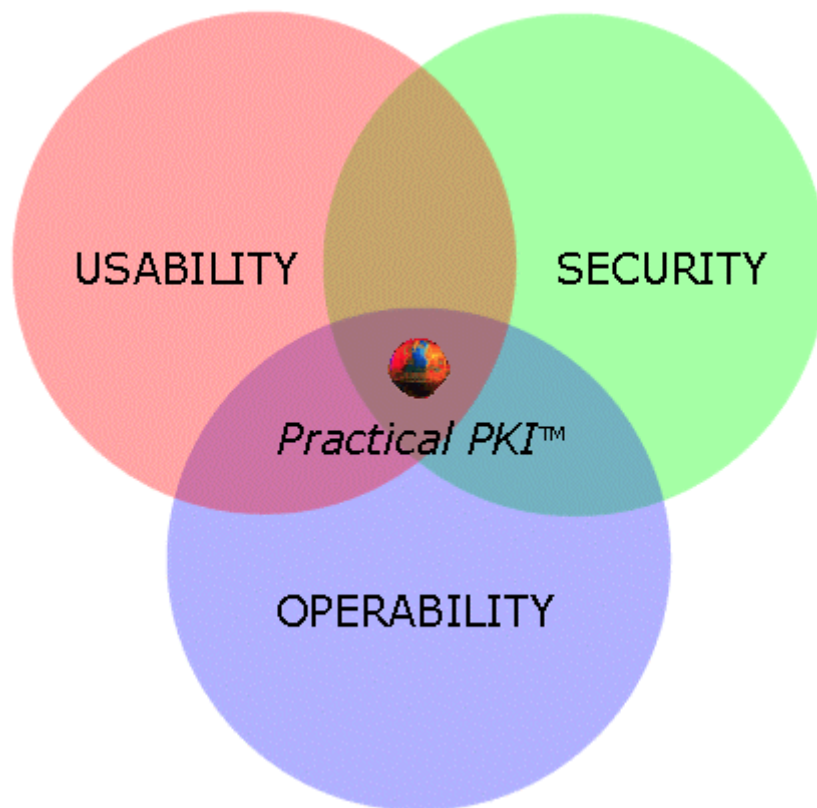


Towards Practical PKI™

A White Paper from SingleSignOn.Net, Inc.



Contact:

Harish Bhatt, CISSP
VP – Business Development
SingleSignOn.Net Inc.
11417 Sunset Hills Road
Reston, VA 20190
Tel: 703-904-4159
Email: hbhatt@singlesignon.net

Table of Contents

- **The Trouble with PKI**
- **Goals of Practical PKI™**
- **The Science – Before the SingleSignOn.Net Product**
- **The Science – The SingleSignOn.Net Solution**
- **The SingleSignOn.Net Product**
- **Using the SingleSignOn.Net Product**
- **Summary**

The Trouble with PKI

Which came first – airbags or cars? Parachutes or planes? Safety in general follows functionality, and with computer and communications security this is almost always the case. If the lag time is small, and eventually security “catches up” then this is a reasonable, if not desirable, situation. However, the pace of change on the Internet, and now Wireless networks, is such that one can actually make the pessimistic argument that security is not catching up, instead its being left further behind!

Security is a big topic, and in this white paper we shall focus on authentication, privacy and non-repudiation. Even in this smaller subset of security functionality the efforts to implement security have not been very successful.

Consider:

- In the physical world consumers usually manage to do all they need to do, with one or two simple authentication mechanisms like driver’s licenses and passports. In the digital world they drown in a sea of passwords.
- In the physical world a merchant opening up a store has many issues to worry about, but issuing IDs to every customer, is not one of them. Imagine a mall where every store required you to have a store card; unthinkable, yet this is the norm on the Internet today.
- In the physical world your bank will not ask for a different method of ID depending on whether you drove to the bank in a car, or rode in a bicycle. Yet, the same bank is forced to expect the customer to authenticate themselves in different ways if they arrive at their cyber-branch over the Internet versus over a wireless phone.

We could continue, but we think the point is self evident that common security functions like authentication, privacy and non-repudiation simply have not been seamlessly integrated into the fabric of the digital world we are building.

So why do we have this state of affairs? Is the **technology** missing? Well no, the core underlying technologies needed to implement authentication, privacy and digital signatures, for example public key cryptography, have

been around for twenty-five years! So is there a **demand** problem? Well, not really – ask any user of computers, consumers on the net, or employees in an organization if they enjoy having the multiple passwords that they have, and they will respond that they live in password hell and are sick of it. Maybe there is a **supply** problem? Hardly. There are a large number of companies offering security technologies and there is a vast array of organizations from driver's license agencies to financial institutions to portals to telcos to many others who want to be the trusted agent deploying the digital driver's license to consumers. Perhaps there is an **infrastructure** problem; it's no use having car manufacturers and people who want to buy cars if the highways do not exist. Even this is not true; consider that almost every browser and web server has built in support for the so called **public key infrastructure (PKI)**, almost every email client has support for secure email.

The technology exists, the infrastructure has been built out, the demand is high and there is eagerness to supply; yet successful PKI never seems to happen. Why is this the case?

The reasons are complex in their details but to summarize the primary problem is simple:

Security solutions are architected by security experts; and their goal is to solve the security problem.

You may well ask: so what's the problem? The problem is akin to designing cars with the sole goal of solving the transportation problem while having the designers all be experts in internal combustion engines! Instead cars are designed (most of the time anyway!) by people who focus hard on the usability, making the car easy to drive safely, and easy to learn to operate by virtually anyone. They are also designed to be operable. Automobile manufacturers understand that the owner and the mechanics that service the car and keep it in operation have no desire to become internal combustion engineers.

The same is also true for security products. They have to be easy to use, or they will not be used. They have to be easy to deploy and operate or else they will not get deployed! So the design of security products involves not just the goal of security, but also goals of usability and operability. The reason PKI has been a non-starter to date, is because existing solutions have been designed purely with security in mind.

We coined the term *Practical PKI™*, to define PKI solutions, which operate in the intersection of usability, operability and yet maintain “PKI-strength” security. This is not as easy to do as it may appear at first glance (else it would have been done already!) This white paper describes what we believe to be the first system that meets many of these goals. Let us describe the specific goals in greater detail.

Goals of Practical PKI™

As we have discussed Public Key Infrastructures (PKI) have been around for a while, and critical components of the infrastructure have been built out; for instance, most web browsers and servers have the ability to allow consumers to use digital certificates and most email clients have the ability to send signed and secured email. Yet, with notable exceptions (e.g. *server side SSL*), this infrastructure remains unused, consumers live in password hell, secure email is hardly used and organizations lack the ability to use digital signatures for electronic commerce.

This is the case because many of today’s PKI solutions suffer from five fundamental problems. A *Practical PKI™* solution is one which solves these five problems, but which still works with all the infrastructure and standards that already exist today.

So, what are the five fundamental problems?

Problem #1: Not easy for consumers to use

Current PKI systems usually require the consumer to somehow ‘carry around’ a very long private key. In the absence of ubiquitous, standardized, easy to install and use smart cards and smart card readers, current solutions are too difficult for the average user.

Practical PKI™ Requirement #1: Provide the convenience of passwords, but with the security of PKI. The user should not have to carry around smart cards (and readers) or leave their private key stored on the hard drive of a computer. For many years, passwords were downright unfashionable in the cryptography community. But, in the last year the PKI industry seems to have come around to deploying solutions that rely on passwords. However, not all password-based solutions are created equal. Most of these systems use the password to retrieve a long private key to the consumer’s PC, a

solution that does not lend itself to effective fraud management. Also we do believe smart cards have a role, and a PKI system should work with passwords today, and seamlessly upgrade to take advantage of smart cards where possible.

Problem #2: Long and complex implementations

Far too many PKI projects languish in “pilot” mode after implementations that drag on for months, even years.

Practical PKI™ Requirement #2: The implementers should be able to go from receiving the system from the vendor to actually distributing IDs in hours. Yes, integration with existing systems is not trivial, but the system itself should be: “take out of the box, plug into network and switch on!”

Problem #3: Difficult to manage

Current PKI systems are often not designed with mission critical reliability in mind and are very complex to operate securely. Further, they usually do not implement the critical principle of *least privilege*. Do you believe that system, network and database administrators of a computing system should have carte blanche access to the system? Should they or an attacker who compromises their accounts have access to sensitive consumer information such as credit card numbers?

Practical PKI™ Requirement #3: The system should be designed for mission critical availability. This usually means the system is designed with strong input from people who have worn pagers and run mission critical systems. The system should enforce the principle of *least privilege*. For instance, the system and security managers should be able to run the system without having access to customer information.

Problem #4: Poor fraud management

Current PKI systems usually have no way of detecting the theft of an ID before the consumer reports it (contrast this with how telephone calling card or credit card fraud is usually detected). Current PKI systems often do not even have instant revocation once the theft is reported, and finally, put the burden on the acceptor of a digital signature to verify if the signer’s ID has been reported stolen.

Practical PKI™ Requirement #4: First, provide the opportunity for “**velocity checking**” to try and detect theft of an ID before the consumer knows it’s

been stolen. Second, provide for instant, complete revocation of an ID once it's been stolen. In recent years, some leading cryptographers have proposed that the burden of checking validity be shifted from the recipient of a signed message to the sender of the message. We agree. In fact we believe, that once an ID is reported stolen, it should simply not be possible to *use* it again.

Problem #5: Current PKI Systems do not provide for reuse of security infrastructures

Most of the *cost* of a security system is in the infrastructure (people and processes) not product. Most of the *weaknesses* are in the infrastructure, not product. Given these problems, it is not realistic to assume that organizations can deploy multiple security infrastructures. Yet today there seems to be the belief that the PKI for wireless can be different from the PKI used on the Internet.

Practical PKI™ Requirement #5: Design reusable security infrastructures that work across both wireless and Internet channels.

But, ...don't reinvent the wheel!

Practical PKI™ Requirement #6: The first five requirements address the problems that most current systems fail to address. However, we believe that any new approach should take complete advantage of the existing infrastructure and standards, and should be interoperable with other PKI systems.

To summarize:

Practical PKI™ is described as a system with the following goals:

1. Easy to use
2. Quick to deploy
3. Simple to manage
4. Velocity checking and instant revocation
5. Reusable for wireless
- ...

But, work with what exists!

The Science – Before the SingleSignOn.Net Product

In conventional or symmetric cryptography, the participants, say Alice and Bob, share a common secret key k that is known only to them. To encrypt a message Alice typically uses some encryption function E and key k to compute the ciphertext $C=E(M,k)$. She then sends C to Bob on a public channel. Bob can decrypt the message using a decryption function D , i.e. $M=D(C,k)$. Typically, the details of E and D are public, but an attacker Eve, who only sees C cannot recover M since Eve does not know k . How do Alice and Bob agree on the shared secret k ? This is known as *the key distribution problem*.

The Theory: Before PKI







Alice and Castle Corp. can use shared "session key" derived from password for authentication and privacy

The Problems

- How to agree on password?
- How to protect the keys at Castle Corp.?
- How to prevent Alice from repudiating transactions?

In public-key or asymmetric systems each entity has a private key known only to that entity and a public key which is assumed to be publicly known. Thus, Alice and Bob each have private keys P_A and P_B respectively and public keys U_A and U_B respectively.

The Theory: Public Key Cryptography

<u>Public keys</u>	<u>Private keys</u>
 Alice	 Alice
 Bob	 Bob
 Castle Corp.	 Castle Corp.

- Privacy
 - Alice *encrypts* message with Bob's public key.
 - Only Bob (using his private key) can *decrypt*.
- Signatures and authentication
 - Alice *signs* message with her own private key
 - Anyone can *verify* signature using her public key

Let us denote the public key encryption function by \underline{E} and the decryption function by \underline{D} . The system has the special property that once a message is encrypted with a user's public-key, it can only be decrypted using that user's private-key. Conversely, if a message is encrypted (or signed) with a user's private-key, it can only be decrypted (or the signature verified) using that user's public-key.

So, if Alice wishes to send an encrypted message to Bob, she "looks-up" Bob's public key U_B , computes $C = \underline{E}(M, U_B)$ and sends C to Bob. Bob recovers M by using his private-key P_B and computing $M = \underline{D}(C, P_B)$. An attacker who makes a copy of C , but does not have P_B , cannot recover M .

Public-key cryptosystems are not very efficient, however, and typically cannot be used for encrypting large messages. Consequently, public and secret key cryptosystems are usually used in conjunction. Alice uses a symmetric encryption function E and session key k to compute ciphertext $C = E(M, k)$. To send the message to Bob, she further encrypts the session key k using Bob's public key with the public key encryption function to get $K = \underline{E}(k, U_B)$. She then sends (C, K) to Bob. Bob first recovers k using his private key, i.e., $k = \underline{D}(K, P_B)$ and then decrypts the message using the symmetric decryption function D , i.e. $M = D(C, k)$. This solves the key distribution problem, without sacrificing the efficiencies of symmetric cryptosystems for large messages.

Public-key cryptosystems can also be used for achieving technical non-repudiation, i.e. a method of "signing" a message such that the signer cannot later repudiate having signed the message. The signer, Alice, computes $S = \underline{E}(M, P_A)$ and sends (M, S) to the recipient Bob. Bob "looks-up" Alice's public-key U_A (often conveyed with the signed message), and then checks to see if $\underline{D}(S, U_A)$ is equal to M . If so then Bob is convinced that Alice signed the message, since computing an S , such that $\underline{D}(S, U_A) = M$ requires knowledge of Alice's private key P_A which only Alice knows. Bob can retain the signature as proof to prevent Alice from later repudiating the message. Moreover, any third party (other than Bob or Alice) can also verify the signature by checking $\underline{D}(S, U_A) = M$. For efficiency typically a one way hash function H is used to hash the message and it is the hash $h = H(M)$ that is signed, not the message itself. Further, by combining the encryption and signature functions, both privacy and non-repudiation can be achieved.

We now review an example, RSA, of a public-key cryptosystem.

RSA is a public-key based cryptosystem that is believed to be very difficult to break. In the RSA system the pair of numbers (e_i, n_i) is user i 's public-key and the number d_i is the user's private key. Here $n_i = p \times q$ where p and q are large primes and $e_i \times d_i \equiv 1 \pmod{\phi(n)}$, where $\phi(n) = (p-1)(q-1)$ is the Euler Totient function which returns the number of positive numbers less than n_i that are relatively prime to n_i (two numbers are relatively prime if they have no common factors greater than 1, for instance 9 and 14 are relatively prime, whereas 13 and 26 are not). After key generation p , q and $\phi(n_i)$ are destroyed. To encrypt a message being sent to user i , user j will compute $C \equiv M^{e_i} \pmod{n_i}$ and send C to user i . User i can then compute $C \equiv M^{d_i} \pmod{n_i}$ to recover M .

The RSA based signature of user i on message M is $S \equiv M^{d_i} \pmod{n_i}$. The recipient of the signed message can compute $M \equiv S^{e_i} \pmod{n_i}$, to verify the signature of i on M . Note that in RSA encryption and signatures can be combined.

So far, we have repeatedly referred to how a user can "look up" another user's public-key. An obvious solution would be to put everyone's public-key in some sort of universally accessible database. This raises the problem of how to secure the connection between the user and the database, as well as how to secure the database itself. The concept of certificates is an elegant way of solving both problems— it addresses the first problem directly and makes the second problem irrelevant.

A certificate is basically a binding between an entity and its public key, as vouched for by some authority. So a certificate in an RSA based infrastructure could contain $Cert = \{I, e_i, n_i\}$. The certificate is signed by a trusted third party called a Certificate Authority (CA). So when i sends j a signed message $S \equiv M^{d_i} \pmod{n_i}$, it is accompanied by sender's certificate signed by the CA's private key. The receiver j can verify i 's public key from the certificate using the CA's public key.

The Theory: Public Key Certificates

Certificate Authority (CA)

- A trusted third party.
- Everyone knows the CA public key.
- It issues “certificates”

ID: Alice
FN: Alice
LN: Smith
Email:alice@cc.com

• Binds Alice's identity to her public key.

• Alice gives certificate to Castle Corp. along with her signature.

• Castle Corp. can verify certificate using the CAs public key.

But what if Alice's private key becomes invalid?

In any infrastructure, some keys will be compromised from time to time, and it then becomes necessary to *revoke* a certificate that has been issued. The issuer of the certificate may have other reasons for revoking the certificate, for instance if an employee leaves an organization, the organization may wish to revoke any certificates granted to the employee. Two solutions have been proposed. One suggestion for solving this is to use the concept of a Certificate Revocation List (CRL). Roughly speaking, ever so often (say once a day), a CA will broadcast a list of certificates that have been revoked, and every potential recipient will maintain a CRL containing all such revoked certificates. When a certificate is presented to a recipient, the recipient will check to see if it has been revoked. The second solution proposed also puts the burden on the recipient of a certificate, but suggests that the recipient

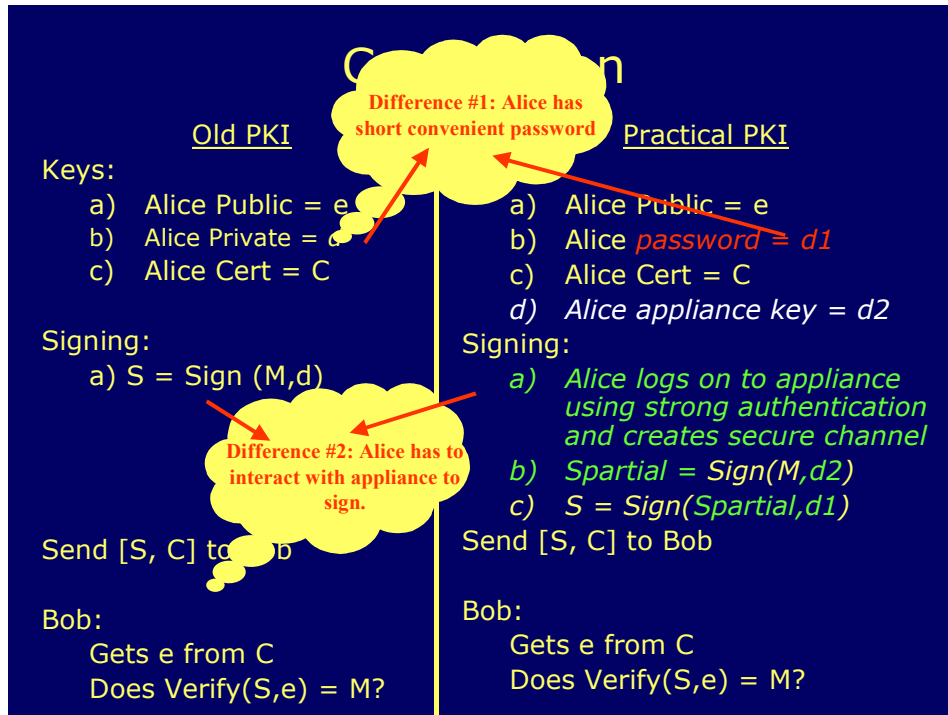
check an on-line database, which is notified every time a certificate is revoked.

Neither of these solutions provide for truly instant revocation once say theft is detected, and neither provide for a convenient mechanism to detect a theft of an ID before it is reported. Contrast this with the credit card or calling card fraud management systems where using velocity checking techniques the organization can detect a theft before the customer knows it.

The Science: The SingleSignOn.Net Solution

Colin Boyd introduced an interesting RSA variation for “digital multisignatures”. In his scheme the RSA private key d is split into multiple portions d_1, d_2, \dots, d_k , where $d_1 \times d_2 \times \dots \times d_k = d$. The i^{th} portion d_i is given to the i^{th} user. The users can then jointly sign a message. For example, if there are two users ($k = 2$), the first user computes and the second user completes the signature by computing $S \equiv S_1^{d_2} \pmod n$. The resulting signature is identical to one signed by the regular RSA private key (i.e. $S \equiv M^d \pmod n$) and can hence be verified, in one operation, using the regular public key (e, n) .

The SingleSignOn.Net system is based on a 3-Key RSA system. We begin with the usual RSA process of generating the public key e, n and the private key d . We then further split the private key d into two parts, d_1 and d_2 where $d_1 \times d_2 = d \pmod{(p-1)(q-1)}$. We give d_1 to the user and d_2 to an online security appliance. d itself is destroyed. Signing a message now happens in two steps, the appliance computes $S_1 = M^{d_2} \pmod n$, and then the user computes $S = S_1^{d_1} \pmod n$. Observe that the resulting signature S , is identical to that which would have been computed, had the user had the entire key d .



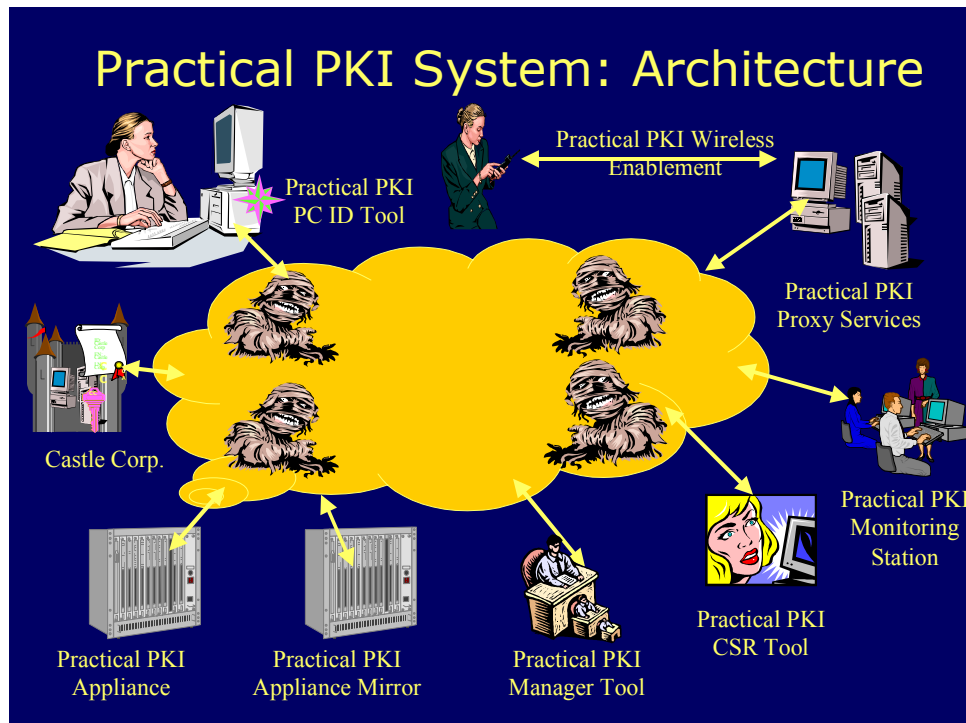
This approach has some extraordinary benefits:

- d_1 can be generated from a short password the user memorizes without compromising security.
- Since interaction with the appliance is required to sign, once a key is revoked it can never be used again -- the appliance will refuse to cooperate. Instant revocation is achieved without certificate revocation lists, or requiring the recipient to contact online servers.
- It is possible by studying the log files at the appliance to detect anomalous usage (what is known in the fraud management world as "velocity checking") so it is possible to try and detect theft of a key even *before* the user realizes it has been stolen.

Yet, as can be observed, it is only the process of creating the signature $M^d \text{ mod } n$ that changes. Everything else, the public key, the certificates, etc., all remain the same, allowing the system to take full advantage of the PKI infrastructure that exists today.

The SingleSignOn.Net Product

The architectural diagram below shows the various components. The components make the worst-case assumption that they are connecting over an insecure network.



- **The Practical PKI Appliance™**

To avoid the problem of running secure software on an insecure general-purpose operating system, we have an appliance architecture, similar to those used by popular firewall manufacturers. For redundancy a mirror replica of the appliance should also be used. The system automatically load balances across the two appliances. The appliance implements a role-based access control model, where there are a few well-defined roles, e.g. system manager and security manager, and each role can only perform the operations that it is permitted to perform. For instance the system manager can manage the number of processes etc., but cannot view customer data. In other words the principle of *least privilege* is strictly enforced.

- **The Practical PKI Manager Tool™**

This is primarily used by the security and system managers to initialize and manage the appliance. It interacts with the appliance over a virtual secure channel, and can therefore be used across an underlying network that is insecure. The tool also contains a self contained 'be your own Certificate Authority' module, in the event that you choose to be your own CA. You do not however, have to be your own CA, and can get the CA of your choice to sign the appropriate keys.

- **The Practical PKI CSR Tool™**

This is used primarily by Customer Service Representatives (CSRs) to add, revoke, and modify end user accounts. The tool also allows for the batch creation of a large number of user accounts at once. The functionality in the CSR tool is also available as an API that can be integrated into your Customer Relationship Management (CRM) systems.

- **The Practical PKI Monitoring Station™**

This tool is used to perform three functions. First it monitors the 'health' of the appliance using the industry standard SNMP management protocol. Second, it exercises the functionality of the system to ensure that everything is working and will trigger alarms should something be amiss. Finally, it is the point at which log files can be analyzed for anomalous behavior. The tool can either be run in-house, or you can utilize the monitoring service we offer.

- **The Practical PKI ID Plug-In™**

To use the system the end user downloads a simple browser plug-in. The plug-in is signed by whichever certificate authority you choose, and displays your brand. The consumer can also use the plug-in to manage their profile information and to revoke their keys.

- **The Practical PKI Proxy Services™**

All the functionality of the ID Plug-In and the CSR tool is available in the form of APIs, which can be integrated into any application of your choice. Wireless gateways, for instance, can access the service as a proxy for the consumer.

Using the SingleSignOn.Net Product

There are several different ways in which the SingleSignOn.Net Practical PKI™ Appliance can be deployed in Internet, Intranet and Wireless environments:

A better password system for your site

The simplest use of the appliance is to use it as a 'better password system'. Users enter their userIDs and passwords onto a web site as before, and the web server runs as a proxy for the user to verify the user's identity and retrieve the user's profile. There are three benefits to this approach:

- Most password systems are vulnerable to the entire password database being copied. Even if the passwords are stored encrypted they are still vulnerable to off-line, password guessing attacks ("dictionary attacks"). The Practical PKI™ system, which uses public key cryptography, does not have these weaknesses.
- Sensitive profile data, e.g. a customer's credit card number, phone number, etc., are now stored in a highly secure appliance where even the system managers do not have access to the information. Only the people or applications entrusted with retrieving the data can do so; enforcing the principle of least privilege.
- It is a simple first step to prepare for deploying IDs and certificates to end users for an existing application.

In this mode the end user is not even aware of the system.

A PKI system where you can act as your own CA

The system allows you to generate your own CA keys, and to issue 'self signed' certificates to your end users. Consider the following uses of this:

Your customers use this to access multiple web sites at your organization, with single login capability. It is highly likely that each of these web sites already has SSL built in, but only has server side SSL turned on. Simply turn on client side SSL, install your root cert on the server, and these web servers are now ready to use strong authentication.

The same process described above can also be used as a simple way to enable employees within an organization to securely access web sites on an Intranet.

All users who have IDs issued by you, can send and receive signed and encrypted email using your existing email system, if they use Outlook, Outlook Express, Netscape Messenger or Eudora as their email client. (Note

that many popular web email services such as Yahoo mail or AOL mail allow you to send and receive email to clients such as Outlook Express).

In general, being your own CA to allow users access to your own sites does not entail any more risk or special protection than your existing password based systems.

A PKI system where you use an existing CA

In this mode a Certificate Authority of your choice will sign the certificates that the Practical PKI™ appliance issues. If this CA has their root cert widely deployed on the Internet, then it is possible for your consumers to access any web site which trusts that CA (and has client side SSL turned on). In this mode the appliance is being used as what is often referred to as a "registration authority" (RA); of course it's a special RA that provides users with the convenience of passwords while retaining the security of public key cryptosystems. Further, unlike other RA solutions, it also provides the opportunity for velocity checking and instant revocation.

Wireless PKI

The Practical PKI™ proxy servers can be implemented at a wireless gateway. For 1-Way SMS phones, the wireless gateway is a trusted proxy that is acting on behalf of the consumer. If the consumer has a WAP phone which supports digital signing, then it is feasible to reduce the trust placed in the proxy, by having the signature occur on the WAP phone.

Summary

All the ingredients for improving the security of Internet and Wireless authentication and digital signatures are in place. Making the promise of PKI become real however requires a fresh approach, one that is predicated on operating in the intersection of usability, operability and security. The SingleSignOn.Net Practical PKI system is the only system designed explicitly to meet these goals.