Developer
Technical
Note

Digital Signatures in the PDF Language

CONTENTS

Introduction

Digital signatures can be used for many types of documents where traditional pen-and-ink signatures have been used in the past. However, the mere existence of a digital signature is not adequate assurance that a document is what it appears to be. For a recipient to fully trust an electronic document, they must be able to verify that:

- the document has not been altered
- the document came from someone they trust

Digital signatures in PDF documents address these needs by providing a way to authenticate digital data based on *public key cryptography*.

This document describes how digital signatures are represented in a PDF document. It explains the digital signature security features supported by PDF, and how they solve the need for trusted documents and signatures. This document does not attempt to explain how the documents are created, interpreted, or dynamically validated.

The discussion is based on the security features defined in the public Adobe PDF Language specification, *PDF Reference – Fifth Edition, version 1.6*, the format used by Adobe Acrobat 7.

Background

While many people are familiar with Public/Private Key (PPK) or Public Key Infrastructure (PKI)-based digital signatures, there are other ways to implement digital signatures. PDF does not limit the type of digital signatures that can be used; third-party developers can define their own signature mechanisms in the form of an Acrobat plug-in signature handler.

The PDF language specifies that each signature in a PDF file is associated with a signature handler that is to be used to process that signature. The signature is placed in the PDF file in a *signature dictionary* (See Table 8.98 in the PDF Reference, version 1.6) which contains the name of the signature handler in the value associated with the **/Filter** key.

Although Acrobat supports third-party signature handlers, Acrobat comes with built-in support for only PPK/PKI-based digital signatures. Because PPK/PKI-based digital signatures are the most common, this document focuses on how such signatures work in the context of the PDF language specification. Many of the facilities described are available using other digital signature technologies, but which facilities and how they are used is dependent on the third-party signature handler.

Digital Signatures Based on Public Key Cryptography

The signature handler built into Adobe Acrobat is based on *public key cryptography*, or *Public/Private Key (PPK) cryptography*. PPK is based on the idea that a value encrypted with a private key can only be decrypted using the public key.

NOTE: The reverse may also be true and is useful in encrypting documents for specific recipients, but that concept is outside the scope of this document.

The usual way in which PPK is used in digital signature applications is as follows:

1. A document to be signed is turned into a stream of bytes.
2. A one-way *hash* of the bytes is generated using a well known algorithm, such as SHA-1.
3. The hash value is encrypted by the signer using their private key, usually using a signature algorithm like RSA.
4. The encrypted hash value (the mathematical "signature" of the document) is attached to the document.

A recipient of the document must then validate the signature to confirm the sender signed it, and that the document has not been modified since it was signed. That requires the following steps:

1. The recipient's application generates a one-way hash of the document using the same algorithm the signer used, excluding the *signature value*.
2. The encrypted hash value in the document is decrypted using the signer's public key.
3. The decrypted hash value is compared to the locally generated hash value.
4. If they are the same, the signature is valid. If not, the signature is not valid.

An invalid signature indicates either that the document has been modified between signing and verification (see "Documents Modified After Being Signed" on page 7), or that the public key used by the recipient does not correspond to the private key used by the signer.

Verifying Who Signed the Document

The use of public key cryptography solves the problem of whether the document has been altered, but additional steps are required to determine who actually signed the document. The easiest solution is for the signer to include their public key along with the document, which ensures that the recipient will have it for verification.

However, the private key and the public key are merely numbers, and anyone can generate a public and private key pair using any number of tools. Thus, there is a need for an independent authority that issues, records and tracks the numerous requests for public/private key pairs.

To associate a particular person with a particular public key, a certificate is used, formatted using the rules specified in the ITU-T X.509 v3 standard. A certificate combines a public key and an identity, as shown in Figure 1. To prevent someone from tampering with the certificate (for example, removing their name and putting yours in its place), the certificate is itself signed using the same process described above.

Typically, a user obtains a *digital ID* which contains their certificate and private key. The digital ID may be stored on a user's system, or on a hardware device such as a smart card or a token. When they sign a document, the certificate portion of their digital ID is embedded in the signature data stored in the document.

However, certificates alone cannot provide complete trust as to who signed the document. For example, Acrobat provides a mechanism to generate a *self-signed* certificate which binds a simple user-provided identity to a public key generated by the application; it is then signed using the corresponding private key. Obviously, there is nothing to prevent someone from generating a self-signed certificate with someone else's name on it. Hence, an unknown self-signed certificate does not have a very high level of assurance. To solve this type of trust problem, a public key infrastructure (PKI) can be used.

Public Key Infrastructure

When signing an important paper document, a person usually signs it in front of a notary public or other trusted authority after providing them satisfactory evidence of their identity. Because the notary is considered to be trustworthy, you can trust the signature the notary witnesses. Using a PKI is a method of providing a similar kind of trust.

The main component of PKI services is a Certificate Authority (CA) company or organization that has its own private and public key. The CA signs the certificates of users, handles the process by which the user's identity is authenticated, and may manage other services such as checking for revoked certificates, secure time-stamping, and support for nonrepudiation (so the signer cannot later deny having signed a document). Some of the well known PKI providers who issue X509v3 certificates to consumers are Verisign, Geotrust and Cybertrust.

PDF Digital Signature Basics

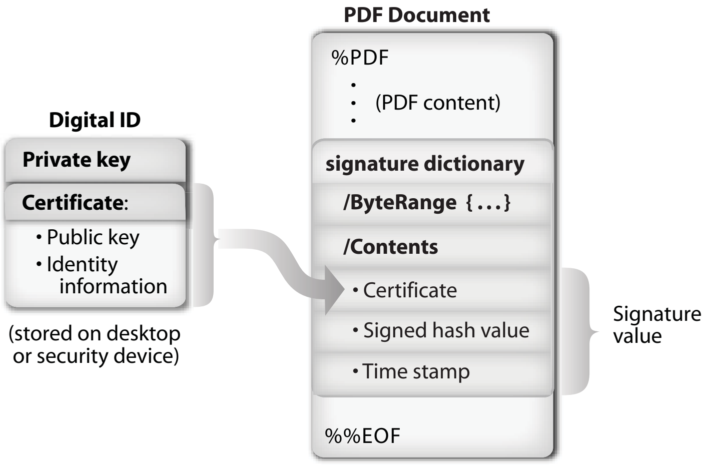
PDF is unique in that it includes support for signatures to be embedded in the document itself, rather than managed as separate data or added on to an existing document format. This means that the viewing application can perform certain types of modification without invalidating the signature. With other digital signature formats, the user may need either two applications to handle both the document and the signature, or would need to manage two separate files for each document.

PDF Signature Data

In PDF, the user's certificate (see "Verifying Who Signed the Document" on page 2) is part of the data included in the PDF file when a document is signed.

Figure 1 shows the relationship between the digital ID, the user's certificate (both are stored on the user's hardware device), and the signature value that is embedded in the PDF document.

FIGURE 1 Digital ID and a Signed PDF Document



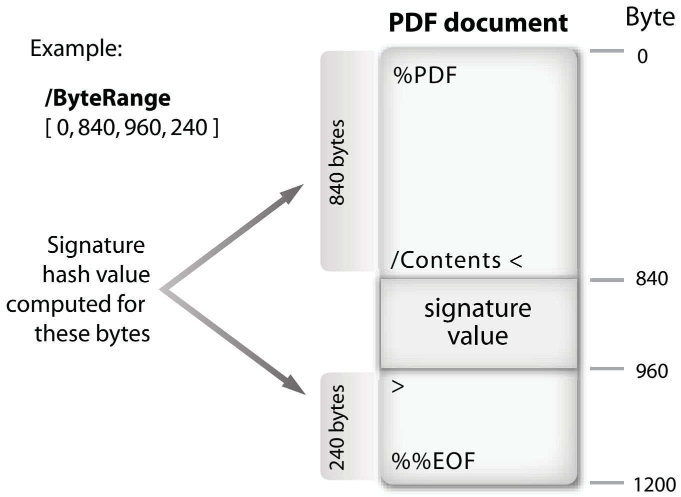
The signature value may also include additional information such as a signature graphic, a *time stamp* from a trusted server, and other data that may be specific to the user, system, or application.

Locating the Signature in PDF: the /Contents and /ByteRange

The signature value generated by signing a PDF file is included in the body of the PDF file. Applications can locate the signature value by using an array of four numbers called the *ByteRange*, which is stored alongside the signature value.

The four numbers are actually two pairs of numbers. The first number in each pair is the offset in the file (from the beginning, starting from 0) of the beginning of a stream of bytes to be included in the hash. The second number is the length of that stream. The two pairs define two sequences of bytes that define what is to be hashed. In between the end of the first sequence and the beginning of the second one is the location for the value of the **/Contents** key, which contains a hex-encoded PKCS#7 object that is the actual signature value. Figure 2 shows an example of a ByteRange value that defines the bytes to be used in the hash calculation.

FIGURE 2 The ByteRange and Signature Value



In this example, the hash is calculated for bytes 0 through 839, and 960 through 1200.

The Save/Hash/Update Process

Acrobat always computes the hash for a document signature over the entire PDF file, starting from byte 0 and ending with the last byte in the physical file, but excluding the signature value bytes.

The signature is placed in a PDF file as follows:

1. The entire PDF file is written to disk, with a suitably-sized space left for the signature value, and with worst-case values in the ByteRange array.
2. Once the location of the signature value is known in terms of offsets in the file, the ByteRange array is overwritten using the correct values. Because the byte offsets must not change, extra bytes following the new array statement are overwritten with spaces.
3. The hash of the entire file is computed, using the bytes specified by the real ByteRange value using a hash algorithm such as SHA-1.
4. The hash value is encrypted with the signer's private key and a PKCS#7 signature object is generated.
5. The signature object is placed in the file on disk, overwriting the placeholder **/Contents** value. Any space not used for the signature object is overwritten with spaces.
6. The PDF file is then re-loaded in Acrobat to ensure that the in-memory version matches the on-disk version.

Multiple Signatures

Some documents may require more than one signature. It is easy to handle that with a paper document by just drawing another line on the paper. With PDF, it is almost as easy; just add another signature field on the document.

In the paper world, a person signing a document would be wise to save a copy of the document as it was signed. Then if someone else comes along and changes the document, the signer can argue that the document had been altered.

With PDF documents, any attempt to alter the document by modifying the bytes of the file will cause the digital signature to be invalid. That is because the hash value calculated at verification time will not match the encrypted hash created at signing time. So how does one add another signature to an existing document without breaking an earlier signature? PDF supports the ability to do incremental updates, which provides support for multiple signatures as well as roll-back (see "Roll-back" on page 7), which is the ability to view the document exactly as it existed when each approval signature was applied.

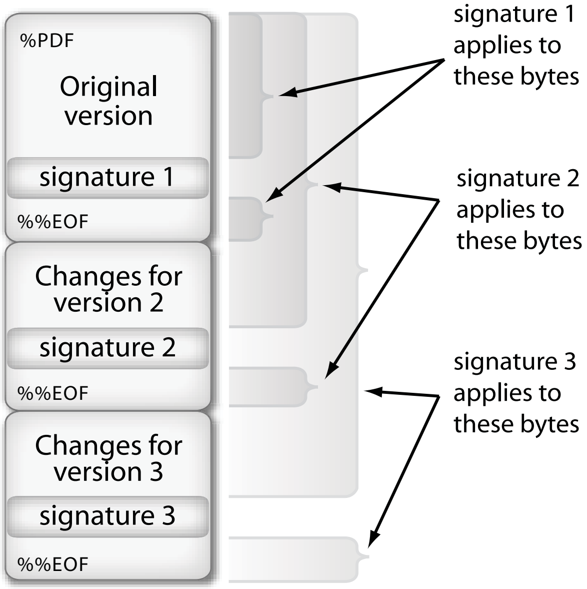
The PDF Incremental Update Facility

The PDF file format defines an incremental update capability. Incremental updates are transparent to the person viewing the document, but allow for the detection and audit of modifications to the file. This feature of the PDF language allows any PDF file (but most importantly for our discussion, PDF files with signatures) to be modified by adding the modification information to the end of the file in an incremental update section. No changes whatsoever are required to the bytes representing the earlier version of the file. This allows additional signatures to be added to a PDF file without modifying any data covered by an earlier signature.

NOTE: The effect of additional signatures on document validity is discussed in "How Field Locking is Done" on page 9.

Each additional signature will cover the entire PDF file, from byte 0 to the last byte, excluding only the signature value for the current signature value. Figure 3 shows how signatures are created for a file with three signatures.

FIGURE 3 Multiple Signatures and Incremental Updates



The PDF incremental update facility allows anyone, signer or not, to change just about anything in the document, including adding pages or modifying text. (See "The Need for Certified Documents" on page 8 for one way in which the PDF language addresses this problem.)

NOTE: In earlier versions of Acrobat, the Save As command would eliminate objects in the PDF file that were no longer needed, and would result in the saving of the entire file as a simple PDF file with no incremental update sections. This would break any existing signatures. Therefore, more recent versions of Acrobat do not optimize away the incremental update sections if a signature is present in the file. There are two solutions: 1) if using Acrobat 5, use only the Save command, not the Save As command; or 2) make sure recipients are using Acrobat 6 or higher.

Roll-back

The incremental update facility of the PDF file format allows PDF viewers like Acrobat to effectively retain all signed revisions of any PDF file. This makes it possible for users to actually see the version of the PDF file that was signed. In Acrobat, you can right-click on a signature and select the View Signed Version command. This will display the document as it existed at the time that signature was applied. This is referred to as the *roll-back* facility of Acrobat. It can be mimicked manually by removing any bytes in the PDF file after the EOF corresponding to the signature.

You can also compare the current version of any document to the signed version using the Compare Signed Version to Current Version command, also available by right clicking on a signed signature field.

Both the Compare Signed Version to Current Version command and the View Signed Version command are also available using the Signatures tab and by choosing the Document > Digital Signatures menu item.

Documents Modified After Being Signed

A document with one signature on it that covers the entire document is relatively simple to validate. Either the hash matches or it does not match. (We will cover PKI-related validation issues later.) If the hash does not match, Acrobat will mark the signature with a red X. If the hash does match, Acrobat will show a green check mark with the signature.

But what if the document was modified using PDF's Incremental Update facility after it was signed? What should Acrobat display? Is the signature valid or invalid?

The signed portion of the PDF has not been modified, and by using the roll-back feature, it is possible to see what the signature covers. But the document may not appear on screen as the signer saw it. So Acrobat displays a green checkmark with a yellow triangle icon. That does not mean that the signature is invalid – it is still valid for the part of the range of bytes to which that signature applies.

The purpose of the warning is to alert the recipient that the document was modified after the signature was applied, but that modification may well be only a second signature that was intended by the originator, and which is fully valid. Adobe recommends that the recipient use the View Signed Version command or the Compare Signed Version to Current Version command to see what was signed or what has changed since signing.

Naturally, a second signature on a document is done using the incremental update facility. So a first signature will be flagged with the yellow triangle once a second signature has been applied. The minimum difference between the signed version and the current version is, of course, the application of the second signature.

NOTE: This problem of the Yellow Triangle alert can be avoided if the author certifies the document, as explained below in "The Need for Certified Documents" on page 8.

The Need for Certified Documents

Unsigned PDF documents can be changed in ways that would make it difficult for even a skilled user to detact. How can the signer of a PDF document be sure that the document is what the originator intended to be signed? In a workflow where a document originates with one individual or organization and is sent to another for a signature, and is then returned to the originator, how can the originator know whether the document was modified before the recipient signed it?

Certified documents solve both of these problems and allow document recipients to know if any changes have been made contrary to the author's intent.

Another issue that certified documents address is the question of document trustworthiness. As Acrobat and other PDF applications have become more powerful, it has become necessary to limit access to some functions based on the notion of *privilege*. By having the author sign a document, it is possible for a recipient to know who the author is and assign a trust level to that author.

How Certified Documents Work

The signatures being discussed to this point are known as *approval signatures*, where someone signs a document to show consent, approval, or acceptance. A certified document is one that has a certification signature applied by the originator when the document is ready for use. The originator specifies what changes are allowed; choosing one of three levels of modification permitted:

- no changes
- form fill-in only
- form fill-in and commenting

Certification signatures are almost identical to approval signatures; for example, a hash value is computed for the entire file, and they are inserted into the file as described above. The differences include:

- a certification signature is always the first signature applied to a document
- Rules are added to the document that indicate what changes are allowed to be made before the author's signature becomes invald.
- An entry is made into a root dictionary that indicates that the signature is a certification signature.

When a certified document is opened, three operations are performed:

- The certification signature is validated against the bytes of the file. This operation includes testing the validity of the certification certificate.
- The document as it existed at the time it was certified is is opened and compared to the document in memory which is currently being viewed, including all incremental changes. A modification analysis is done and the recipient is notified if there have been modifications that were prohibited by the author.
- The certification certificate is compared to the user's list of trusted identities to establish the level of privilege the document will have.

NOTE: If the certification signature specified that no changes are allowed (that is, no approval signatures or comments), the modification analysis does not need to be performed. It is only necessary to check if any incremental changes have occurred since the document signature was applied or if any changes have been made in memory. If any changes were made, the document was modified and the certification signature is invalid.

Modification Analysis

To detect if any impermissible changes were made, Acrobat analyzes the differences between the author-signed version of the document and the current version of the document that is in memory. Any impermissible changes will cause the associated signature to become "invalid".

NOTE: Acrobat 6 (PDF 1.5) created object hashes and verified them by comparing the embedded value with the value computed when the document was opened. Acrobat 7 still creates the object hashes for backward compatibility purposes, but does not use them for verification.

Document Trust

With any PKI-based document signature, some sort of identity information about the signer is provided by the signer's certificate, included as part of the signature. This allows recipients to trust documents differently, based on who signed it. Users (or, more likely, administrators) can establish trust settings for particular end user certificates, or all certificates issued by a particular *intermediate* or *root certificate* authority. Documents without a certification signature, or signed with unknown or untrusted certificates, should be treated with great caution.

Locking Input Fields with Signatures

Many documents that require signatures are forms. Some forms may have multiple signatures fields, with different signers providing data in certain other form fields. In such cases, it is useful to lock the form fields associated with a particular signature field once the signature has been applied. This saves having to look at the version of the document to which the signature applies, to see if the value of a field was changed between that signature and the current version.

How Field Locking is Done

When a signature field is created, the form author can designate which form fields should be locked when the field is signed.

When a signature field with field locking specified is signed, a normal document signature is produced.

When validating the signature, in addition to the normal signature validation over the bytes of the PDF file, the viewing application will compare the values of the fields locked by the signature with the current values of those same fields. This allows the application to detect attempted, but prohibited, changes.

NOTE: In Acrobat 6, the comparison used object hashes computed at signing time and at validation time to determine if a locked field had changed. Acrobat 7 produces these objects hashes for backward compatibility, but does not use them when doing the comparison.

Signature Seed Values

Sometimes the author of a form wishes to limit the choices a user can make when signing a particular signature field. This can be done using a signature field seed value. A seed value specifies an attribute and a value for that attribute. The author can make the seed value a preference or a requirement. Attributes that can be specified with signature field seed values include:

- a filter (the internal name of a signature handler)

- a minimum filter version
- a sub-filter (the internal name of a type of signature, such as *adbe.pkcs7.detached*, intended to be verifiable by signature handlers other than the one that created it)
- reasons (a string indicating why the signer is signing the document)
- certificate attributes

Certificate seed values are the most common kind of seed values. They allow the author of a form to restrict signing to particular certificates, or certificates issued by particular certificate authorities, or certificates that have particular certificate policy OIDs. Again, these can be preferences or requirements. If a certificate cannot be found that matches a required certificate seed value, a URL can be provided by the form author to allow the signer to get more information, such as how to obtain an appropriate certificate.

Resources

For more information on digital signatures in PDF, see:

http://partners.adobe.com/public/developer/acrobat/devcenter.html

Terms Used in This Document

approval signature A signature used to indicate approval of the document terms.

ByteRange An array of pairs of integers (starting byte offset, length in bytes) describing the exact byte range for the digital signature hash calculation (also called digest calculation).

certificate Consists of a public key and its associated user identity. Typically stored on a user’s system or on a security hardware device. When a document is signed, the certificate is embedded in the signature dictionary as part of the signature value associated with the **/Contents** key.

certification signature A Digital Signature applied using an Individual Digital ID or Organization Digital ID for the purpose of establishing the authenticity of a document and the integrity of a document's content, including its appearance and business logic.

certified document A document to which a certification signature has been applied.

digital ID An electronic representation of certain information, based on the ITU-T X.509 v3 standard, associated with a person or entity that is often stored in a password-protected file on a computer, USB token, smart card, or other security hardware device and is used to apply a Digital Signature. A Digital ID is sometimes referred to as a signing certificate, an identity certificate, an e-mail certificate, a credential or a private key.

digital signature An electronic signature that can be used to ensure the integrity of a document and to authenticate the identity of the signor through the use of Public Key Infrastructure (PKI) technology. A digital signature is created through the use of a Digital ID.

hash A mathematical operation performed on a sequence of bytes that results in a virtually unique numeric value (or fingerprint). The hash is also often called a *digest*.

private key In Public/Private Key (PPK) cryptography, documents or signatures are generally encrypted using the private key, while the public key is distributed to allow a recipient to decrypt.

public key The part of the public/private key pair that is generally distributed, and which can be used to decrypt an item encrypted using the private key.

signature value The data inserted into the location pointed to by the **/ByteRange** value. It contains the encrypted hash value, the author's certificate (Identity information and pubic key); and potentially other items such as a time stamp object from a trusted time stamp server.

time stamp A hash of the signed contents in the signature dictionary that is signed by a trusted third party. The unencrypted hash is sent to a time stamp server via http, it is encrypted (signed) with the time stamp server's private key (digital ID) and then the signed hash along with the time stamp server's public-key is returned to Acrobat via http where it is added to the unsigned portion of the signature dictionary.

Adobe Systems Incorporated
345 Park Avenue, San Jose, CA 95110-2704 USA
http://www.adobe.com

28 March 2006