PorKI: Making User PKI Safe on Machines of Heterogeneous Trustworthiness*

Sara Sinclair and Sean W. Smith
Department of Computer Science
Dartmouth College
{sinclair, sws}@cs.dartmouth.edu

Abstract

As evidenced by the proliferation of phishing attacks and keystroke loggers, we know that human beings are not well-equipped to make trust decisions about when to use their passwords or other personal credentials. Public key cryptography can reduce this risk of attack, because authentication using PKI is designed to not give away sensitive data. However, using private keys on standard platforms exposes the user to "keyjacking"; mobile users wishing to use keypairs on an unfamiliar and potentially untrusted workstation face even more obstacles.

In this paper we present the design and prototype of PorKI, a software application for mobile devices that offers an alternative solution to the portable key problem. Through the use of temporary keypairs, proxy certificates, and wireless protocols, PorKI enables a user to employ her PKI credentials on any Bluetooth-enabled workstation, including those not part of her organization's network, and even those that might be malicious. Moreover, by crafting XACML policy statements that limit the key usage to the workstation's trustworthiness level, and inserting these statements into extensions of the proxy certificates, PorKI provides the user or the relying party with the ability to limit the amount of trust that can be put in the temporary keypair used on that workstation, and thus the scope of a potential compromise.

1 Introduction

Using client-side *public key infrastructure (PKI)* can solve many problems for user authentication to remote services, as well as enable signature and encryption functionality. However, its effective application in large enterprises requires several preconditions, including:

1. Private keys must remain private.
2. Private keys must be used only for the operations the users intend and authorize. (Secrecy of a private key does not help much if the adversary can still use it at will.)
3. The PKI must integrate with the standard desktop/laptop computing environments users and enterprises already employ.
4. Users should be able to use PKI from any machine in the enterprise.
5. These machines may have varying levels of trustworthiness. (E.g., in a university environment, these machines may range from dedicated, well-maintained single-user machines all the way to de facto public access workstations.)

In this paper, we present research and prototype results for our *PorKI* project, intended to solve this problem of portable PKI.

Background. The *keyjacking* work of Marchesini et al showed that conditions #1 and #2 fail in situations satisfying #3 [11]. However, even if an enterprise solves the problem of securing a user's private keys at one standard machine, it is still necessary to make user PKI *portable*, and to accommodate the fact that not all machines in the enterprise will have the same level of trustworthiness.

Marchesini et al followed the keyjacking work with a partial solution, *Secure Hardware Enhanced MyProxy (SHEMP)* [12]. SHEMP uses a central hardened repository to store a user's keypair along with an *eXtensible Access Control Markup Language (XACML)* policy. This policy maps machine types to appropriate key uses (per user and according to enterprise preferences). System administrators can sign certificates for machines they administer, indicating degrees of trustworthiness; more trustworthy machines will also employ stronger measures to keep their keys private. When a user wishes to access remote services from a given client machine, the repository uses the user's main private key to sign a *proxy certificate (PC)* for a short-lived keypair on that client, and specifies the appropriate use restrictions as extensions in that PC.

SHEMP addresses the keyjacking problem by automatically and transparently limiting the damage that a weak client can cause. However, SHEMP requires the deployment and scalability overhead of this centralized repository. It also provides no effective way for a user to authenticate to the repository via a potentially untrustworthy client: authentication is by password, which could be keylogged if the workstation is compromised.

*This research has been supported in part by Intel and the NSF (CCR-0209144, EIA-9802068). This research program is a part of the Institute for Security Technology Studies, supported under Award number 2000-DT-CX-K001 from the U.S. Department of Homeland Security, Science and Technology Directorate. This paper does not necessarily reflect the views of the sponsors.

Our Approach. To solve the problem of portable PKI in heterogeneous trust environments, we start with the SHEMP framework. However, we remove the centralized repository, and add an additional set of elements to the mix: each user carries a Bluetooth-enabled PDA or smartphone, which plays the role of the repository for that user—and also provides a trustworthy channel to authenticate the user to the repository. This approach also provides the foundation for additional capabilities to aid users in making trust judgments.

This Paper. Section 2 starts by considering the problem of authentication to remote services from the user's perspective. Section 3 presents the design of our PorKI solution, and Section 4 discusses the prototype. Section 5 discusses potential applications of this tool. Section 6 considers prior work in this space, and how PorKI measures up. Section 7 concludes with some directions for future work.

2 User Authentication

One of the most natural ways for humans to authenticate in the digital world is through something they know. However, humans are not adept at choosing or protecting their standard credential, passwords. We encourage users to choose strong passwords and change them often and design systems to force them to do so. However, no matter how strong a password, it cannot be protected if the user gives it away. One recent study showed that 80% of college students sampled were willing to exchange their password for a plastic dinosaur or a squirt gun. Another found that 93% of participants (including faculty members) were willing to enter their password into a website that was spoofed to look like an official college page [20]. In the past year, such phishing attacks in the wild have grown from a rare occurrence to a commonplace affair. In February of 2004, the Anti-Phishing Working Group collected a total of 282 unique phishing attack reports; in February of 2005, the same organization collected reports on 2625 active phishing sites, almost a tenfold increase [8, 9].

Stronger Solutions for Authentication. Two-factor authentication schemes, which require that a user both have something and know something in order to authenticate, can reduce the risk of social engineering attacks that take advantage of humans' weakness at making trust decisions. RSA's SecurID system requires that users posses a device that generates passcodes in a predictable yet hard-to-guess manner. RSA maintains that this scheme can help prevent phishing attacks from being successful, for even if a user is willing to give away her SecurID passcode at one moment, it will not be valid at a later time [6]. Because SecurID mimics the password interface– the only difference is that the user doesn't have to be responsible for generating the secure passcode– this system is very usable.

Proponents of PKI also advocate that public key cryptography can enable authentication that is more secure than a single-factor password approach. In the case of PKI, the user possesses a certificate signed by a Certification Authority (CA) that binds the user's keypair to the identity or permissions put forth in the certificate. The private key of this keypair is necessary for authentication, but authentication can be performed without sharing the private key. Thus, although a site performing a phishing attack might be able to convince the user to attempt to authenticate using her keypair, she would not give away her credentials in doing so. Keyjacking attacks against private keys exist; however, by the very nature of PKI it is much more difficult to convince a user to give away a key than it is to give away a password. PKI also offers other secure functions, such as encryption and digital signing of documents, in addition to authentication. And, unlike SecurID, no proprietary software or hardware devices are necessary; furthermore, a user can use the same credential storage device (be it a software keystore or a hardware token) to store keys that enable her to authenticate to multiple relying parties.

A Solution for Users. Unfortunately, PKI is notorious for being difficult to implement in a usable way. Although security experts are fluent in the language of keypairs and certificates, these concepts are not intuitive to the average user. In particular, traditional PKI schemes pose a big problem with regards to portability; most relying parties expect users to store their keys in a software repository on their personal machine, and make no allowance for users who have multiple machines or need to authenticate from multiple locations. Specialized devices, such as smartcards, offer some portability, but requiring users to purchase and carry extra devices with likely limited functionality will not contribute to the popularity of the solution. Users want to be able to have multiple keys for use with multiple relying parties; they want to be able to use their keys anywhere without worry of compromise; in short, they want to be able to be mobile and use their keys securely in environments of heterogeneous trust.

3 PorKI Design

The goal of the PorKI system is to present users with a usable and secure way to store and use their private keys, without requiring the purchase of special devices, but while enabling them to make safe trust decisions in heterogeneous environments.

We will now consider the PorKI design, including an overview of the system followed by a more in-depth consideration of the repository, the workstation policy infrastructure, and the generation and transfer of the temporary credentials.

3.1 System Overview

The generation and transfer of X.509 PKI credentials to a workstation from a PDA PorKI repository is accomplished in the following manner:

1. A user pairs her PDA with a workstation to initiate a Bluetooth connection between the two devices. (In Section 3.4, we will discuss the implications of current weaknesses in the Bluetooth pairing handshake.)
2. The user unlocks the PorKI repository.
3. If the workstation is equipped to provide PorKI with a credential regarding its status, the user launches an application on the workstation that will enable it to authenticate to the PDA using its keypair.
4. The user chooses a key in the PorKI repository she would like to use to issue a temporary credential. If only one is available, this one is selected by default.
5. If the user's key usage policy allows this issuance to this workstation, then the PorKI PDA prepares a temporary credential.
   (a) An XACML policy statement about the workstation and user-policy-specified key usage limitations is generated.
   (b) A temporary keypair is generated.
   (c) A proxy certificate is generated that testifies to the public key of the temporary keypair, including a limit on the duration of its validity. The proxy certificate has an extension that contains the XACML policy statement.

   (In Section 3.3, we discuss these policy techniques further.)
6. The proxy certificate and temporary keypair are transfered to the workstation via Bluetooth.
7. The temporary credentials are imported into a keystore on the operating system.
8. The Bluetooth connection is closed.

We note that the user need only interact with a device on steps (1) through (5). Furthermore, step (3) requires interaction only if a PorKI-supporting application exists on the workstation, and step (5) only if it does not and the user is required to import the temporary credentials manually. In Section 4 we consider what the working PorKI prototype indicates about time it will take to accomplish these steps.

3.2 Long-Term Key Repository

A user often receives a long-term keypair from the organization that issues her a certificate for it; other times she generates it on her personal workstation. (PKI standards provide for users to generate their own keypairs and have them certified by multiple organizations. In practice, however, key generation often becomes part of the certification process and is sometimes done by the certificate issuing authority.) Because of this, we envision that a user will most often choose to import her keypair into her PorKI repository on her PDA after the keypair is generated elsewhere (although PorKI is capable of generating keypairs on the PDA, should the user choose). We discuss the possibility of PorKI playing a larger role in the certificate-issuing process in Section 5, but we note that an organization should be able to come up with a reasonably secure way to get the keypair to the PDA. These methods would likely mirror those currently used to transfer keypairs to general-purpose workstations or specialized devices.

Many PKI users find themselves with a proliferation of certificates and keypairs. As more organizations move to PKI as an authentication mechanism, it is possible to envision a user needing to access a collection of keypairs from a variety of different organizations. The PorKI repository stored on the PDA is thus equipped to keep multiple credentials and to allow users to choose which keypair they would like to use at a given time.

The current PorKI design provides for a single step to unlock the PorKI repository, with a password. This generally protects the repository from compromise in the case of theft of the PDA; although using a password for protection opens the repository to dictionary or social engineering attacks, it is still the most intuitive way for a user to authenticate. The current design of PorKI cannot protect against more advanced attacks, such as those targeted at the PDA hardware. Thus, a savvy attacker in possession of the PDA could (with enough time and knowledge) almost certainly compromise the repository. By that time, however, we hope that the user would have noticed its absence and the credentials stored on it would have been revoked. In Section 7 we will consider how the repository might be protected from this kind of attack as well, although it is generally beyond the scope of this project.

Because portable devices are prone to being lost or stolen, many users might wish to back up the keys stored in their repository, if just to have a record so they can report the loss for revocation purposes. To this end, PorKI design includes the ability to export the keystore in a protected format (similar to the mechanism currently available in many web browsers). The keys should only be exported to a trusted workstation. From this workstation, users should transfer the copied repository directly to an offline format, such as a CD in the proverbial sock drawer.

3.3 Policy Based on the Workstation

Passwords provide a binary form of authentication: one either has the required knowledge or one does not; one's

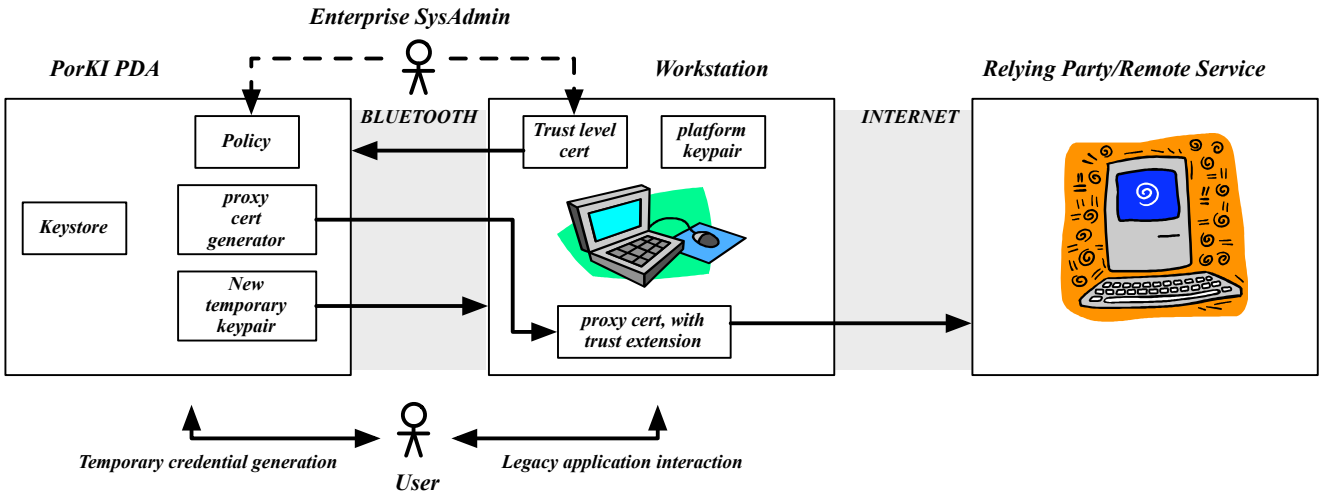

Figure 1. Sketch of an example use of the PorKI system. Here, an enterprise officer helps craft the key usage policy and also provides a workstation trust-level certificate.

actions are either trusted or they are not. However, in environments of heterogeneous trustworthiness, it is obvious that this model is not sufficient. A user could have multiple passwords with which to log into an online resource account from differently trusted machines: one password that allows him read privileges, one that also allows him write privileges, and one that also allows him to change the permissions of other users. It is true that this model would provide the relying party to which the user is authenticating a good way to limit the damage if the user is authenticating from on an untrusted workstation where the passwords might be compromised. However, users want to maintain the smallest set of credentials possible. More importantly, users are often ill-equipped to make trust decisions of when to use their credentials at all.

PorKI takes the task of providing relying parties with information about the trustworthiness of the authentication environment out of the user's hands. Judgments on the trustworthiness of a workstation are done programatically through standard PKI methods: if a workstation is in possession of a keypair and list of characteristics, and if these credentials have been signed by a certificate authority that PorKI trusts, then PorKI can use that list of characteristics to craft policy statements.

Following the model of SHEMP, a PorKI PDA will generate an XACML policy statement if a workstation is able to provide it with such a trustable list of characteristics. This set of characteristics might include facts about the workstation's physical location (in public library, in a private office), its virtual location (behind a firewall), its hardware (is it enhanced with a trusted computing platform?), or its software (when was the last time it received critical operating system patches?). This requires that the signed list of characteristics be in a format that PorKI can parse and interpret, such as a standardized XACML statement. We assume that if there exists a path of trust between PorKI and the workstation, it is because PorKI has been equipped to interpret statements signed by the issuer of the workstation's credentials.

In the original SHEMP scheme, users (perhaps in conjunction with their enterprise administrators) constructed a key usage policy. This policy specified what usages (both in terms of functionality—e.g., "client-side SSL"—as well as application-specific constraints—e.g., "but not for Privacy Act or HIPPA data") were acceptable for what types of environments. The environment types would include a default "untrusted" type, for machines with no administrator-issued certificates, all the way to higher-security SELinux/TCG platforms (e.g., [10]) with TPM hardware protecting the platform's private key and binding its use to platform software satisfying certain properties. The key usage policy allows a user to pre-define a matching between workstation trustworthiness and key usages. If a platform can be coaxed or tricked into somehow sharing its credential and private key with another, then presumably (and hopefully) the vulnerability that permitted this would be implicitly reflected in the trust level specified by the workstation's credentials. Therefore, only machines of low trustworthiness should be exploitable in this manner, with highly trustworthy workstations better able to maintain the integrity of their credentials.

In PorKI, we extend this approach to also put the workstation characteristics themselves within the policy extension in the proxy certificate. The relying party that receives

the PC thus possess this information during authentication; its system administrators, having pre-defined a set of authentication policies tied to various workstation types, are able to dynamically limit the amount of trust awarded to a user when she is authenticating. This embodies a sound principle of secure system design: the best-qualified party should make trust decisions. In SHEMP, users were able to define their own key usage policies according to workstation type. It would seem that users might tend to create more open policies for the sake of usability: "I wanted to make sure I could have full privileges from any workstation," not knowing the security sacrifice they might be making for the whole system. By also enabling the relying party to limit the capabilities of a particular session based on the workstation information presented at authentication, PorKI enables the relying party to counteract this user tendency. By retaining the user-defined key usage policy, it also still allows users to set limits on trust should they choose to do so.

Once the policy statement is transfered via the proxy certificate, in order for the relying party to decide what permissions the user should be granted, the relying party must be able to parse the XACML policy statement, and have in place permission profiles for the user that correspond to various levels of workstation trustworthiness. Figure 2 shows what a small set of permission profiles might look like in abstract form. We see that user `ssmith` has administrative privileges when authenticated from a TMP-protected server, whereas user `ssinclair` has user privileges. Both users have read-only access when authenticating from an untrusted workstation.

It should be straightforward for the relying party to interpret a well-formed policy statement if it and the workstation maintainer use the same syntax for expressing characteristics. And, again, if the relying party has a path of trust to the workstation maintainer, then we hope that they have been able to agree on a characteristic mapping from one organization to another.

Considerations of policy expression across organizational boundaries is an important problem for many areas of PKI research, and establishing a scheme to facilitate mapping expressions for PorKI is an area of future work for this project. Projects such as the Higher-Education Bridge Authority [5] facilitate the exchange of trust among independent organizations, and these efforts may increase interest in developing cross-institutional mappings.

This brings to attention, however, the possibility that the relying party not have a path of trust to the workstation, even if the user does. Because SHEMP was only intended to be deployed within an individual organization, PorKI must expand the model to also include information about the maintainers of the workstation in the policy statement. For example, say Alice is a part-time professor at both institutions A and B. Her PorKI repository can find a path of trust to a workstation w at A, and is able to use the signed set of characteristics given to her by w to craft a policy statement p. Now, when she tries to authenticate to a resource belonging to institution B, we cannot automatically assume that B has reasons to trust A's machines, or that B has agreed with A on what constitutes a "patched" machine, for example, even if they use the same general format.

In the worst case, if the relying party is unable to establish a path of trust to the workstation, if there is an incompatibility in the policy statement, or if the workstation simply does not have a certificate to offer the PorKI user, the policy statement can be left at a default "untrusted" (either in the PC or in the interpretation of the unrecognized credentials by the relying party). Because proxy certificates are X.509-based and PKIX standards-compliant, a PorKI user should be able to use her temporary credentials with any relying party that supports X.509 infrastructure and the set of Internet community standards. (Compatibility with legacy applications depends on the certification path validation algorithm implemented by the application and on the specific PKI of the organization. Not all will have support for Proxy Certificates; restrictions, including those on certification path depth or key usages in the PKI, could also hamper proper validation.) This means that a PorKI user can authenticate (albeit with low-security permissions) from virtually any workstation using her PDA as easily as she can from a highly trusted workstation within her organization's network.

3.4 Credentials and Transfer

Once PorKI has received information from the workstation and crafted a policy statement, it is able to generate the temporary credentials. PorKI currently generates the keypair on the mobile device. However, there may be cases in which it might be desirable to perform this generation on the workstation, to reduce the computation load on the mobile device, to reduce the transfer time of the credentials, or to reduce the risk of key compromise during the transfer. Work by Boneh et al [2] includes a scheme for generating keypairs on PDAs with the help of an untrusted workstation; luckily, because PorKI allows the workstation full access to the temporary key without risk, the entire task of key generation could be offloaded.

PorKI transfers the temporary credentials in a standard format, such as PKCS12, so that they are easy to manually import into an OS or web browser keystore. This means that no client software is necessary for the workstation; PorKI uses the Bluetooth file transfer protocol (which most Bluetooth-enabled workstations implement by default) to transfer the credentials. We can also envision the user downloading a small application that serves as a signing and

User	Workstation Type	Permission Level
ssinclair		
	TMP-server	USER
	Personal-comp	USER
	Public-comp	USER
	Untrusted	READ-ONLY

User	Workstation Type	Permission Level
ssmith		
	TMP-server	ADMIN
	Personal-comp	ADMIN-NO-CHMOD
	Public-comp	USER
	Untrusted	READ-ONLY

Figure 2. Example profiles mapping workstation types to permissions for the remote resource.

repository utility, either directly from PorKI on the PDA or from the relying party's website. We could also eventually envision a web application by a trusted remote party that interacts directly with PorKI over Bluetooth for authentication.

Bluetooth was a very attractive medium for PorKI credential transfer; it traverses operating systems and platforms wirelessly, and is fast becoming the de-facto standard for communication among low-power devices. It is also easy and inexpensive to retrofit a workstation for Bluetooth communication. However, despite these benefits, there is considerable concern regarding Bluetooth's security; the pairing process and its use of PINs has come under particular scrutiny recently, with Shaked and Wool [19] describing a passive attack that cracks 4-digit PINs used in pairing in 0.06 seconds on a Pentium IV.

The simplest way to improve security of the Bluetooth connection is to use a longer PIN in pairing, thus increasing the amount of time necessary to crack it. PorKI could eventually even force a user to use a longer pin by simply choosing a random one and displaying it so the user can input it on the other device. However, this does not protect against repeat attacks in which the adversary has days to crack the pin before the PorKI user comes back to use a workstation again (Bluetooth devices usually "remember" their pairings from one session to another). However, as soon as PorKI is able to exchange public keys with the workstation, an additional layer of encryption beyond the Bluetooth default can protect communication between the two devices. Alternatively, we could consider using another wireless medium to transfer the credentials- perhaps near field communications (NFC). In Section 7, we consider this other methods of securing the trust bootstrapping process between PorKI and a workstation.

Again, one of the benefits of using temporary and limited-capability credentials is that even if the Bluetooth connection or the workstation were to be compromised, the window during which an adversary could make use of the credentials for malicious purposes is small. This fact notwithstanding, it is imperative to the integrity of the system that the Bluetooth flaws be compensated for; as it stands, Bluetooth is the "weak link," and compromise of the communication channel would be a compromise of the whole system.

4 PorKI Prototype

The current prototype of PorKI runs on an Apple Powerbook equipped with a 1.33 GHz G4 processor using the built-in Bluetooth device. The key storage and credential generation is performed in Java, but the Bluetooth communication is implemented in Objective C, so as to take use of the OS X Bluetooth libraries (Bluetooth implementation for J2SE is not standard, although it is in J2ME.)

The feature set of this prototype is minimal, as we wanted to determine the feasibility of the project before moving to a full implementation on a PDA, which is the next step in our development. Currently, PorKI is able to pair with a workstation, generate a temporary keypair and a proxy certificate, sign them with a long-term private key, and transfer them to a workstation in PKCS12 format. It is also able to import keys, and provides the user with an interface to choose which key in the repository to sign. It protects the repository in a Java Keystore.

4.1 Performance

Key Generation. As we noted in Section 3, we considered while designing PorKI the possibility of offloading the temporary keypair generation onto the workstation. To estimate the load on a PDA, we measured how long it takes to generate the temporary credentials on the Powerbook, and found that it took 1.67 seconds to perform both the key and proxy certificate generation and signing, with slightly over half that time being spent just on the key generation. If we assume all other things being equal, transferring the PorKI software directly to a PDA with a 200 MHz processor (the speed of a lower-end Bluetooth-enabled business model at this writing) would yield credential generation time of roughly 11 seconds, about half of which is dedicated to the keypair.

While this amount of time is certainly significant—the lag would be very noticeable to a user—we note that Boneh

et al [2] cited 1024-bit RSA key generation (the same task being performed in PorKI) as taking 15 minutes on a PDA in 2000. If the computation power of PDAs has improved that much in five years, we can only imagine that key generation in PorKI would soon not create a noticeable lag.

Click Counts. Because we have not yet developed software to interface with PorKI on the workstation, we rely on the operating system keystore to manage the temporary credentials. Additionally, we rely on the OS X Bluetooth device discovery utility to identify the workstation to which we would like to transfer credentials. Bearing in mind the fact that a more integrated version of PorKI will require less user interaction, we counted how much was necessary to generate, transfer, and import credentials. PorKI itself required two clicks to launch, two clicks to initiate device discovery and choose an alias, and a password to unlock the keystore. Device discovery took three, but could easily have taken one with a customized interface, for a grand total of five including launch. Once on the workstation, the Windows certificate import wizard took eight clicks and a password, whereas Apple's Keychain utility only used four and a password.

This interface is not as seamless as that of a USB token or a smartcard, which usually just require one click and a password. However, PorKI offers the added functionality of being able to store keys and certificates from many different sources, in addition to the added benefits associated with policy statements about the workstations. Thus, we feel that the added interaction is worth the increased functionality.

5 PorKI Applications

We can envision several scenarios in which PorKI could directly replace existing authentication schemes while providing a security advantage. We can also envision more advanced scenarios in which PorKI could be adapted to perform other tasks. We will consider examples of both of these in turn.

Base Form Applications. One scenario in which PorKI could be easily applied is with highly mobile students in a diverse computing environment. This application is particularly motivating, given our location. Right now, passwords are the norm, and efforts to make USB tokens mandatory have been stymied by lack of token driver support for non-Windows operating systems.

Another scenario is that of a computer repair person with contracts at several offices. She might need to authenticate to her own office network to gain access to software tools or billing information, but could not be guaranteed that the client's computer would have any particular software installed, or even that it was to be trusted.

Potential Future Applications. Other scenarios could use an expanded version of PorKI to accomplish more advanced tasks. For example, a professor might want to delegate access to a grades database to his TA. If both users had PorKI installed on their portable devices, he could accomplish this quickly and easily by bringing these devices in proximity to one another and issuing a few short commands. To stretch things even further, we can envision that a certification authority might issue certificates directly to a PorKI keystore, in a way reversing usual flow of information in PorKI. This form of certificate issuance could potentially be much simpler than those for USB PKI devices, in the case that a CA requires a user to appear in person to receive their certificate.

6 Prior Work

In this section, we will consider some principal previous work in the space of dongle and PDA-enabled password alternatives that provide authentication capabilities.

RSA SecurID Tokens. One of the first authentication alternatives on the market was the SecurID card (initially from Security Dynamics, but now from RSA). SecurID provides two-factor authentication with a userid and a passcode-generating token. In addition to being available on specialized devices, a token can also be installed on a PDA as a software program [17]. However, SecurID requires the relying party be running specialized server-side software, and the purchase of seeds for every token deployed. Additionally, it is difficult for a user to use his SecurID token for multiple relying parties.

Mobile Phones and Proxy Servers. Wu, Garfinkel and Miller presented a highly mobile authentication application for use on potentially hostile workstations [26]. This system, which makes use of a mobile phone and a remote trusted proxy server, can be used to authenticate to any password-using relying website from any workstation. However, the protocol between the relying party, the proxy server, and the mobile phone is somewhat complex, and it is unclear how this affects the speed at which users can authenticate. The several steps of human interaction required may affect the overall usability of the system. Moreover, because the user is still providing passwords to the relying party, this solution does not prevent a user from exposing her credentials in a phishing attack, nor does it help her in making trust decisions.

In earlier work, Clarke et al explore camera-based solutions [3]. (Gobioff et al explored an even more primitive version of the problem [4].)

USB Tokens and Smartcards. A simple way to make a public key cryptography keypair portable, and thus take advantage of PKI's strengths over passwords as an authentication measure, is to put a user's private key on a hardware device, such as a smartcard. The private key is available to the workstation once the smartcard is inserted, and unavailable

upon removal. However, smartcards require special readers and drivers to interface with the workstation; although the newer incarnation, USB PKI tokens, no longer requires a special hardware interface, the need for drivers can be an obstacle to using this scheme in new environments.

These portable PKI devices prevent compromise through theft by requiring authentication to unlock them, either through a password via the workstation operating system or directly via a biometric, such as a thumbprint; they are also designed to be resilient against hardware tampering. Because the keypair is issued directly to the device, it is easy for the issuer to keep a secure backup; because the interface software is designed to not allow export of the key, it is hard for the user to give away her credentials.

However, if the user is using a compromised workstation, it is not impossible that vulnerabilities in the operating system might allow an adversary access to the private key that is stored on the device. Indeed, the keyjacking work by Marchesini et al showed that keys stored on the Spyrus Rosetta USB token and the Aladdin eToken were vulnerable to attack through the Windows CryptoAPI (CAPI) system, which those devices use to enable the interface between the private key and applications on the workstation [11]. PKI devices such as these do not aid the user in making trust decisions, nor do they offer relying parties with a way to judge whether the private key may have been subjected to keyjacking by a malicious workstation.

Furthermore, in our experience with pilot deployments of USB PKI dongles, it took over three minutes to issue a key to a user. In large-scale deployments, this lag imposes a significant time penalty to using this technology.

PDA Signing Proxy. Work by Balfanz and Felten [1] showed that a PDA running a specialized software application could be used as a smartcard with added benefits. Their system is beneficial when used on untrusted workstations, because the PDA can perform private key operations without ever giving the workstation access to private key or to sensitive data that has just been decrypted. Also, because the user authenticates directly to the PDA to unlock the private key, the smartcard vulnerability to an attack of password logging and device theft is eliminated (although the security of the private key if the PDA is stolen is more in doubt, because standard PDAs are not hardened against hardware attacks). Although it was not implemented in [1], the interface between the PDA and the workstation could be via a standard wireless protocol, thus removing the need for specialized drivers. However, transferring data between the workstation adds overhead to PKI operations, and PDAs are less well-equipped to perform compute intensive cryptographic operations than workstations.

Further work by Oprea et al [15] uses a trusted PDA to delegate credentials to an untrusted workstation that allows it temporary and limited access to resources on the user's home computer. The PDA and the home computer have exchanged certificates out-of-band and can thus communicate securely through the untrusted workstation. They use this channel to agree on a single-use password for the workstation to use, as well as a time limit on the connection of the workstation. Here, the PDA maintains a direct link to the home computer for mouse/keyboard events, so the workstation cannot inject anything into the communication. The workstation is only used for reading and viewing things on the home computer.

Section 7 will discuss some ongoing work by Reiter, van Oorschot, and others on PDA-based authentication, and how we plan to pursue incorporating some of those ideas in PorKI.

SHEMP. As discussed earlier, In an effort to create a portable PKI authentication scheme that reduces the vulnerability to keyjacking, Marchesini developed SHEMP [12], a key repository application that allows users to store their long-term private keys on a centralized server with hardware protection, and delegate their credentials to workstations using temporary keys signed by X.509 proxy certificates [23, 25]. Because the private key never leaves the SHEMP repository, the extent of the damage a malicious workstation is capable of is limited by the abilities afforded to the temporary keypair by the PC. SHEMP builds on previous MyProxy credential repository systems [14, 7].

Although SHEMP uses temporary keys on workstations, it can still enable decryption or signing with the private key by allowing the user to transfer the necessary data back to the server for those operations. Thus, keypairs stored in SHEMP, like keypairs stored on smartcards, are good for more than just authentication.

However, SHEMP also requires that the workstation have the SHEMP client installed, and that it have access to a SHEMP repository server. The need for a centralized repository brings with it all concerns of a centralized server, including the need for a system administrator to maintain it and unavailability of the private keys should the server go down. As noted earlier, SHEMP also does not provide a secure way for the user to authenticate— the user must enter a password via a potentially untrustworthy client.

Finally, SHEMP, like smartcards, is not ultimately portable; it is designed to be run on workstations within an organization's network. Modifications could enable a user to gain access to her private key from an out-of-network workstation, if she were to install the SHEMP client there and configure it to connect to the appropriate SHEMP server. However, this installation could be non-trivial and impractical for a user wishing to authenticate for a short amount of time.

7 Conclusion and Future Work

PorKI is an effective and highly usable solution to the problem of secure authentication among mobile users. It both improves on the security of current authentication schemes through the use of X.509 PKI, and adds a new dimension of trust expression to the authentication process through its use of XACML policy statements. It is our hope that its simple yet flexible design will encourage other developments of a similar nature to enable secure authentication in heterogeneous environments.

In the future, we envision several directions for expansion of PorKI. These include advanced delegation capabilities and the ability for a certification authority to issue a key directly to the PorKI repository, as discussed in Section 5. We might also allow a PorKI user to further limit the trust level expressed in a policy statement of a proxy certificate beyond that which would normally be allowed a given workstation, in order to follow the principle of least privilege. We consider in this section few other areas of potential expansion.

Email Signing and Encryption. Currently, PorKI does not provide for digital signing using the long-term private key, nor does it provide users with a mechanism for decrypting something that has been encrypted with their long-term public key. The temporary credentials cannot be used for signing due to their short validity period. Similarly, only the long-term private key can decrypt that which has been encrypted with its public half.

A simple solution would be to transfer the data upon which these operations need to be performed directly to the PDA. Given the increase in computation power of PDAs as noted in Section 4, this is not an infeasible solution, although it is less desirable because the need to transfer so much data could incur significant overhead. We would like to pursue alternative schemes that does not involve so much data transfer, as well as work further on implementing this one.

Repository Protection. Protecting the repository on the PDA with a simple password is an effective scheme, provided that the password is well chosen and not shared. However, entering a password into a PDA is not a very quick process, either by stylus or miniature keyboard. It would be advantageous to consider alternative of passcodes, such as a series of gestures with a stylus or other forms of "graphic passwords" [22] or a biometric solution, if one had a PDA with a fingerprint-reading attachment.

Beyond password alternatives, it would also be worth considering employing the PDA's operating system for additional measures of protection, depending on the features it provided.

Methods of protecting the repository when the PDA is subjected to advanced hardware attacks is a difficult proposition. Encryption of the repository on the PDA helps, but is likely not sufficient. The only way to truly prevent this kind of attack is by using a hardened PDA. No such device is currently available, although the U.S. Government has contracted to have one built [21].

7.1 Bluetooth Trust Bootstrapping

As we indicated in Section Section 3, it is imperative that a user choose a strong PIN when pairing two Bluetooth devices. We suggested that PorKI might help the user by generating a long random PIN, but this would clearly be cumbersome for the user to input into the workstation. To reduce the risk of compromise, we would thus like to explore alternate ways of establishing trust between PorKI and the workstation over Bluetooth. This might include more creative and usable ways to generate PINs, but it might also involve a more PKI-oriented approach, by which the PDA and the workstation exchange public keys and use some additional method to verify that the exchange was not subject to a man-in-the-middle attack. This might include the use of a visual hash of what the PDA received, which can be verified by the user if it is also displayed on the workstation, as in [16]. It could include instead a way for the PDA to receive a hash out-of-band which it can then verify, as in [13], where McCune et al use a camera built into a smartphone to take an image of a visual hash displayed on a workstation' screen.

An alternative to trying to secure Bluetooth for PorKI would be to simply use a different wireless transfer mechanism, such as near field communications (NFC) [18]. NFC implements communication at very short ranges (its proponents bill it as "touch-based"), although it is not yet clear exactly how much this quality improves its security over other similar protocols. Also, Bluetooth is currently the most widely supported and widely implemented short-range wireless protocol; also, its file-transfer mechanism is very useful to PorKI when no client software is installed on the workstation. It is clear that if a new and more secure protocol becomes widely adopted, it would be easy to replace the less secure Bluetooth transfer. In the meantime, we will focus primarily on using the tools we have at hand to make an effective and deployable solution.

7.2 Location-Aware PorKI

With the rise of wireless devices has come the potential for "location-aware" computing. If devices can tell where they are, this awareness of location can serve to help users perform tasks, or aid in designing secure systems. In [24], van Oorschot and Stubblebine propose location aware devices as a mechanism for further verifying that a user is who he says he is and where he says he is, with an interest

to preventing identity theft. Integrating location awareness into PorKI could enable users to employ the keys stored in their repository to accomplish such tasks as gaining authorization to physical locations.

References

[1] Dirk Balfanz and Edward W.Felten. Hand-Held Computers Can Be Better Smart Cards. In *Proceedings of USENIX Security*, pages 15–24, Washington, DC, August 1999.

[2] Dan Boneh, Nagendra Modadugu, and Michael Kim. Generating RSA Keys on a Handheld Using an Untrusted Server. In *INDOCRYPT '00: Proceedings of the First International Conference on Progress in Cryptology*, pages 271–282, London, UK, 2000. Springer-Verlag.

[3] Dwaine E. Clarke, Blaise Gassend, Thomas Kotwal, Matt Burnside, Marten van Dijk, Srinivas Devadas, and Ronald L. Rivest. The Untrusted Computer Problem and Camera-Based Authentication. In *Pervasive '02: Proceedings of the First International Conference on Pervasive Computing*, pages 114–124. Springer-Verlag, 2002.

[4] H. Gobioff, S.W. Smith, J.D. Tygar, and B.S. Yee. Smart Cards in Hostile Environments. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, pages 23–28, 1996.

[5] Higher Education Bridge Certification Authority. http://www.educause.edu/hebca/.

[6] RSA Security Inc. Protecting Against Phishing by Implementing Strong Two-Factor Authentication. http://www.antiphishing.org/sponsors_technical_papers/PHISH_WP_0904.

[7] M. Lorch, J. Basney, and D. Kafura. A Hardware-secured Credential Repository for Grid PKIs. In *4th IEEE/ACM International Symposium on Cluster Computing and the Grid*, April 2004.

[8] Dan Maier. Phishing Activity Trends Report, March 2004. http://www.antiphishing.org/APWG.Phishing.Attack.Report.Feb2004.pdf.

[9] Ronnie Manning. Phishing Activity Trends Report, March 2005. http://www.antiphishing.org/APWG_Phishing_Activity_Report_Feb05.pdf.

[10] J. Marchesini, S.W. Smith, O. Wild, J. Stabiner, and A. Barsamian. Open-Source Applications of TCPA Hardware. In *20th Annual Computer Security Applications Conference*. IEEE Computer Society, December 2004.

[11] J. Marchesini, S.W. Smith, and M. Zhao. Keyjacking: the Surprising Insecurity of Client-side SSL. *Computers and Security*, 4(2):109–123, March 2005. http://www.cs.dartmouth.edu/~sws/papers/kj04.pdf.

[12] John Marchesini and Sean Smith. SHEMP: Secure Hardware Enhanced MyProxy. Technical Report TR2005-532, Department of Computer Science, Dartmouth College, February 2005. ftp://ftp.cs.dartmouth.edu/TR/TR2005-532.pdf.

[13] Jonathan M. McCune, Adrian Perrig, and Michael K. Reiter. Using Camera Phones for Human-Verifiable Authentication. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 110–124, May 2005.

[14] J. Novotny, S. Tueke, and V. Welch. An Online Credential Repository for the Grid: MyProxy. In *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*. IEEE Press, August 2001.

[15] Alina Oprea, Dirk Balfanz, Glen Durfee, and D. K. Smetters. Securing a Remote Terminal Application with a Mobile Trusted Device. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, Tucson, AZ, December 2004.

[16] Adrian Perrig and Dawn Song. Hash Visualization: a New Technique to improve Real-World Security. In *International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC) 1999*, 1999.

[17] RSA Security Inc. *RSA Security Unveils Innovative Two-Factor Authentication Solution for the Consumer Market*. http://www.rsasecurity.com/press_release.asp?doc_id=1370.

[18] Phillips Semiconductors. Nokia, sony and phillips establish the near field communications (nfc) forum. March 2004. http://www.semiconductors.philips.com/news/content/file_1053.html.

[19] Yanic Shaked and Avishai Wool. Cracking the Bluetooth Pin. In *MobiSys 2005*, 2005. http://www.eng.tau.ac.il/~yash/shaked-wool-mobisys05/.

[20] Sean W. Smith. Probing User-End IT Security Practices—via Homework. *The Educause Quarterly*, 27(4):68–71, November 2004.

[21] IT-Observer Staff. NSA PDA for the Government. *IT-Observer*, August 2005. http://www.it-observer.com/articles.php?id=848.

[22] Julie Thorpe and P.C. van Oorschot. Graphical Dictionaries and the Memorable Space of Graphical Passwords. In *Proceedings of USENIX Security*, 2004.

[23] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. Internet X.509 Public Key Infrastruture Proxy Certificate Profile, 2003. http://www.ietf.org/internet-drafts/draft-ietf-pkix-proxy-10.txt.

[24] P.C. van Oorschot and S. Stubblebine. Countering Identity Theft Through Digital Uniqueness, Location Cross-Checking and Funneling. In *Financial Cryptography and Data Security 2005*, 2005.

[25] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, and F. Siebenlist. X.509 Proxy Certificates for Dynamic Delegation. In *3rd Annual PKI R&D Workshop Pre-Proceedings*, pages 31–47, April 2004.

[26] Min Wu, Simson Garfinkel, and Rob Miller. Secure Web Authentication with Mobile Phones. In *MIT Project Oxygen: Student Oxygen Workshop 2003 Proceedings*, 2003.