Digital Signatures in a PDF

This document describes how digital signatures are represented in a PDF document and what signature-related features the PDF language supports. Adobe® Reader® and Acrobat® have implemented all of PDF's features and therefore provide comprehensive support for the authentication of digital data based on public key infrastructure (PKI) technologies. Third-party developers can define their own mechanisms in the form of an Acrobat plug-in signature handler.

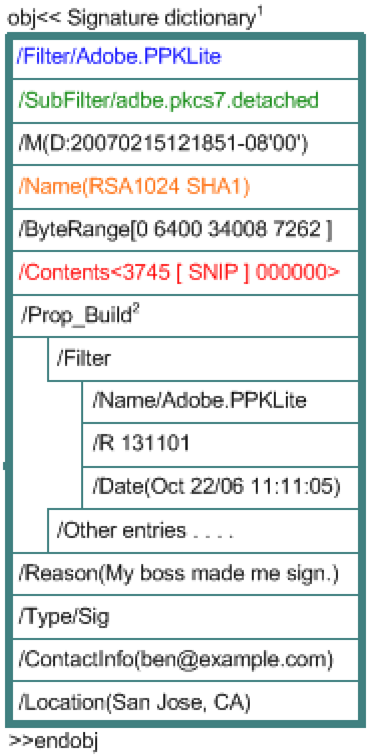
Digital signatures can be used for many types of documents where traditional pen-and-ink signatures were used in the past. However, the mere existence of a digital signature is not adequate assurance that a document is what it appears to be. Moreover, government and enterprise settings often need to impose additional constraints on their signature workflows, such as restricting user choices and document behavior during and after signing.

For these reasons, the PDF language provides mechanisms for two broad categories of tasks:

- Fully trusting an electronic document by enabling verification that the signed document has not been altered and that it was signed by someone the recipient trusts.
- Creating and controlling feature-rich and secure digital signature workflows.

Irrespective of the PDF viewing application, the PDF language supports the following:

- Standards support
- Support for alternate signature methodologies
- Support for two signature types
- Signature interoperability
- Robust algorithm support
- Multiple signatures
- Incremental updates
- Viewing previously signed document versions
- Comparing current and signed document versions
- Locking form fields
- Controlling post-signing changes
- Legal content attestations
- Enabling features via document-based permissions
- Rich certificate processing
- Controlling signature workflows via seed values



```
obj<< Signature dictionary[1]
/Filter/Adobe.PPKLite
/SubFilter/adbe.pkcs7.detached
/M(D:20070215121851-08'00')
/Name(RSA1024 SHA1)
/ByteRange[0 6400 34008 7262 ]
/Contents<3745 [ SNIP ] 000000>
/Prop_Build[2]
    /Filter
        /Name/Adobe.PPKLite
        /R 131101
        /Date(Oct 22/06 11:11:05)
    /Other entries . . . .
/Reason(My boss made me sign.)
/Type/Sig
/ContactInfo(ben@example.com)
/Location(San Jose, CA)
>>endobj
```

6.1 Representing a signature in a PDF file

In a PDF, signature information is contained in a signature dictionary. Objects in the dictionary are defined by the PDF Reference. The signature dictionary can reference, or be referenced by, other dictionaries, and it usually is (Figure 1). The entries in these dictionaries determine the nature and features of the signature, and by extension, what data can be available to any PDF viewer designed to process the signature data.

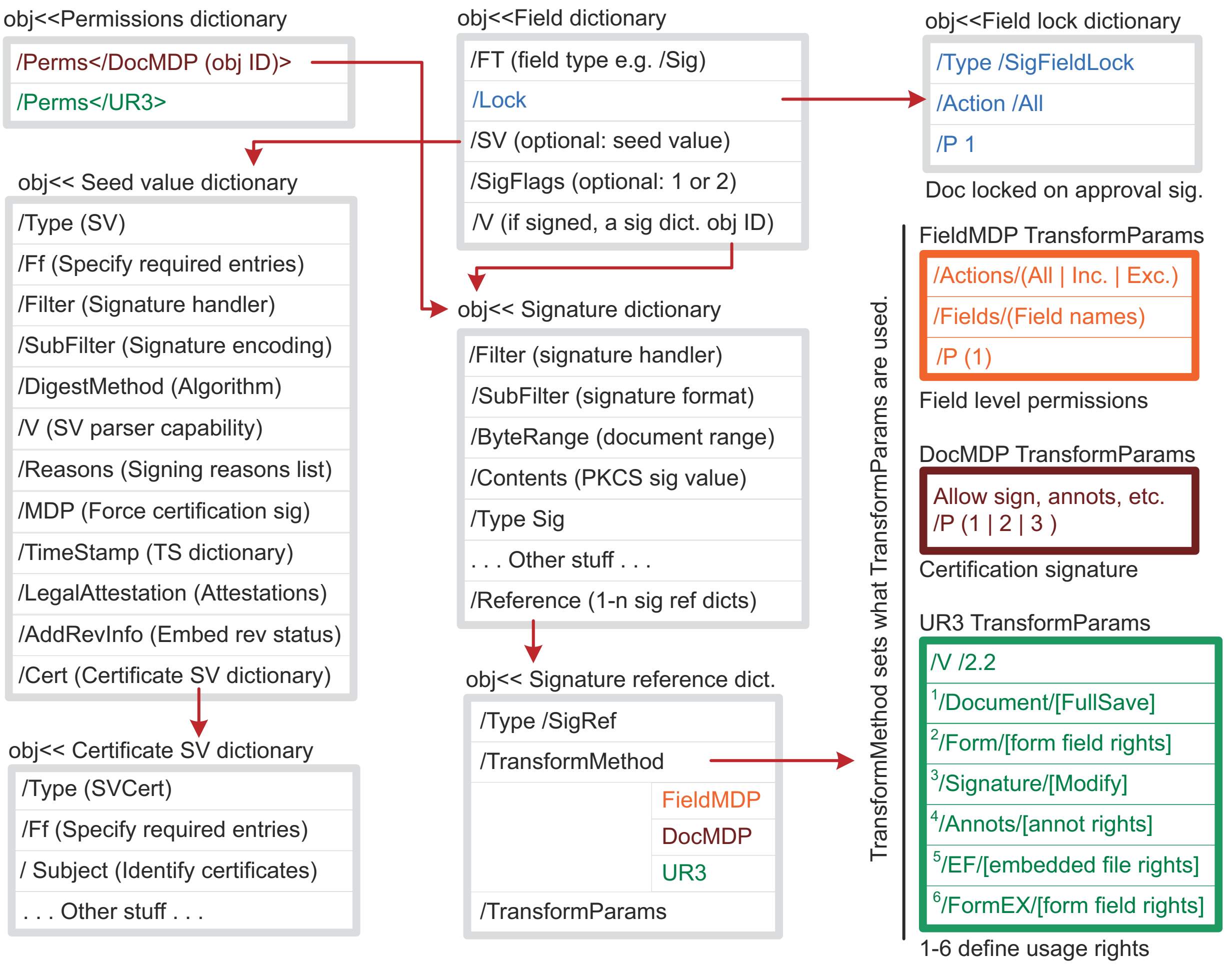
While other viewers may vary in their support of PDF language features, The Acrobat family of products supports all of those features. At a high level, these features can be grouped into these categories:

- Adding a digital signature to a document.
- Checking that signature for validity.
- Permissions and restrictions that control the signature workflow.

Naturally, PDF includes features which are related to these activities but are not essential to them. For example, support for adding signing reasons is tangential to signing, but valuable for many workflows.

Tip: For complete PDF language details, refer to the PDF Reference at http://www.adobe.com/devnet/pdf/pdf_reference.html.

Figure 1 PDF language dictionaries

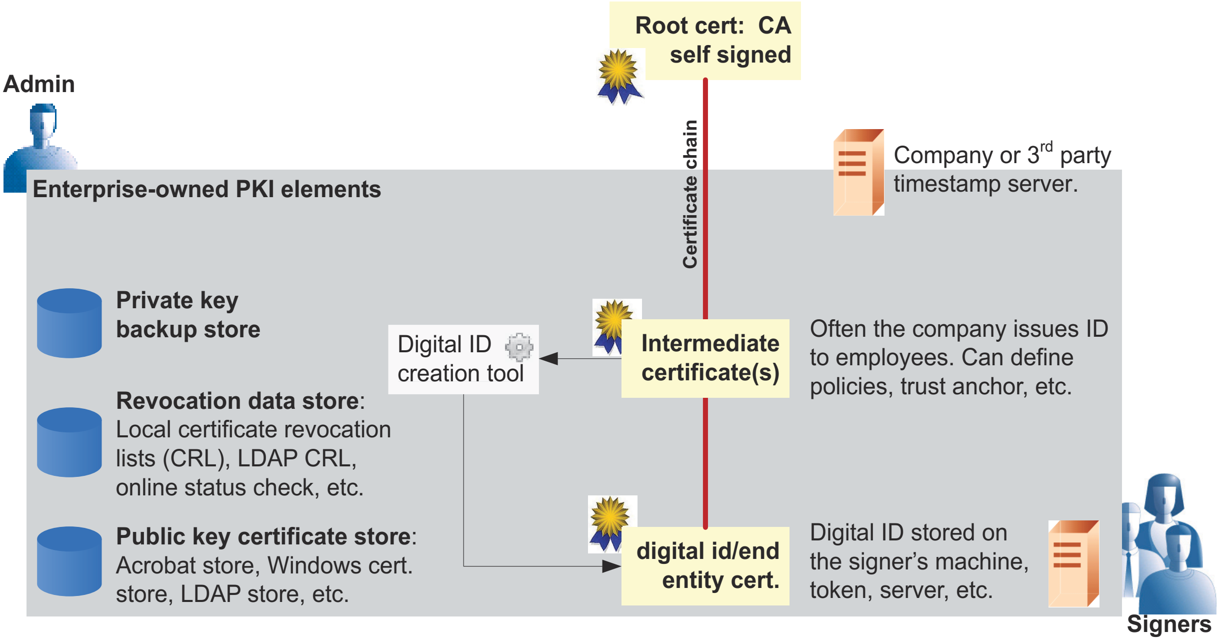

obj<<Permissions dictionary
/Perms</DocMDP (obj ID)>
/Perms</UR3>
obj<<Field dictionary
/FT (field type e.g. /Sig)
/Lock
/SV (optional: seed value)
/SigFlags (optional: 1 or 2)
/V (if signed, a sig dict. obj ID)
obj<<Field lock dictionary
/Type /SigFieldLock
/Action /All
/P 1
Doc locked on approval sig.
obj<< Seed value dictionary
/Type (SV)
/Ff (Specify required entries)
/Filter (Signature handler)
/SubFilter (Signature encoding)
/DigestMethod (Algorithm)
/V (SV parser capability)
/Reasons (Signing reasons list)
/MDP (Force certification sig)
/TimeStamp (TS dictionary)
/LegalAttestation (Attestations)
/AddRevInfo (Embed rev status)
/Cert (Certificate SV dictionary)
obj<< Signature dictionary
/Filter (signature handler)
/SubFilter (signature format)
/ByteRange (document range)
/Contents (PKCS sig value)
/Type Sig
. . . Other stuff . . .
/Reference (1-n sig ref dicts)
obj<< Certificate SV dictionary
/Type (SVCert)
/Ff (Specify required entries)
/ Subject (Identify certificates)
. . . Other stuff . . .
obj<< Signature reference dict.
/Type /SigRef
/TransformMethod
FieldMDP
DocMDP
UR3
/TransformParams
TransformMethod sets what TransformParams are used.
FieldMDP TransformParams
/Actions/(All | Inc. | Exc.)
/Fields/(Field names)
/P (1)
Field level permissions
DocMDP TransformParams
Allow sign, annots, etc.
/P (1 | 2 | 3)
Certification signature
UR3 TransformParams
/V /2.2
^{1}/Document/[FullSave]
^{2}/Form/[form field rights]
^{3}/Signature/[Modify]
^{4}/Annots/[annot rights]
^{5}/EF/[embedded file rights]
^{6}/FormEX/[form field rights]
1-6 define usage rights

6.2 Public key infrastructure

PDF's digital signature capabilities are designed for compatibility with all the standards associated with mainstream public key infrastructures (PKI) deployed in enterprise and government settings. A PKI is the set of people, policies, procedures, hardware, and software used in creating, distributing, managing, and revoking, and using the digital IDs that contain the public/private key pairs used when signing a PDF.

In the context of PDF signature workflows, "PKI" generally refers to the digital ID issuers, users, administrators, and any hardware or software used in those workflows. PDF viewers that implement and conform to the PDF language specification are able to interact with all of these components in a seamless and robust way.

Figure 2 Common PKI elements in signature workflows


Root cert: CA self signed
Admin
Certificate chain
Company or 3rd party timestamp server.
Enterprise-owned PKI elements
Private key backup store
Digital ID creation tool
Intermediate certificate(s)
Often the company issues ID to employees. Can define policies, trust anchor, etc.
Revocation data store: Local certificate revocation lists (CRL), LDAP CRL, online status check, etc.
Public key certificate store: Acrobat store, Windows cert. store, LDAP store, etc.
digital id/end entity cert.
Digital ID stored on the signer's machine, token, server, etc.
Signers

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 deemed trustworthy, you can trust the signature the notary witnesses. Using a PKI is a method of providing a similar kind of trust.

Some common PKI components directly related to providing trust include:

- **Certificate authority (CA)**: An ultimate trust authority that sells or issues digital IDs (such as Verisign or Geotrust). The CA signs it's own certificate (self-signs) and its certificate is typically the "root" certificate at the top of the certificate chain.
- **Intermediate certificates (ICAs)**: A type of CA whose certificate resides in the certificate chain between the end entity and root certificates. The certificate is not self-signed, and the ICA often provides services such as policies, timestamping, revocation lists, etc.
- **End entity certificate (EE)**: The signer's certificate and the last element of a signing chain. By definition, an end entity certificate does not contain the basic constraint value CA.

- **Digital ID**: An electronic representation of data based on the ITU-T X.509 v3 standard, associated with a person or entity. It is stored in a password-protected file on a computer or network, a USB token, a smart card, etc. A digital ID contains a public key certificate, a private key, and other data.
- **Public key certificate**: A file that contains the numeric public key portion of a public/private key pair along with the associated extensions and attributes used to define the certificates owner, validity period, and usage.
- **Private key**: The secret key in a PKI system, used to validate incoming messages and sign outgoing ones. A Private Key is always paired with its Public Key during those key generations.

While the digital ID and its issuing entities are central to any PKI, the PKI also includes many other enterprise-owned and 3rd party items. A PKI administrator will usually manage the creation and distribution of digital IDs, LDAP servers, timestamp servers, revocation lists, and other items. The PDF language supports all the data needed to interface with those components.

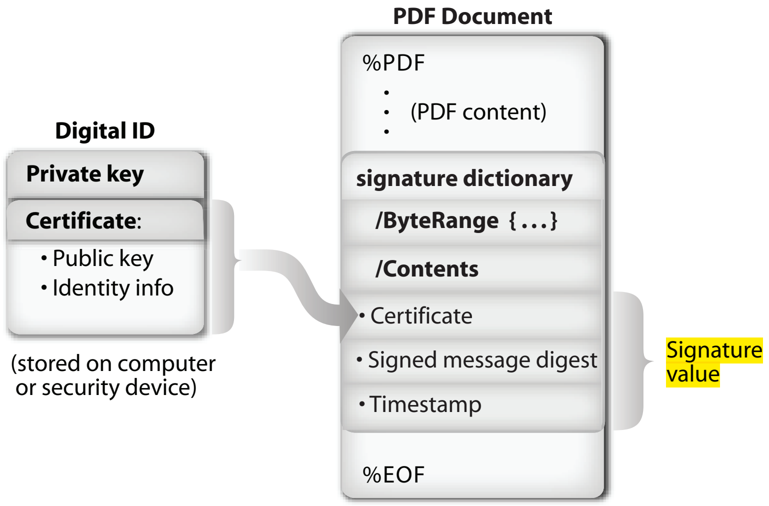
6.2.1 PKI, PDF, and signing

PDF 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 signed document.

Each digital signature in a PDF document is associated with a signature handler. The signature is placed in a PDF signature dictionary which contains the name of the signature handler which will be used to process that signature (Figure 3). The signature handler built into Adobe Acrobat leverages Public/ Private Key (PPK) cryptography technologies. PPK is based on the idea that a value encrypted with a private key can only be decrypted using the public key (the reverse may also be true when encrypting documents for specific recipients, but that is outside the scope of this document).

When a PDF is signed, the signer's certificate is embedded in the PDF file. Figure 3shows the relationship between the digital ID stored on the user's hardware device and the signature value embedded in the PDF document. The signature value may also include additional information such as a signature graphic, a time stamp, and other data that may be specific to the user, system, or application.

Figure 3 Digital ID and a signed PDF document



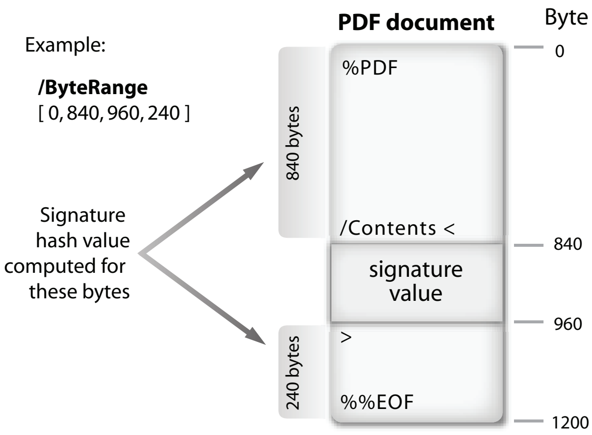
The signing process is as follows:

1. A document to be signed is turned into a stream of bytes.

2. The entire PDF file is written to disk with a suitably-sized space left for the signature value as well as with worst-case values in the `ByteRange` array.

   `ByteRange` is an array of four 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. The actual signature value is stored in the `/Contents` key between the end of the first sequence and the beginning of the second one. In Figure 4, the hash is calculated for bytes 0 through 839, and 960 through 1200.

Figure 4 The ByteRange and signature value



3. 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 zeros.

4. The hash of the entire file is computed, using the bytes specified by the real `ByteRange` value using a hash algorithm such as SHA-256. 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.

5. The hash value is encrypted with the signer's private key and a hex-encoded PKCS#7 object signature object is generated.

6. 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 zeros.

7. The PDF file is re-loaded in Acrobat to ensure that the in-memory and on-disk versions are identical.

   Tip: This is a high level view. For a more detailed quick key that shows some application-level configuration options, refer to Signature creation workflow.

6.2.2 PKI, PDF, and signature validation

Since private and public keys are merely numbers, anyone can generate a public and private key pair using any number of tools. Applications like Acrobat provide 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. Hence, an unknown self-signed certificate does not have a high level of assurance.

To solve this type of trust problem, organizations use a PKI that includes an independent authority that issues, records, and tracks digital IDs. Because PDF supports embedding the signer's public key as part of the signature, document recipient always have it for signature validation. To validate a signature, the validator simply retrieves the signer's certificate and compares it to their own list of trusted certificates:

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 identical, the signature is reported as known.

Tip: Whether or not the signature is trusted or valid are separate issues. Signature trust depends on the recipient's application configuration. Signature status also depends on a document integrity check.

6.3 PDF language signature features

6.3.1 Standards support

PDF is itself an open ISO standard. Digital signature support in PDF is fully described in ISO 32000, and Adobe provides tools for interacting with PDF and the Acrobat family of products' APIs in its open SDK. However, support for other standards is also built in to PDF, and PDF viewers like Acrobat and Adobe Reader should adhere to the accepted standards listed in Table 2.

6.3.2 Support for alternate signature methodologies

The majority of signatures are purely mathematical, such as the public/private-key encrypted document digest produced by Acrobat's default signature handler. However, they may also be a biometric form of identification, such as a handwritten signature, fingerprint, or retinal scan. Signature handler process the data and controls the form of authentication according to the rules defined in the PDF ISO standard.

6.3.3 Support for two signature types

PDF defines two types of signatures: *approval* and *certification*. Both types are byte range signatures over all file contents. Both take a visual snapshot of the document at the time it was signed and thus provide a high level of document integrity.

The differences are as follows:

- **Approval**: There can be any number of approval signatures in a document. The field may optionally be associated with FieldMDP permissions. See Locking form fields.
- **Certification**: There can be only one certification signature and it must be the first one in a document. The field is always associated with DocMDP (See Controlling post-signing changes) and Legal content attestations. They may optionally be associated with FieldMDP permissions.

6.3.4 Signature interoperability

PDF is designed to allow interoperability between signature handlers and conforming readers; that is, a PDF signed with handler ABC should be able to be validated with handler XYZ from a different vendor.

When present, the SubFilter entry in the signature dictionary specifies the encoding of the signature value and key information, while the Filter entry specifies the preferred handler that should be used to validate the signature. There are several defined values for the SubFilter entry, all based on public-key cryptographic standards published by RSA Security and also as part of the standards issued by the Internet Engineering Task Force (IETF) Public Key Infrastructure (PKIX) working group.

6.3.5 Robust algorithm support

As security concerns have evolved over time, PDF has extended support to increasingly strong encryption algorithms as shown below. Digest algorithms can be specified via seed values or at the application preference level such as the registry.

Table 1 Algorithm support

	Subfilter value		
	adbe.pkcs7.detached	**adbe.pkcs7.sha1**	**adbe.x509.rsa.sha1 (a)**
Message digest	SHA1 (PDF 1.3)	SHA1 (PDF 1.3) (b)	SHA1 (PDF 1.3)
	SHA256 (PDF 1.6)		SHA256 (PDF 1.6)
	SHA384 (PDF 1.7)		SHA384 (PDF 1.7)
	SHA512 (PDF 1.7)		SHA512 (PDF 1.7)
	RIPEMD160 (PDF1.7)		RIPEMD160 (PDF1.7)
RSA algorithms support	Up to 1024-bit (PDF 1.3)	See adbe.pkcs7. detached	See adbe.pkcs7.detached
	Up to 2048-bit (PDF 1.5)		
	Up to 4096-bit (PDF 1.5)		
DSA algorithms support	Up to 4096-bit (PDF 1.)	See adbe.pkcs7. detached	No

(a) Despite the appearance of sha1 in the name of this Subfilter value, supported encodings are not limited to the SHA1. The PKCS#1 object has an identifier that indicates which algorithm is used.

(b) Other algorithms may be used to digest the signed data field; however, SHA1 is used to digest the signed data.

6.3.6 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. 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 another person changes the document, the signer can easily argue that the document had been altered.

However, with PDF, any attempt to alter the document by modifying the file (such as signing it again) will invalidate the existing digital signature. This is so because the hash value calculated at verification time will not match the encrypted hash created at signing time.

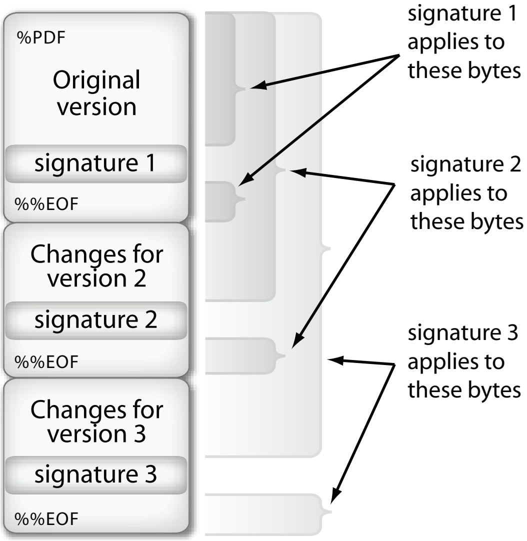
PDF solves this problem by supporting the ability to do incremental updates (see Incremental updates). As long as additional signatures are not prevented by other permissions restrictions (e.g. DocMDP and FieldMDP), a signer can just add another signature field to the document and sign it without invalidating the earlier signature

6.3.7 Incremental updates

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 generally, and of signed PDF files specifically, allows any PDF file 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.

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 5 shows how signatures are created for a file with three signatures.

Figure 5 Multiple signatures and incremental updates



6.3.8 Viewing previously signed document versions

The Incremental updates facility of the PDF language allows PDF viewers 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.

Acrobat takes full advantage of PDF's ability to "remember" a document's state at the time of signing by providing two features:

- **View Signed Version**: Display the document as it existed at the time that the signature was applied by right-clicking on a signature and choosing **View Signed Version**. It can be mimicked manually by removing any bytes in the PDF file after the EOF corresponding to the signature.
- **Compare Signed Version to Current Version**: Compare a document's current version with the signed version by right-clicking on a signature and choosing **Compare Signed Version to Current Version**.

6.3.9 Comparing current and signed document versions

See Viewing previously signed document versions.

6.3.10 Locking form fields

Form fields include both signature and non-signature fields, and forms often contain many form fields, some designed for signatures and other for form data. The PDF language allows authors to control whether additional fields can be filled in after a document is signed, either through signing or by entering any kind of data. When creating a document, an author can specify the following:

- Whether form fields can be filled in without invalidating the approval or certification signature.
- That after a specific recipient has signed the document, any modifications to specific form fields shall invalidate that recipient's signature. In this case, a separate signature field is designated for each recipient

The FieldMDP transform method is used detects changes to the values of a document's form fields (Figure 1).

6.3.11 Controlling post-signing changes

PDF provides a mechanism for limiting post-signing. That mechanism is the DocMDP transform method (shorthand for modification detection and prevention). The P entry in the DocMDP transform parameters dictionary indicates which of the following changes to the document will invalidate the signature (Figure 1):

- No changes
- Form fill-in and digital signatures
- Annotations (commenting), form fill-in, digital signatures

A document can contain only one signature field that contains a DocMDP transform method, and it must be the first signed field in the document. That signature is called a "certification" signature. This feature enables the author to specify what changes are permitted and what changes invalidate the author's signature. However, most users will perceive the effect of DocMDP as specifying what they can do to a document.

A certification signature should have a legal attestation dictionary that specifies all content that might result in unexpected rendering of the document contents, along with the author’s attestation to such content. This dictionary may be used to establish an author’s intent if the integrity of the document is questioned.

Acrobat allows the first signer to certify a document with a certification signature and set the permissions and legal attestations on the fly during signing.

When a certified document is opened, the signature is validated as usual, except that the document as it existed at the time it was certified is opened and compared to the document in memory which is currently being viewed, including all incremental changes. A modification analysis is done and any modifications that were prohibited by the author are reported. Permission violations result in an invalid signature.

6.3.12 Legal content attestations

Given its intrinsic richness, the PDF language provides a number of capabilities with the potential to cause variance in the rendered appearance of a PDF document (e.g. multimedia or JavaScript). These capabilities could be used to construct a document that misleads a document recipient, intentionally or unintentionally. These situations are relevant when considering the legal implications of a signed PDF document.

In order to facilitate document trust, conforming writers of certification signatures such as Acrobat should also leverage PDF’s legal attestation dictionary. Dictionary entries specify all content that may result in unexpected rendering of the document contents. Additionally, authors may provide further clarification of such content by means of the Attestation entry. Reviewers should establish for themselves that they trust the author and document contents.

6.3.13 Enabling features via document-based permissions

PDF enables a fully featured client with rich PDF interaction to grant document-specific permissions to less capable clients so that they can also use some of those features. Using this mechanism, a client that does not have digital signature capability may be granted that ability. When the permission is granted at the language level for a particular document, any associated signature related user interface elements would become enabled.

For example, an author using Acrobat can grant permissions to enable additional features in Adobe Reader, using public-key cryptography. It uses certificate authorities to issue public key certificates to document creators with which it has entered into a business relationship. Adobe Reader verifies that the rights-enabling signature uses a certificate from an Adobe-authorized certificate authority. Thus, an Adobe Reader user that opens a reader enabled document can sign, fill in form fields, and otherwise perform actions that would otherwise be prohibited.

This mechanism is called usage rights signatures and are typically transparent to end users. Usage rights signatures are referred to from the UR3 entry in the permissions dictionary (Figure 1). The signature enables additional interactive features that may not available by default in a conforming reader and validates that the permissions have been granted by a bona fide granting authority. The transform parameters dictionary specifies the additional rights that shall be enabled if the signature is valid. If the signature is invalid for any reason, additional rights are not granted.

6.3.14 Rich certificate processing

PDF enables feature rich certificate processing and handling because it certificate data is embedded in the signature. PDF viewers and signature handlers can be designed to use this data as needed. For example, when PKCS#7 signatures are used, the signature object can contain some or all of the following:

- Timestamp information
- Embedded revocation information
- Revocation checking details for both CRLs and OCSP
- Certificate polices and attribute certificates

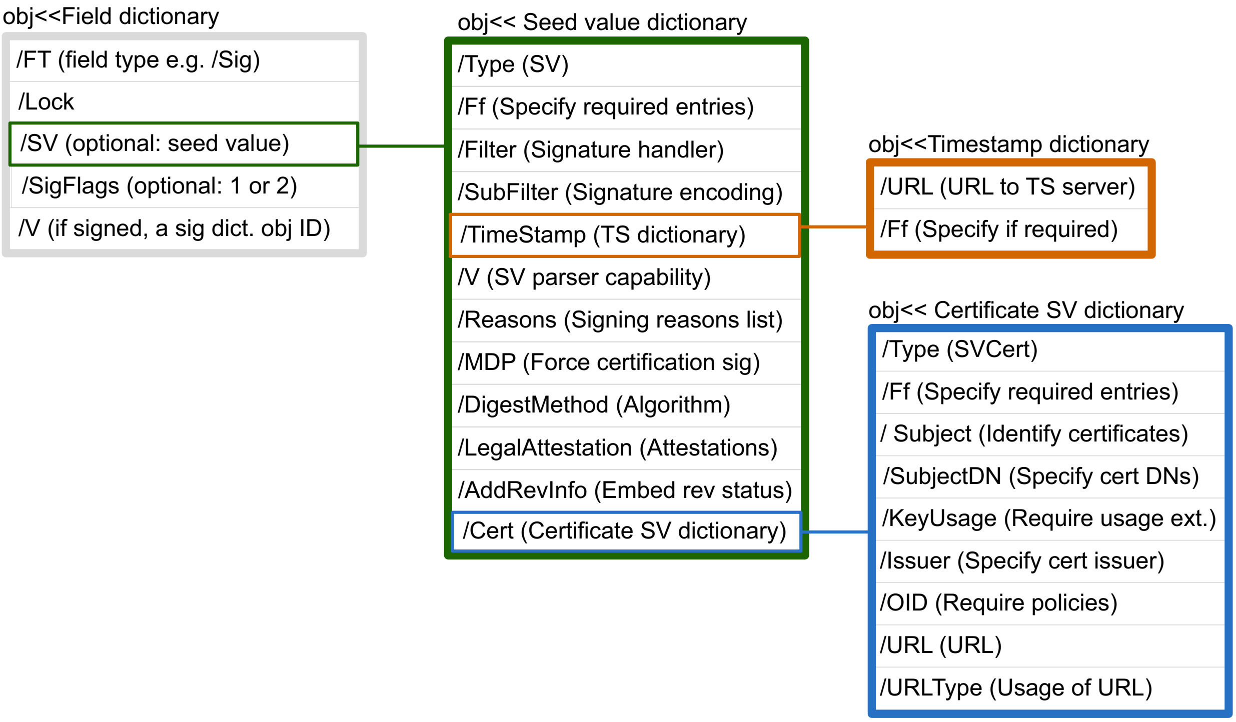
6.3.15 Controlling signature workflows via seed values

PDF's support for seed values provides authors with field-level control over document behavior once it has been routed to the signer. A seed value specifies an attribute and attribute value, and the author can control whether the specified parameter is optional or required for any particular field.

For example, you can use seed values to limit a user's choices when signing a particular signature field, such as requiring signing with a certificate issued by a particular CA. When a signer signs a "seeded" field, the author-specified behaviors are automatically invoked and enforced.

If a field dictionary contains an SV entry referencing a seed value dictionary then that dictionary is used when the field is signed. An Ff entry specifies whether the other entries in the dictionary shall be honoured or whether they are merely recommendations. Acrobat's default handler supports all the seed values defined by the PDF standard. Acrobat provides APIs for seeding fields.

Figure 6 Seed value dictionaries


obj<<Field dictionary
/FT (field type e.g. /Sig)
/Lock
/SV (optional: seed value)
/SigFlags (optional: 1 or 2)
/V (if signed, a sig dict. obj ID)
obj<< Seed value dictionary
/Type (SV)
/Ff (Specify required entries)
/Filter (Signature handler)
/SubFilter (Signature encoding)
/TimeStamp (TS dictionary)
/V (SV parser capability)
/Reasons (Signing reasons list)
/MDP (Force certification sig)
/DigestMethod (Algorithm)
/LegalAttestation (Attestations)
/AddRevInfo (Embed rev status)
/Cert (Certificate SV dictionary)
obj<<Timestamp dictionary
/URL (URL to TS server)
/Ff (Specify if required)
obj<< Certificate SV dictionary
/Type (SVCert)
/Ff (Specify required entries)
/ Subject (Identify certificates)
/SubjectDN (Specify cert DNs)
/KeyUsage (Require usage ext.)
/Issuer (Specify cert issuer)
/OID (Require policies)
/URL (URL)
/URLType (Usage of URL)

6.4 Signature creation workflow

The workflow is configurable through the GUI and registry settings.[3]

Supports 3rd party handlers.

Create signature appearances with the UI or programmatically. Contents are defined by the SigAP dictionary included in the form field.[4]

The signature appearance is included in the message digest.

The DSA/RSA algorithm is specified by the signer's digital ID private key in a P12/PFX file, smart card/token, roaming ID server, Mac Key Chain, or Windows store.

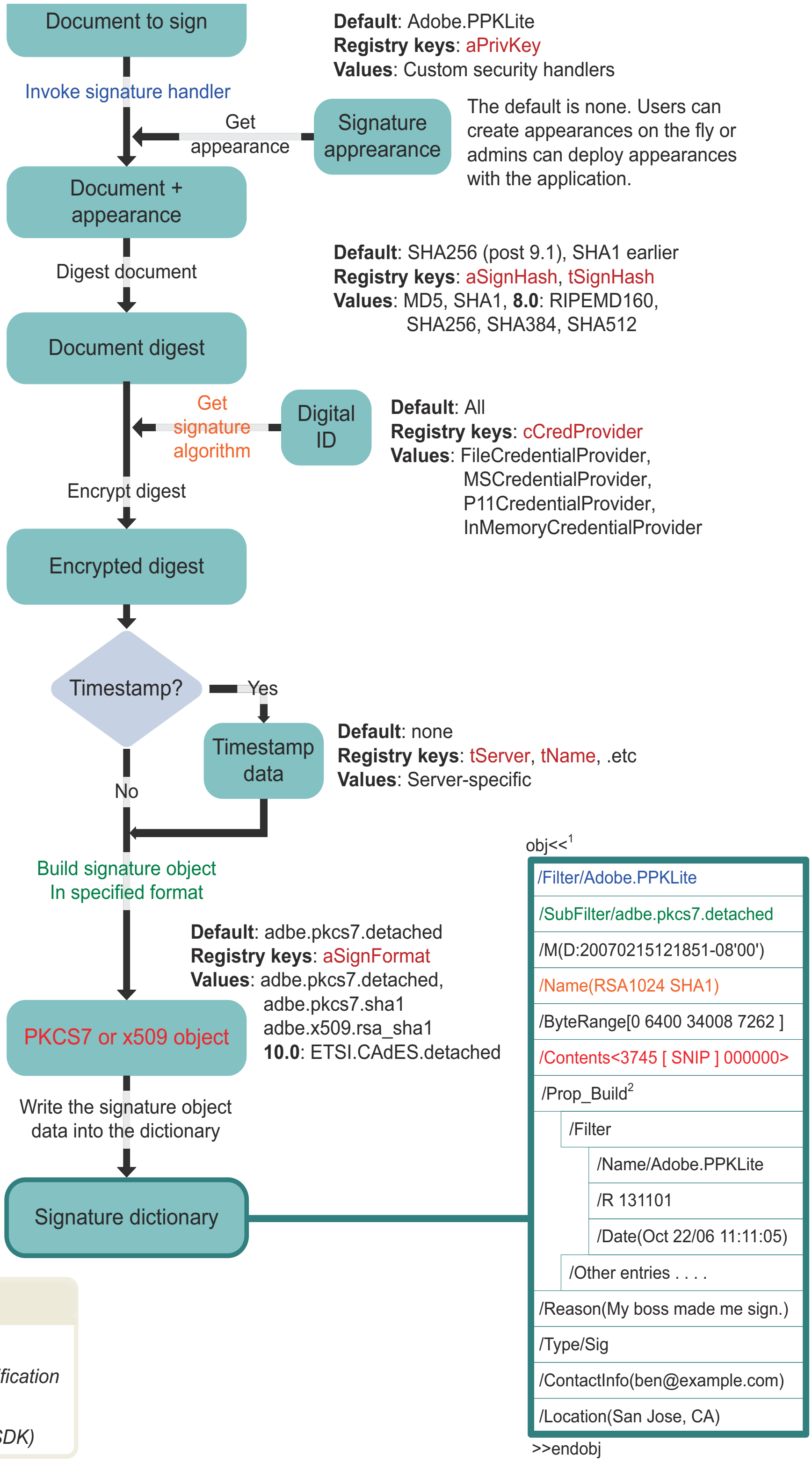
Revocation checking is configurable for any certificate in the chain, including the signers, the ICA, any timestamp CA, etc.[3]

If not time stamped or time stamping fails, use the local time.[3]

The signature object is built in memory until all the revocation checking, timestamp, and certificate data is retrieved.

The signature object is encoded using the Distinguished Encoding Rules (DER). The DER object is hex encoded and padded with zeros to make the signed byte range match the size that was set aside when the signing process began.

Dictionary contents are customizable through UI and registry settings such as cReasons, cContactInfo, etc.[3]



Default: Adobe.PPKLite
Registry keys: aPrivKey
Values: Custom security handlers

The default is none. Users can create appearances on the fly or admins can deploy appearances with the application.

Default: SHA256 (post 9.1), SHA1 earlier
Registry keys: aSignHash, tSignHash
Values: MD5, SHA1, **8.0**: RIPEMD160, SHA256, SHA384, SHA512

Default: All
Registry keys: cCredProvider
Values: FileCredentialProvider, MSCredentialProvider, P11CredentialProvider, InMemoryCredentialProvider

Default: none
Registry keys: tServer, tName, .etc
Values: Server-specific

Default: adbe.pkcs7.detached
Registry keys: aSignFormat
Values: adbe.pkcs7.detached, adbe.pkcs7.sha1 adbe.x509.rsa_sha1 **10.0**: ETSI.CAdES.detached

```
obj<<[1]
/Filter/Adobe.PPKLite
/SubFilter/adbe.pkcs7.detached
/M(D:20070215121851-08'00')
/Name(RSA1024 SHA1)
/ByteRange[0 6400 34008 7262 ]
/Contents<3745 [ SNIP ] 000000>
/Prop_Build[2]
    /Filter
        /Name/Adobe.PPKLite
        /R 131101
        /Date(Oct 22/06 11:11:05)
    /Other entries . . . .
/Reason(My boss made me sign.)
/Type/Sig
/ContactInfo(ben@example.com)
/Location(San Jose, CA)
>>endobj
```

REFERENCES

[1] *PDF Reference*
[2] *Digital Signature Build Dictionary Specification*
[3] *Digital Signatures Acrobat*
[4] *Digital Signature Appearances (in the SDK)*

6.5 Supported standards

Table 2 Standards support

Reference	Feature
PDF Reference 1.7 (ISO 32000-1) http://www.adobe.com/devnet/pdf/pdf_reference.html. See also PDF for Archive (PDF/A) and PDF for Exchange (PDF/X) at http://www.iso.org.	Representing signatures in the PDF language.
RFC 3280, Internet X.509 Public Key Infrastructure, Certificate and Certificate Revocation List (CRL) Profile. http://www.ietf.org	CRL revocation checking, chain building, path validation, cross certificates, multiple chains.
RFC 2560, X.509 Internet PKI Online Certificate Status Protocol-OCSP. http://www.ietf.org	OCSP revocation checking.
RFC 3161, Internet X.509 Public Key Infrastructure Time-Stamp Protocol http://www.ietf.org	Timestamping: signing and signature validation.
RFC 3281, Attribute Certificate Profile, S. Farrell, R.Housley April 2002. http://www.ietf.org	Attribute certificates.
RFC 2437, PKCS #1: RSA Cryptography Specifications Version 2.0. http://www.ietf.org	A format used for creating a digital signature object which is embedded in a document.
RFC 2898, PKCS #5: Password-Based Cryptography Specification Ver. 2.0. http://www.ietf.org	Password security.
RFC 2315, PKCS #7: Cryptographic Message Syntax, Version 1.5. http://www.ietf.org	A format used for creating a digital signature object which is embedded in a document.
RFC 1321, The MD5 Message-Digest Algorithm. http://www.ietf.org RFC 3174, US Secure Hash Algorithm 1 (SHA1). http://www.ietf.org	Creating a document hash during signing.
FIPS PUB 186-2, Digital Signature Standard, describes DSA signatures. http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf	Digital signatures.
FIPS PUB 197, Advanced Encryption Standard (AES). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf	Certificate security.
ISIS-MTT Specification v.1.1 March 2004. http://www.isis-mtt.org/index.php?id=460&L=1	Attribute certificates.
NIST PKITS "Public Key Interoperability Public Key Interoperability Test Suite Certification Path Validation", Draft Version 0.7 June 30, 2003	Chain building and path validation, including cross certificates and multiple chains.
OIDS. ASN.1	Object identifiers (OIDs)
RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax. http://www.ietf.org	All.
RFC 2595, Using TLS with IMAP, POP3 and ACAP. http://www.ietf.org	The PLAIN authentication mechanism used by the roaming ID feature.
RFC 3778, The application/pdf Media Type. Adobe Systems Incorporated.	Describes PDF media type, digital signature and encryption

6.6 Additional resources

More information on digital signatures, PDF, and the Acrobat family of Products reside in the security feature library at http://learn.adobe.com/wiki/display/security/Document+Library.

In particular, see:

- *Digital Signatures in Acrobat*: Implementation details describing how the Acrobat family of products uses digital signature capabilities of PDF.
- *Digital Signatures and Rights Management in Acrobat and Adobe Reader*: Configuration, usage, and technical details of interest to administrators.