Supporting Digital Signatures in Mobile Environments

Scott Campbell
Department of Computer Science and Systems Analysis, Miami University
scott@cc-campbell.com

Abstract

Digital signatures, like physical signatures, can verify that a specific user affixed their signature to a document and they can also verify that the document is the same as when the user affixed the digital signature. Digital signature systems (DSS) use public key cryptography methods to create digital signatures. The integrity of the digital signature is tied to the security of the user's private key. As long as the user's private key is secure, then only the user can affix their digital signature to a document. In this paper we examine methods and risks involved in creating digital signatures on workstations other than the user's primary workstation. The challenge is to allow the user to create a digital signature, which requires their private key, at workstations where we can not guarantee the key's security.

1. Introduction

A user creates a digital signature to act much as a physical signature does on a written document. A signature indicates the user's agreement with the document. With written documents, we can verify a signature and show that the user in question actually signed the document. This characteristic, non-repudiation of origin means that a user can not deny sending the message as defined by Zhou [1]. For signing documents we have it mean a person can not later claim they did not sign the document. It is also important to verify the document is currently the same as when the signature was affixed to the document. With physical documents this is done by examining the document and copies looking for changes such as the use of whiteout or other visual changes. This ensures that the terms and conditions agreed to by the user have not changed. Digital signatures and digital documents need to offer similar properties so they can be as legally binding as written signatures [2].

Physical documents run the risk of having forged signatures or of having unauthorized changes. Scanners and other tools can place realistic copies of signatures on physical documents that are difficult to detect. Copiers, printers and computers can modify documents so precisely that it is difficult to detect the changes. Typically people sign and keep multiple copies of a document so all parties to an agreement have their own physical copy. Still, it is difficult to resolve conflicts between copies if there are differences using physical documents. These problems become even more acute when the actual documents and signatures are digital files.

Current digital signature systems (DSS) can create digital signatures that allow verification of a digital signature's authenticity and at the same time allow verification of the document's integrity. Using a DSS, a person can create a digital signature for a document. Their also must be a mechanism to validate the document's digital signature and integrity. To provide these properties, the digital signature is unique to a specific instance of a document. The digital signature is also unique for each user. Several systems exist that support digital signatures, see for example Gradkell [3] or Adobe [4].

Digital signature systems use public key cryptography. A digital signature is simply a numeric string or value that is unique to the document and has the property that only the signatory can create the appropriate signature. Every user has two special and related values, the public key and the private key, that the DSS uses for creating and verifying a digital signature. The DSS creates the digital signature with the user's private key and with methods such that it is unique to a given document. The signing process, shown in figure 1, begins by taking a hash of the document. This simply reduces the size of the signature and protects the document's privacy. The DSS then encrypts this hash using the user's private key. This encrypted hash, appropriately structured, is the digital signature. The user can append the signature to the document or store the signature separately from the document. The DSS also validates a signature. Validation can be done by anyone as it uses the signer's public key which is by definition available to anyone. The DSS validates a signature by first creating a hash of the document. The DSS next decrypts the signature using the user's public key. The signature is valid if the document's hash matches the decrypted signature's hash. This can only occur when the document is the same as when the user created the digital signature and when the user is the one who created the digital signature. Only the user could create a valid signature since only the user has access to their private key [5].

The security and integrity of a user's digital signature depends upon the keeping their private key secure. Compromising a private key allows anyone to create that

Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03)
1080-1383/03 $17.00 © 2003 IEEE

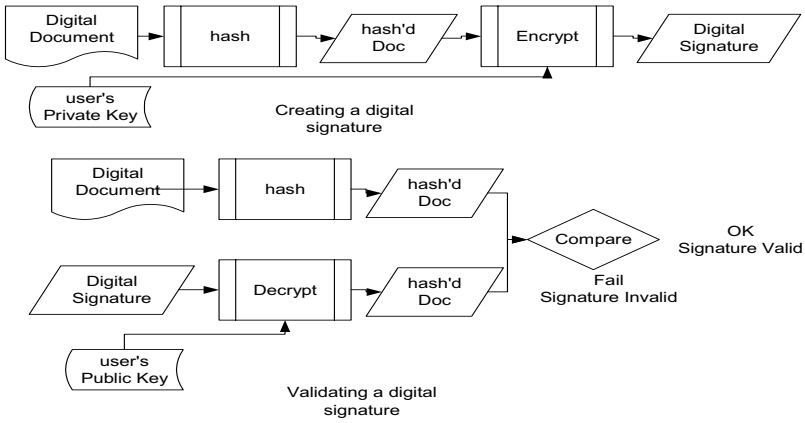

Figure 1. Creating and Validating a Digital Signature

user's digital signature. The DSS needs the private key to create the signature and at the same time must keep the key secure. Typical protection mechanisms protect this key with encryption, file security and/or storing the key on specialized devices. It is possible to use encryption to encrypt the key. The user then supplies this second key which allows the DSS to create a signature. Typically the DSS uses a smaller key for this encryption, one the user is able to remember and enter at the workstation. File security protects the private key with access control mechanisms or by storing the key on a separate floppy disk. An improvement upon storing the private key on a floppy is to use a hardware device. The hardware device securely stores the key and the DSS then retrieves the private key from the hardware device after appropriate user authorization [6].

The problems of protecting the key expand when we wish to sign documents away from our primary workstation. Adding mobility requires that we protect the key in environments that we do not entirely control or trust. It is not possible to know what software or hardware methods are on these other computers that can compromise the privacy of our private key. We define mobile signing as signing a document at a workstation other than our primary workstation. This typically requires transferring our private key to this workstation. It is possible to transfer the key using an encrypted channel or via a secure device like a smart card [47]. Asokan, et. all propose a method for using servers to support signature signing [8]. Their system sends uses a hash-chain for creating unique signatures that minimize the key's exposure. However their solution does not take into account malicious hosts. Malicious hosts can send the wrong document to the server for signing and their server will then sign this bogus document. Sanders [9] discusses some methods of overcoming malicious host attacks including using undetachable signatures. These signatures do not appear to be directly compatible with existing DSS systems. Our goal is to provide a mechanism for using signatures servers that avoid or minimize risks from malicious hosts.

2. Compromising Private Keys

The problem is that we can not guarantee protection of the private key if it is copied to the remote workstation. Creating a digital signature means the DSS must have access to the private key. However that workstation might have software or hardware designed to steal or copy the private key. Once the private key is copied, then anyone can create a digital signature in the user's name. Hence we need methods to absolutely protect the integrity of the key but in such a fashion that we can still sign documents while at remote workstations.

To protect the private key, some DSS use specialized hardware devices to perform the signing process. The document is sent to the hardware device and the signature created on that device. The private key is stored on this hardware device and never transferred to the workstation. As long as the private key can not be transferred from the hardware device, then the key remains secure. This approach is mobile as the hardware device can be used on any workstation. These devices utilize a USB interface so the hardware interface is no longer a significant problem. However the user must still install and use software on the workstation to communicate with the signing device. The user invokes the signing software and the signing software actually transfers the authentication and document to the signing device.

There are several methods to compromise such a system. The first is to simply steal the device. Using strong authentication methods can minimize this problem. However it is still possible to compromise the authentication as the limitation of passwords for authentication is well known. It is possible to retrieve the private key from the device by hacking the hardware. Attacks on these devices are well beyond our current scope of investigation but they do exist. Another attack is

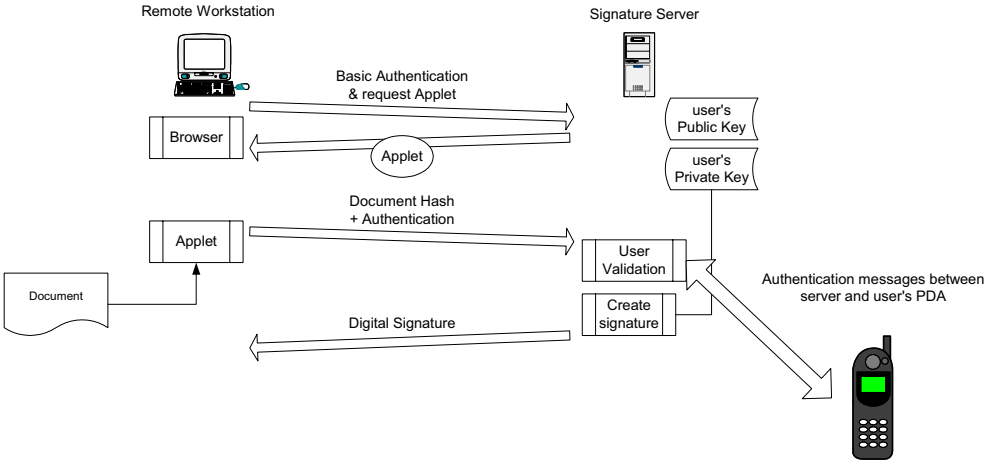

Figure 2. Signing Process with PDA Authentication

to use timing to trick the device into signing a "hack" document instead of the user's document. We assume that the signer has little to no control over the remote machine they are using. The attacking software gets the user to authenticate and request the signing of their document but instead sends the "hack" document to the device. Thus the device will create a digital signature for the "hack" document.

3. Personal Signature Server

To keep the private key secure and at the same time allow the user to sign documents we create a server, the Personal Signature Server (PSS) that will sign documents. The server runs on the user's primary workstation and never sends the private key to any other workstation. The PSS holds both the user's public and private key and performs the actual encryption of the document hash to create the digital signature. The process begins with the user contacting the PSS using a standard browser over a SSL encrypted TCP connection. The server validates the user and then sends a java applet to the remote workstation. Using the applet, the user selects the document to be signed. The applet then computes the document's hash and sends this hash to the PSS. The PSS then computes the digital signature using the user's private key, stores a copy of the signature in its archive and finally returns the signature to the user at the remote workstation. This approach maintains the private key's integrity since the PSS uses but does not transfer the private key. At no time is the key purposefully sent from the signature server.

Verification is done by a separate applet and does not require involvement of the PSS. The PSS will return the user's public key but if the verifier already has the public key the PSS does not need to be involved. Since verification uses only the public portion of the key there are not security issues involved with key management. The verifier interacts with the verification applet and selects both the document and the digital signature for validation. The applet then performs the verification process. Alternatively, the PSS provides a browser based interface to verify documents but this requires sending the entire document to the PSS. It is necessary to ensure that the verification messages sent to the user actually originate from the applet or the PSS. We do not want malicious programs to intercept a verification failed message and display a bogus success message.

Key security is the major benefit of using the PSS but use of the PSS also simplifies key management issues. Management is a major issue in dealing with keys and security in general. Revoking a key, keeping track of what documents have been signed, and keeping older keys available for verifying signatures are all important issues. Using the PSS simplifies these tasks. Revoking an existing key and creating a new key is done at the PSS's console and changes take effect immediately. The signature server will continue to store older public keys to verify previously signed documents.

There are several vulnerable points with the PSS. It is possible to trick the PSS into signing the wrong document much like the timing attack described above. Tricking the

PSS into signing the wrong document arises from the fact that a malicious program on the remote workstation can arrange to inject its own document hash into the signature process after the proper user authentication. To combat this problem the PSS uses additional encryption for all traffic with the PSS. This is in addition to the SSL encryption. The java applet encrypts all its messages to the signature server using a shared key coded into the applet. Another program will not be able to inject its own messages to the signature server unless it can obtain this key. Obtaining the private key is possible by disassembling the java applet code but is not a straightforward process. As a second level of protection the PSS generates a fingerprint of the hash it is signing. The PSS sends this fingerprint to the signing applet which displays its version of the fingerprint and the PSS's version of the fingerprint for the user. The user compares these two fingerprints and thus verifies that the PSS is signing the correct document.

The PSS must have strong user authentication. Anyone who passes the authentication steps can create a digital signature for that user. Poor authentication destroys the PSS's integrity. We attempt to improve the strength of the authentication by adding an out of band authentication step to the signature server process. The PSS sends an authentication request to the user via the user's PDA and the PDA's wireless Internet connection as shown in figure 2. The PSS requires that all signatures be approved by the user via the PDA device. The PDA periodically contacts the signature server to let the PSS know of its location.When the PSS receives a signature request, the PSS sends the PDA an encrypted message asking the user to validate the signature request. This message contains both a unique token as well as the document fingerprint. The user can compare the data from the PDA with the actual document's data and ensure the correct document is being signed. The user validates the signature request and the PSS completes the signature process.

We are currently investigating vulnerabilities in this current implementation. We also are looking at other means of providing similar authentication without having access to a wireless internet. We are looking at how to use cell-phones and their internet browsing capability to validate signing requests. Additionally one could use the Short Messaging System (SMS) available on many phones for validating the signing request. The final methods we are looking at is to use pagers to receive information from the PSS. In these instances the user could receive a validation code from the PSS and send their validation back to the PSS via the java applet. We have not fully explored these methods or their associated risks.

4. Integrating the PSS and Hardware Signing Devices

In the introduction we discussed devices that securely store the private key and perform the signature process on the device. We view that the major liability with these devices is their susceptibility to theft. We are currently implementing a system that combines the PSS and a mobile signature device using an Ibutton hardware device. The Dallas I-Button is a small, inexpensive device capable of running java applets on the device and connects to a workstation using a USB interface. The Ibutton supports physical tamper security measures that make it difficult to improperly access data on the Ibutton to extract the private key. We are working on implementing a system where the java applet on the Ibutton performs the actual signing process. The workstation applet computes the document hash and then sends the hash to the Ibutton. However prior to signing the document, the Ibutton communicates, through the applet on the remote workstation, with the PSS to obtain authorization for the signing. Using the PSS allows us to revoke keys in the case of a stolen Ibutton, keep a running log of signed documents, and to update the keys used by the Ibutton to sign documents.

The major advantage of using the Ibutton over the PSS is flexibility. We define a set of rules that control how often the Ibutton must contact the server before creating a signature. The user can tailor these rules to meet their acceptable level of risk. Currently we support rules that require the Ibutton to contact the PSS based upon length of time since last authorization and number of signatures since last authorization. We hope to increase the rules type and flexibility. For example, if the user is working with common types of documents, like expense statements, then we envision rules where the Ibutton can sign reports with totals under set limits, say $1000, and but must obtain PSS authentication for larger totals. The use of these rules allows the Ibutton to create signatures in situations where connectivity is not available but still maintain some benefits of using the PSS.

5. Conclusion

Digital Signature Systems allow users to create a digital signature, a numeric string, that is unique to a specific document and can only have been created by the user. Through the use of public key cryptography, the DSS uses the user's private key to create the digital signature. The resulting signature has key properties of non-repudiation and verifying the documents integrity since signing. In order to provide for valid digital signatures it is imperative to keep a user's private key secure. Our current research looks at methods of allowing

users to sign documents at workstations where we can not guarantee the security of the private key. Instead we use a Personal Signature Server to perform the signing process. The PSS stores the private key in the PSS and never sends the key to the remote workstation. Authentication and validation of the signature request becomes a major task in implementing the PSS. We need to ensure that the document the PSS is signing is the document the user desires to sign and not a document substituted by rogue software running on the remote workstation. The PSS must verify the user's identity prior to signing any documents. Our current version of the PSS uses a separate authentication step to the user's PDA to authenticate both the request and verify the integrity of the document.

References

1 J. Zhou and D. Gollmann "A Fair Non-repudiation Protocol," *IEEE* Symposium on Research in Security and Privacy

2 K. Gerwig, "Will the Digital Signature Transform E-Commerce?" Business: the 8th layer, Volume 4, Issue 3, (September 2000) ACM netWorker.

3 http://www.gradkell.com/pks.htm

4 http://www.adobe.com/

5 W. Stallings "Network Security Essentials: Application and Standards," 2000.

6 P. Blomgren "E-signatures with USB crypto-tokens" [http://www.itworld.com/Sec/4039/NWW_2-12-01_signatures], February 12, 2001.

7 B. Gelbord "Viewpoint: signing your 011001010," *Communications of the ACM* vol. 43, no. 12, pp. 27-28, 2000.

8 N. Asokan, G. Tsudik and M. Waidner "Server-Supported Signatures," *ESORICS* pp. 131-143, 1996.

9 T. Sander and C.F. Tschudin "Protecting Mobile Agents Against Malicious Hosts," *Lecture Notes in Computer Science* vol. 1419, pp. 44-49, 1998.