what they are doing, but there could be many identity providers operating independently with collections of web sites behind each of them. They would not be aware of one another.

Several problems with this sort of approach are:

- There are limits on the maximum size of a URL and of cookies, and if applications are already running close to those limits adding authentication information can push them over. Remember though, that you can (with some cost in performance) always substitute a URL, an encryption key, and a cryptographic hash for an arbitrary amount of information and then store the real information at the URL encrypted with the key.
- A web site might want to affiliate itself with multiple identity providers and allow the users logged into any one of them to authenticate seamlessly. The problem is that the site doesn't know to which identity provider it should redirect a user. If the identity providers cooperate, there are ways of smoothing over this, but without browser changes or configuration restrictions there are no perfect solutions.
- Similarly, a user might have many identity providers, and might want to have accounts at many service providers, where not all the service providers are affiliated with the same identity providers. Currently advocated schemes are largely silent on the issue of dealing with multiple identity providers.
- Although in theory user authentication to the identity provider can be done with any sort of technology, it is usually deployed with a username and password. If it were done with a stronger mechanism, say smart cards, it would lose the benefit of being deployable on any browser from any machine. Given the fragility of server authentication to the user, if a user can be tricked into typing the user's identity provider username/password to a site impersonating the identity provider to the user, that evil site can then impersonate the user, not just at the identity provider, but at any affiliated site.
- If the identity provider is currently not reachable, there is no way for the user to attach to any of the service providers.

Identity Provider solutions do help the issue of service provider impersonation (phishing), because the Identity Provider site is not likely to have a relationship with the evil site, nor would the user have given permission for any of the user's information to get shared with the evil site. If the user has authenticated to the real identity provider, the user will not be tricked by unaffiliated service providers.

However, especially with user authentication based on username and password, the Identity Provider approach is very vulnerable to a user being spoofed by a site impersonating the Identity Provider, especially because the user will often not type in the URL for the Identity Provider, but instead be redirected to the page in which the user is prompted for his username and password. And the username/password at the Identity Provider is particularly disastrous to divulge to an impostor site, because compromise of that information compromises the user at all sites the user has linked with that identity provider account.

Looking at how well this sort of solution meets the other design goals, identity providers are unlikely to actually provide single sign-on, but rather a reduced number of sign-ons. That's because it is unlikely that a single identity provider will sign up all of the sites a user wants to access. Banks, brokers, and other particularly sensitive sites will likely affiliate with no identity provider or only with very specialized ones. These schemes are totally dependent on third parties, introducing both security and availability issues. They are likely not to work very well with shared accounts or users with multiple accounts because the http redirection technology is hard to make both flexible and transparent. They handle roaming well. They handle carrying specific kinds of user configuration information along like credit card numbers and other authentication-related information, but they are unlikely to deal with ad-hoc information like lists of favorite sites. They are well qualified to carry information that requires attestation, like age, memberships, and corporate affiliations, and can enhance privacy by proving affiliations without identifying the individual requestor to the web site. In principle, they should be able to do a good job of credential changes because credentials need only be revoked in one place (at the IDP). While client-only enhancements are largely transparent to web sites, Identity Providers are often transparent to the client software. Web sites can deploy them without requiring that users install any new software on their machines (which is particularly important when dealing with kiosks, specialized devices (like cell phones), and a user base running a variety of types of browsers).

3.4 CardSpace

CardSpace [4] is a system that allows a user to select one of several "cards" (signed assertions about various attributes of a user) to present to a service provider. These cards might be signed by an identity provider, in which case CardSpace becomes similar functionally to the systems we described in section 3.3. It improves on those schemes by having local storage for remembering things like what credentials are associated with which sites and by having the user be prompted with a difficult to spoof UI asking what information the user would like to pass to the web site. The cost of those security improvements is that it requires additional software be installed on all clients as well as all servers and it asks users lots of questions (breaking the transparency of other single sign-on solutions).

CardSpace provides another mode of operation with self-issued cards, which is like having a private identity provider on the client. This eliminates the need for any initial user registration with an identity provider, and the need to have another machine in the authentication loop. With self-issued cards, CardSpace is similar to one aspect of the scheme we propose in Section 4 (registering the user's public key directly with the service provider as a method of authenticating the user).

3.5 Controlling Access to User Information

Section 3.3 only discusses how the Identity Provider solution addresses single sign-on. An enhancement to the Identity Provider Federation solutions is to also store user information (such as address, credit card, phone number, etc.) at some location. It could be at the Identity Provider, or at another machine, or some of the user's information might be at one server, and some at another. For simplicity, we will assume all the user information is stored at the Identity Provider.

The vision is that the user can set policy on each item of personal information, as to which information should be made available to which web sites. Information that has been approved to be sent to a site is sent automatically to that site when needed.

This might be a mechanism for keeping user personal information more secure, because then the service provider would not need to keep the user's information in a database, and there would be fewer databases on the web with personal information for the user. Since each database has some probability of being broken into, the fewer places that user information is stored, the more likely it is that it will remain secure.

However, it is likely that service providers will keep the information. They might use the identity provider to retrieve the information when the user is first setting up an account, but it is in the service provider's interest (less bandwidth, latency, and work), to store the user information on the service provider site so that in subsequent interactions it would not need to be retrieved from the identity provider.

3.6 Authenticated Attributes

Some attributes are intuitively ones that need more proof than just having the user fill in fields on a form. Examples are:

- I am a member of an organization that is authorized to use the services of this site, for free.
- I am over 18.
- I am not a citizen of one of the countries that the company I am communicating with is barred from allowing downloads to.
- My credit card number is X.

Other attributes seem like they would not need proof, such as:

- Address
- Telephone number
- Email address

First, even attributes that seem like they would not need proof might, in certain situations. It would seem as though the "rent out my house for a week" service, or the "demolish my house" service would need some sort of proof of address. And the "register my

telephone number to be called every time this stock changes value" service ought to verify a phone number.

Email address is routinely verified by web sites, and they do this by sending a token to the web site, which you have to return by clicking on the link in the received email. This low tech solution is secure enough in practice.

For authenticating the attributes listed above as needing to be authenticated, there are only ad hoc and not very secure mechanisms deployed. For downloading export-controlled code, typically a check of IP address is made, with perhaps a form that you have to click agreeing that you are not a citizen of one of the frowned-upon countries. This does not prevent someone from using a proxy site to disguise his IP address, or downloading it at a public machine while visiting an approved country.

The identity provider solutions have a solution, in that identity providers can sign the attributes. However, this implies that there are sites out there that are trusted to assert all the needed attributes for a user, that the user has some mechanism for authenticating to those sites and requesting the signed attributes, and – perhaps hardest of all – there is some infrastructure in place that enables users to find them.

3.7 Strong Password Techniques

A strong password protocol solves the problem of doing mutual authentication using a weak secret (such as a password) in such a way that neither side in the exchange (client or server), or an eavesdropper, can do a dictionary attack. This form of mutual authentication might be a more reliable way of ensuring that the user is not fooled by an imposter web site than TLS, or TLS alone, because of the problems we mention in section 3.9. Because this form of authentication is mutual, TLS authentication is redundant except when establishing the shared secret initially.

The straightforward mechanism of sending a password does not do mutual authentication. There are other protocols such as CHAP[23] in which the password is used as a secret in a challenge/response protocol. Variants of such protocols can be used for either one-way authentication or mutual authentication. However, these sorts of exchanges are subject to dictionary attack, by at least one side in the exchange, and by an eavesdropper.

There are a number of cryptographic techniques for authenticating with a weak secret (a password) in a cryptographically strong way. The first of these was EKE[1], though there are others SPEKE[13], SRP[28], PDM[18].

Roughly, these protocols weave a password into a Diffie-Hellman exchange in such a way that someone impersonating either side can only verify one password guess during an authentication, and an eavesdropper cannot even verify a single password guess. A strong password protocol provides mutual authentication. Both the server and the user must know the password in order for authentication to complete.

For usability, it would be desirable for the user to use a single password at all the service providers. The obvious vulnerability there is that all the service providers could then impersonate the user at the other service providers (at which the user used the same password). To solve this problem it has been suggested (e.g., in PDM[18]) that the password at a site named "X" be a hash of the user's single password, and the string "X".

Problems with strong password solutions as currently advocated:

- The user must first install a password in a secure way at each service provider.
- If the user changes his password periodically, the user will wind up with unsynchronized passwords, and be confused about which passwords to use at which sites.

Strong password protocols might be useful as a mechanism of authenticating to the Identity Provider in a way that is more secure than sending the user's password directly, or somewhat more secure than doing a protocol such as CHAP in both directions. However, it does require changing software at both client and server machines.

3.8 Kerberos

Kerberos [16] is a secret key based system commonly used within an enterprise to authenticate non-web based clients and servers. There are a number of ways that it could be used for web authentication with different side effects.

In Kerberos-based systems, a KDC stores a secret (or with the PKINIT[30] enhancement, a public key) for each user, and a secret for each server. Each time a user wants to connect to a server, the user first contacts the KDC and obtains a (short-lived) ticket to that server.

The KDC can impersonate the user at any server in the realm (as can an Identity Provider), and although it is possible to do cross-realm authentication, Kerberos is mainly practical within an enterprise.

The initial authentication is usually done with a password, though PKINIT is used in organizations in which all employees are provided with smart cards.

A KDC is similar in some ways to an identity provider. However, with Kerberos, the ticket provides a shared key between the user's machine and the server, and a cryptographic mutual authentication exchange is done between the user's machine and the server. In contrast, with the http-based Identity Provider solutions such as Shibboleth and Liberty, the client machine is given secret information in a redirected URL, and the client machine sends that information to the server. Given that it is possible despite TLS, (for instance, by presenting a self-signed certificate claiming to be the server name and having the user click "OK") to impersonate a server to the user's machine, the Kerberos approach is somewhat more secure than the Identity Provider approaches, since the actual secret is not sent to a server.

One way to integrate Kerberos with web authentication is for an Identity Provider to use Kerberos tickets as the secrets it sends to servers. In this configuration, the browser client is unaware of the use of Kerberos. The Identity Provider goes to the KDC to get Kerberos tickets for the authenticated client to forward to the service. In this configuration, the security properties are essentially the same as with other Identity Provider schemes.

Another way to do the integration is for the client to authenticate to the server using Kerberos as an HTTP authentication mechanism within the TLS encrypted tunnel. This is most useful in scenarios where the clients and servers are already provisioned with Kerberos in support of other protocols. The security properties resemble those of other intranet protocols.

The challenge in both cases is scalability. It is logistically difficult to securely provision all of the components in the system with secret keys, and Kerberos is designed for configurations where all components are willing to fully trust a centrally managed service.

3.9 SSL Client Certificates

The SSL and TLS protocols provide for the ability to do mutual authentication (user to server as well as vice versa), but as deployed, authentication is almost always just server to user.

Public key based authentication of clients to servers eliminates the straightforward man-in-the-middle (phishing) attacks because the server learns nothing that would allow it to impersonate the user to another server. But the UI problem remains that users are unlikely to notice that they are connected to impostor web sites, so if they enter confidential information on a web page, that information might be given to an unauthorized server. Users can be tricked by web sites for several reasons:

- Confusion around server names, where a misspelled name might not be noticed, or worse yet, internationalized names can look identical to the expected name
- Confusion around URLs, where understandably users might not know which part of the URL is the real DNS name
- Faulty trust anchors, in which an evil-doer manages to trick any of the trust anchors into issuing a faulty certificate
- Also, because there are so many cases in which the web site the user wants to visit chooses to use a self-signed certificate or an expired certificate rather than go through the hassle and/or expense of obtaining a certificate from one of the commonly configured trust anchors, users have been trained to ignore security warnings around certificates and accept anything. Therefore, it would be relatively easy for a web site to present itself as any server name, using a self-signed certificate, and many users will merely accept the certificate signed by the "unknown trust anchor".

Client side PKI is not usually deployed for use between unaffiliated clients and servers over the Internet. It is used within organizations (where the user is an employee/student/etc. of the organization operating the server). Nevertheless, use of certificates between unaffiliated clients and servers is rare. There is work on Bridge CAs that might enable PKI use across organization boundaries, but it hasn't penetrated the broad population yet. There have been some CAs deployed to give users certificates. A typical method of doing this is:

- the user contacts the CA and tells the CA the user's email address and public key
- The CA sends a token to the email address
- The user, after receiving the email, sends the token to the CA (proving that the user has access to the email account)
- The CA sends the client the certificate, which is stored in the browser.

Why are client certificates not widely used in practice?

- It is confusing for users to obtain a certificate.
- Some CAs, wanting to be able to provide more assurance to their certificates, would require onerous procedures such as mailing them a notarized letter. They may also charge for certificates both initially and for renewals.
- Historically it has been hard to move certificates from browser to browser (vendors believed they were making certificate based authentication more secure by introducing this inconvenience) making roaming difficult.
- Users are likely to have different usernames with different service providers, and today browsers don't remember which certificate to send to a particular web site (the user must select it each time).
- Certificates are generally issued to some name for the user, for example, his email address, which might not be the same as the account name at the server, and might not even be stored as a property of the account, so it is not necessarily straightforward for the server to know which account is being accessed, based on the certificate.

Another issue with currently deployed PKI-based solutions is that TLS has no ability for the client to request a particular certificate from the server. If there were many CAs in the world, and a particular user chooses to trust only some subset of them, then there is no way for the server to be able to guess which certificate to send a user.

A fairly small change to TLS would enable this sort of negotiation. Such a change is proposed in [11], and a usable subset is specified in RFC4366 [2], but it has not been widely deployed.

One big advantage of PKI over the other solutions is that the CA need not be on-line, which is important for the reasons we gave in section 2.3.

3.10 Obtaining the user's private key

If user authentication is to be done using public keys, the first challenge is obtaining the user's private key(s). Unlike a password, it cannot be carried in the user's head (because user memorable passwords are nearly always weak keys, and weak private keys offer little value). Ideally a user would have a smart card, but there are problems with smart cards that have prevented them from being widely deployed:

- Users lose them, and need a backup plan for when the smart card is either not with the user, or permanently lost.
- Not all machines have a method for attaching a smart card.

If the user does not have a smart card, another approach is to store **{priv}pwd** (the user's private key encrypted with the user's password), in some location on the net (for instance a directory). There are various means of downloading the private key to the machine the user is using, using only a password.

- The simplest scheme is to have **{priv}pwd** be world-readable. If the user's password were sufficiently strong this would even be reasonably secure, but most users do not choose passwords strong enough to withstand an offline dictionary attack. Kerberos version 4 [14] was deployed in this mode (anyone could request a credential for the user which would enable an offline dictionary attack).
- The next enhancement is to have a simple challenge/response protocol (perhaps in a TLS-protected exchange based on server-side authentication) in which the user proves knowledge of her password before the server sends **{priv}pwd**. This is similar to the "pre-authentication" that was introduced in Kerberos version 5[16]. Someone impersonating the server would be able to do a dictionary attack, but others would only be able to do an on-line attack. Without TLS protection, an eavesdropper would also be able to launch a dictionary attack. However, in practice, it is more difficult to arrange to be eavesdropping on the private key download exchange than merely to send a message to the server requesting it to send **{priv}pwd**. When used with TLS server-side authentication, this approach would be quite deployable and quite secure today. Various schemes for doing this have been proposed (for example, Pvault [5] and SACRED [6]).
- Another approach would be to use a strong password-based credentials download protocol such as in [17].

Ultimately, of course, it would be desirable for users to have smart cards. USB connections are nearly ubiquitous today, and there are USB-based smart cards. Or phones can communicate with a work station using bluetooth technology.

With strong password techniques, someone impersonating the server or client can only do an on-line dictionary attack, and someone eavesdropping gets no opportunity for a dictionary attack. However, if someone were to steal the server database, then one would be able to do an off-line attack, since the server must store the user's private key encrypted with a password.

4. TOWARDS A USER-CENTRIC PKI

In this section we describe a number of ways of addressing, not only the single sign-on problem, but also the issue of providing user information in a way that is convenient, safe, and under control of the user. One aspect of our approach, doing mutual authentication without using 3rd party CAs, is philosophically similar to that used for email in [9], as well as to self-issued cards in CardSpace. We also address other issues, though, including one-stop revocation and switchover to a new user credential, that have not been addressed before.

4.1 User Information/User "Wallet"

Once a user has obtained a private key (either with a smart card or having downloaded the private key from the net and decrypted it with a password), we propose that information about a user be available on the net. We will call this information the user's "wallet". It can contain information such as the user's credit card number, address, and passport number. We will advocate that it also carry information that can enable any machine to recreate the environment of the user's home machine, such as browser bookmarks, preferences, and long-lived cookies.

This requires the installation of support software (perhaps a browser plug-in) on the host machine, which should be easy for the user's own machine and could be constructed so as to be safely installable on a kiosk machine.

The Liberty approach has user information stored at a server and retrieved by other servers. The server that stores the user's information has access to all the user's information at all times, so if that server is broken into, all user information for all users can be stolen.

Another approach is to store this information in the user's machine. The problem with this is that this information would not be available if the user were working at a different machine.

We advocate that the user information be stored at a server in the web, but that this information be encrypted with the user's public key. This information would therefore not be accessible to the server on which it was stored. This greatly increases security for the user.

We also claim that it will be easier for the user to reliably and conveniently decide policy for which web sites should be allowed to access which information than it would be if the user had to set up the policy at a remote server.

The user's machine should download and decrypt the user's wallet. When a web site requests information, the user's machine can let the user know what information is flowing, and the user can make decisions such as "let anyone see this information", "don't let anyone other than my machine access this information", "always let this web site see this information". It might be a burden if the user must make a yes/no decision for every piece of information, every time, but it can be made quite convenient, for instance, by having the wallet software fill in all the fields in a form, and then let the user change or erase fields. Also, for fields like "credit card number", the user could select from a list of credit cards.

This approach does require reaching a 3rd party in order to download the wallet (2.3) when logging in from a new client platform. However, this access only needs to take place once per login session (or once ever, from the user's home machine), as opposed to solutions such as those in section 3.3, which require accessing the identity provider for a token every time a service provider is accessed.

4.2 Least Privilege

Suppose a user is using a less trusted machine, and only wants to unlock a subset of his personal information. For instance, he may want his name and address book, but not credit card information.

If all information in the user's wallet is unlocked simultaneously, then there would be no way for the user to get enough information to print a boarding pass at the machine in the hotel lobby without also giving that machine credit card numbers and access to all the confidential files which the user is authorized to access.

This problem can be solved by having some of the information in the wallet be protected with an additional level of encryption, most likely based on a password. It would be burdensome to have every item of information separately locked, but it would be practical to have some of the information directly unlocked by the private key, and other information require an additional key to unlock. Or to have separate wallets stored on the network, one for low-risk information, and one for higher-risk information. The

user would separately authenticate and download the various wallets, presumably with different credentials.

For instance, if it is desired to support authorization based on which authentication method was used (as described in section 2.5), there might be a wallet that is unlockable with a password, and more secure information might be only unlockable with the private key in the user's activated smart card.

4.3 Establishing a relationship with a service provider

So far the user has a private key, and no certificates. Certificates are not strictly necessary for public key authentication. We propose instead a scheme where each server either certifies or stores the user's public key.

A user establishes an account with a web site through the following steps:

- The client machine makes an initial contact with the server, using current server-side authentication to be reasonably certain the user has reached the correct server.
- The user creates an account. The account is given a username, through some mechanism such as is done today. The account name on that server is stored in the user's wallet. If the user creates multiple accounts on the same server, the wallet keeps track of all the usernames, and software allows the user to select from these when the user visits that service provider.
- The user's machine sends the server machine the user's public key. The server either stores this information with the user's account, or certifies the public key and sends the client machine the resulting certificate, which the user stores in the user's wallet.
- The server sends its public key to the client machine. The client either stores the merchant's public key in the user's wallet, or certifies the public key and sends the server the resulting certificate.

4.4 Connecting to a service provider

Now assume that a user already has an account with a service provider. The user's machine has first downloaded the user's wallet. The user asks to connect to a service provider. The wallet software finds the account names for that user at that service provider, and if there is more than one, asks the user to select one. The user's machine and the server perform mutual authentication based on the server's and user's public keys.

Mutual authentication will be done with public keys. Either each side will have stored the other side's public key, and there is no need for certificates, or one or both sides can instead store a certificate signed by the other side, and present that certificate during subsequent visits.

When would it be the case that the service provider would sign a certificate for the user's key (and have the user store this certificate and present it during future authentications), rather than simply storing the user's public key with the account? A good use case for this is when the service provider might not store any information associated with the account, but instead sign a certificate authorizing use of the account to a particular public

key. For instance, a user might pay a site that stores information a monthly subscription fee, allowing unlimited free downloads during that month. The user might pay the site with anonymous cash, and create a public key (which the user keeps in her wallet) for use at that site, along with the certificate from the site which authorizes that public key to use the service for a specified time.

When might it make sense for the service provider to send the client machine a certificate (signed by the user), rather than having the user store the server's public key in the user's wallet? A possible reason for this is to keep the user's wallet smaller.

4.5 Unlinkable Accounts

A user might want it to be difficult (or preferably impossible) for two web sites to compare accounts and discover which accounts are for the same identity. If the user uses the same public key at different sites, it is evident that the accounts belong to the same user.

For this reason, a user might want to use different public keys to authenticate to different sites. This can be accomplished by using the "main" private key (the one stored in the user's smart card), for unlocking the user's wallet, and storing other public key pairs, and the sites on which each public key is to be used, in the user's wallet.

CardSpace automatically creates a different public key for signing self-issued cards, for each service provider.

4.6 Revocation of the user credentials

Suppose the user's smart card were stolen, and he wishes to change his private key. With our scheme, there is direct authentication between the user and a server. The user is acting as his own CA, and therefore there needs to be some sort of revocation mechanism. In CardSpace, in the mode where the user is his own identity provider, revocation would need to be done manually, with the user remembering every site with which he'd registered with the stolen credential, and having some method of authenticating, (without the stolen credential) to assert the authority to revoke the old credential.

This model is impractical, both because the user will not remember all the sites, and because the process of individually revoking the old credential at each of those sites would be onerous.

We advocate a one-stop revocation mechanism by use of a "revocation service". Such a service is trusted to inform servers about revocation status of a user's public key.

If a user needs to revoke his public key, he contacts the revocation service, authenticating with some sort of method other than using his private key (because he likely does not have access to his private key when he needs to revoke it). This might be with a phone call in which they call his home phone back, or through an email interaction similar to what is done today for password resets. This need not be super secure because the threat is a denial of service threat, not an impersonation threat, and the user has to go through the same sort of procedure as he would have needed to do in order to establish the account with revocation service in the first place.

In our scheme, when the user registers with a web site (see section 4.3), in addition to telling the web site his public key, he also tells the site the URL for the user's revocation service.

There are several possible ways in which a site can find out about the revocation status of a user's public key:

- Each web site can check the revocation status of each user account, communicating with that user's revocation server periodically, or each time the user logs in.
- The web site can register with the revocation service when the user creates the account, so that the revocation service will notify each of the user's web sites if the user's key has been revoked.

There is a potential privacy issue. The user might not want the revocation service to know all the web sites that the user is registered with. However, if he uses different keys for different sites, and possibly even different revocation servers for the different public keys, this will help. All revocation schemes face painful performance vs. timeliness trade-offs.

4.7 Key Rollover

Although section 4.6 addresses the problem of informing all the relevant service providers of the invalidity of the user's old public key, the user will want to be able to resume use of his account with a new public key.

One method of doing this is as follows:

- Before the user's current public key becomes unusable, the user creates a "next" public key pair.
- He certifies the next public key with his current private key. Call this certificate the "next key" certificate.
- He escrows the "next" private key by breaking it into n shares, with a quorum of k [22], encrypts each share with the public key of an escrow agent, and stores all the encrypted pieces in his wallet, or at some trusted storage location on the net, along with the certificate signed by his current private key that delegates to the next public key.
- If he needs to revoke his key, he obtains the escrowed "next" private key (through some out of band method of authenticating to the site that is storing the escrowed pieces, or authenticating to the escrow agents), and stores the "next key" certificate at the revocation service URL.

A web site that is checking for the user's key validity will either retrieve from the revocation service URL in the user's account a status of "public key is P", or one or more "next key" certificates.

If a web site has not checked the validity of the user's public key recently, it might encounter a string of "next key" certificates. The web site might be storing P for the user, and the revocation service URL might contain "P", "next key is P1" (signed by P), "next key is P2" (signed by P1), and so forth.

While the reassembly of the "next key" and revocation of the previous one can be automated, the process by which a user chooses escrow agents and later authenticates to them cannot. It's intended that the genuine user can complete this procedure while the person who has stolen the user's credentials cannot; that makes it inherently messy and fragile. Making the user experience straightforward, secure, and robust will be a real challenge.

4.8 Preventing Loss of the Wallet Information

If the user loses his smart card, and the user's wallet is only stored encrypted with the key in the smart card, the user will lose all this important state.

For this reason, it would be valuable to escrow the user's private key, or escrow some other key with which the user's wallet information is encrypted, if the user would not like to share knowledge of his private key with the escrow agent.

Escrow of a key can be done quite securely. The key can be broken into shares using a quorum scheme [22] such that k out of n shares can recover the secret. Each of the shares is then encrypted with the public key of each of n escrow agents, and stored someplace safe. The encrypted pieces need not be given to the escrow agents until the key needs to be recovered.

4.9 Authenticated Attributes

Some of the systems we covered in section 3, (e.g., Shibboleth) have some capability for authenticated attributes, such as proving membership in an organization whose members are allowed to use the service for free. In theory a user could acquire signed credentials, by someone authorized to assert such attributes, and present them to a service provider when required to access a service. These credentials could be acquired at time of access to the service (as in Shibboleth), or collected and stored in the user's wallet as certificates (asserting the attribute to a public key owned by the user). If acceptable to the service, the user's privacy could be protected by only proving the required attribute and not revealing the user's identity. There are issues, as we discussed in section 3.6, such as how a user finds a site that is trusted by the service provider to assert the attribute. In theory there might be a single site that is completely trusted by a service provider to make decisions about all user attributes. This attribute assertion site could make complex decisions about all the attributes, in whatever way is appropriate for each, and then simply make assertions to the service provider.

A fully general solution to authenticated attributes has not been proposed or deployed, and is largely orthogonal to what we propose here. If solved, it could be added to the authentication, revocation, and credential rollover solutions we propose here.

5. CONCLUSIONS

In our solution there is no need for federations of identity providers and service providers. Also, no traditional PKI is needed, except for initial contact by a user with a service provider as provided with TLS today (and which would be a necessary step in an Identity Provider solution as well).

The basic ideas are:

- The user carries his private key around on a smart card, or downloads it from a server.
- Once the user obtains his private key, he downloads the rest of his "wallet" information from a server, which stores it encrypted with the user's public key.
- When a user wishes to enroll with a web site, the user's machine exchanges public keys with the server, and either certifies the server's key with the user's private key, or stores the server's public key in the user's

wallet. Likewise, the server either stores the user's public key, or certifies the user's key and gives the user's machine the certificate to store and present on future visits.

- Authentication to a server with which the user has an ongoing relationship is done using public keys.
- When the user creates an account at a service provider, in addition to telling the service provider the user's public key, the client also tells the service provider an associated revocation service.
- The service provider registers with the revocation service if it wishes to be notified when the user revokes his public key. (or else, the service provider checks periodically).
- If the service provider registers with the revocation service, the revocation service informs the service provider if the user's key has been revoked.

Properties of this approach:

- The user has the perception of single sign-on, which is done when the user activates and inserts his smart card, or downloads her private key from a server (goal 2.1).
- Authentication to web sites is strong, so no credentials can be compromised by phishing (goal 2.2).
- Contact between a user and a server can be done directly. There is no need for redirection to and from identity providers. This is more efficient (goal 2.3)
- Contact between a user and a server works even if no other machines other than that server are currently reachable (goal 2.3).
- User information is still available from anywhere on the web, but encrypted, so that the user's information is safe and directly under the user's control (goals 2.3, 2.5, and 2.6).
- There is no on-line trusted service whose compromise can enable user impersonation (goal 2.3).
- There is no on-line trusted service whose compromise can leak private information for large numbers of users (goal 2.3).
- Credential backup through a quorum of backup agents safely protects against credential loss (goal 2.7)

These enhancements do require changes to both clients and servers (failing to meet goal 2.8), but they can be deployed incrementally with benefits accruing with each step.

6. ACKNOWLEDGMENTS

We wish to thank Eve Maler, Ray Perlner, and the anonymous reviewers for helping with background information and helpful comments.

7. REFERENCES

- [1] Bellare, S. and Merritt, M. 1992. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, May 1992. pp. 72-84
- [2] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and Wright, T., 2006. Transport Layer Security (TLS) Extensions. Internet RFC 4366 April, 2006. DOI=<http://www.ietf.org/rfc/rfc4366.txt?number=4366>.
- [3] Cantor, S., Kemp, J., Philpott, R., and Maler, E. 2005. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS Standard March 2005. DOI=<http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>.
- [4] CardSpace. DOI=<http://msdn2.microsoft.com/en-us/library/aa480189.aspx>.
- [5] Chandra, R., Mehrotra, S., and Venkasubramanian, N. 2005. Pvault: A Client Server System Providing Mobile Access to Personal Data. Proceedings of the 2005 ACM workshop on Storage security and survivability (StorageSS), 2005.
- [6] Farrell, S. 2004. Securely Available Credentials Protocol. Internet RFC 3767 June, 2004. DOI=<http://www.ietf.org/rfc/rfc3767.txt?number=3767>.
- [7] Farrell, S. and Housley, R. 2002. An Internet Attribute Certificate Profile for Authorization. Internet RFC 3281 April, 2002. DOI=<http://www.ietf.org/rfc/rfc3281.txt?number=3281>.
- [8] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and Stewart, L. 1999. HTTP Authentication: Basic and Digest Authentication. Internet RFC 2617. June, 1999. DOI=<http://www.ietf.org/rfc/rfc2617.txt?number=2617>.
- [9] Garfinkel, S., Miller, R., "Johnny 2: A User Test of Key Continuity Management with S/MIME and Outlook Express", Symposium on Usable Privacy and Security, 2005.
- [10] Herzberg, A. and Gbara, A. 2004. Security and Identification Indicators for Browsers against Spoofing and Phishing Attacks. Cryptology ePrint Archive, Report 2004/155. DOI=<http://eprint.iacr.org/2004/155.pdf>.
- [11] Hess, A., Jacobson, J., Mills, H., Wamsley, K., Seamons, K.E., and Smith, B. 2002. Advanced Client/Server Authentication in TLS. Network and Distributed System Security Symposium, San Diego, CA. February 2002.
- [12] Housley, R., Polk, W., Ford, W., and Solo, D. 2002. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Internet RFC 3280 April, 2002. DOI=<http://www.ietf.org/rfc/rfc3280.txt?number=3280>.
- [13] Jablon, D. 1996. Strong Password-Only Authenticated Key Exchange. Computer Communication Review, Vol. 26, no. 5, ACM SIGCOMM. October 1996. pp. 5-26
- [14] Kaufman, C., Perlman, R., and Speciner, M. 2002. Network Security: Private Communication in a Public World. Prentice Hall PTR. pp 307-336.
- [15] Morgan, R., Cantor, S., Carmody, S., Hoehn, W., and Klingenstein, K. 2004. Federated Security: The Shibboleth Approach. Educause Quarterly. pp. 12-17
- [16] Neuman, C., Yu, T., Hartman, S., and Raeburn, K. 2005. The Kerberos Network Authentication Service (V5). Internet RFC 4120. July, 2005. DOI=<http://www.ietf.org/rfc/rfc4120.txt?number=4120>.
- [17] Perlman, R. and Kaufman, C. 1999. Secure Password-Based Protocol for Downloading a Private Key. Proceedings of the 1999 Network and Distributed Systems Security. February 1999.
- [18] Perlman, R. and Kaufman, C. 2001. PDM: A new Strong Password-based Protocol. 10th USENIX Security Symposium, August 2001.
- [19] Ross, B., Jackson, C., Miyake, N., Boneh, D., Mitchell, J., "Stronger Password Authentication Using Browser Extensions", 14th Usenix Security Symposium, 2005.
- [20] Rubenking, N. 2003. PC-Mac PasswordVault 2.0. PC Magazine, 2/13/03.
- [21] Scavo, T. 2005. Shibboleth Architecture: Technical Overview. Working Draft 01, January 9, 2005. DOI=<http://shibboleth.internet2.edu/docs/draft-scavo-shibtechoverview-01.pdf>.
- [22] Shamir, A. 1979. How to Share a Secret. CACM vol 22. no. 11, pp 612-613.
- [23] Simpson, W. 1996. PPP Challenge Handshake Authentication Protocol (CHAP). Internet RFC 1994. August, 1996. DOI=<http://www.ietf.org/rfc/rfc1994.txt?number=1994>.
- [24] Sxipper, <http://www.sxipper.com>.
- [25] Tourzan, J. and Koga, Y. 2004. Liberty ID-WSF Architecture Overview. Version 1.0. Liberty Alliance Project.
- [26] Web Services Architecture. 2004. W3C Working Group Note. DOI=<http://www.w3.org/TR/ws-arch/>.
- [27] Wu, M., Miller, R., and Garfinkel, S. 2006. Do security toolbars actually prevent phishing attacks? Proceedings of the 2006 SIGCHI conference on Human Factors in computing systems. 601-610. DOI=<http://portal.acm.org/citation.cfm?id=1124863>.
- [28] Wu, T. 1998. The Secure Remote Password Protocol. Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium. March 1998. pp. 97-111
- [29] Yee, K., Sitaker, K., "Passpet: Convenient password management and phishing protection", Proceedings of the second symposium on Usable privacy and security (SOUPS), 2006.
- [30] Zhu, L. and Tung, B. 2006. Public Key Cryptography for Initial Authentication in Kerberos (PKINIT). Internet RFC 4556. June 2006. DOI=<http://www.ietf.org/rfc/rfc4556.txt?number=4556>