Windows Authenticode Portable Executable Signature Format

Version 1.0 — March 21, 2008

Abstract

Authenticode® is a digital signature format that is used to determine the origin and integrity of software binaries. Authenticode is based on Public-Key Cryptography Standards (PKCS) #7 signed data and X.509 certificates to bind an Authenticode-signed binary to the identity of a software publisher. This paper contains the structure and technical details of the Authenticode signature format.

This paper does not discuss issuing or processing X.509 code signing certificates, use of Windows Software Development Kit tools to sign binaries, deployment of a code signing infrastructure, or related Windows® APIs. Information on these topics is available in ”Resources” at the end of this paper.

This information applies for the following operating systems:

- Windows Server® 2008
- Windows Vista®
- Windows Server 2003
- Windows® XP
- Windows 2000

References and resources discussed here are listed at the end of this paper.

For the latest information, see:

http://www.microsoft.com/whdc/winlogo/drvsign/Authenticode_PE.mspx

LICENSE AGREEMENT

Microsoft Windows® Authenticode® Portable Executable Signature Format Specification
Revision 1.0

Note: This specification is provided to aid in the development of certain development tools for the Microsoft Windows platform. However, Microsoft does not guarantee that it is a complete specification in all respects, and cannot guarantee the accuracy of any information presented after the date of publication. Microsoft reserves the right to alter this specification without notice.

Microsoft will grant a royalty-free license, under reasonable and non-discriminatory terms and conditions, to any Microsoft patent claims (if any exist) that Microsoft deems necessary for the limited purpose of use in software tools to generate digital signatures and in EFI firmware to verify the signatures, each exclusively in Portable Executable and Common Object File Format images.

Microsoft may have intellectual property rights covering subject matter in this specification. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this specification does not give you any license to any intellectual property rights, and no other rights are granted by implication, estoppel, or otherwise.

This specification is provided "AS IS." Microsoft makes no representations or warranties, express, implied, or statutory, as (1) to the information in this specification, including any warranties of merchantability, fitness for a particular purpose, non-infringement, or title; (2) that the contents of this specification are suitable for any purpose; nor (3) that the implementation of such contents will not infringe any third party patents, copyrights, trademarks, or other rights.

Microsoft will not be liable for any direct, indirect, special, incidental, or consequential damages arising out of or relating to any use or distribution of this specification.

Microsoft, Authenticode, MS-DOS, MSDN, Visual C++, Win32, Windows, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other product and company names mentioned herein may be the trademarks of their respective owners.

The foregoing names and trademarks may not be used in any manner, including advertising or publicity pertaining to this specification or its contents without specific, written prior permission from the respective owners.

Document History

Date	Change
July 29, 2008	Updated two URLs in Reference section. Did not change date or version number.
March 21, 2008	First publication

Contents

Introduction

Authenticode® is a digital signature format that is used, among other purposes, to determine the origin and integrity of software binaries. Authenticode is based on the Public-Key Cryptography Standards (PKCS) #7 standard and uses X.509 v3 certificates to bind an Authenticode-signed file to the identity of a software publisher.

One important use of Authenticode signatures is to digitally sign portable executable (PE) files, which include .exe, .dll, and .sys files. This paper describes the signature format that is used to sign PE files by using Signtool.exe with X.509 v3 certificates. The signatures can be verified on the following Windows versions:

- Windows Server 2008
- Windows Vista®
- Windows Server® 2003
- Windows® XP
- Windows 2000

Note: The format discussed in this paper is for the versions of Windows in the preceding list. Updates to this signature format might add new structures and exhibit new behavior. Earlier versions of Windows might be able to verify the signature format described in this paper.

This paper is limited to the Authenticode signature format for PE files and assumes a working knowledge of X.509 v3 certificates, public key infrastructure (PKI), PKCS #7, PKCS #9, and the Windows PE file format. For information on topics such as policies for issuing a code signing certificate, rules for processing X.509 certificates, Microsoft code signing tools (including SignTool.exe), deployment of code signing infrastructure, code signing APIs, or Authenticode signature formats for other file formats, see “Resources” at the end of this paper.

Overview

The Authenticode signature in a PE file is in a PKCS #7 **SignedData** structure. The signature asserts that:

- The file originates from a specific software publisher.
- The file has not been altered since it was signed.

The signature itself does not convey any information about the intent or quality of the software. However, signatures that are associated with programs such as the Windows Logo Program—that sign software only if it passes certain tests—can convey quality information.

A PKCS #7 **SignedData** structure contains the PE file's hash value, a signature created by the software publisher’s private key, and the X.509 v3 certificates that bind the software publisher’s signing key to a legal entity. A PKCS #7 **SignedData** structure can optionally contain:

- A description of the software publisher.
- The software publisher's URL.
- An Authenticode timestamp.

  The timestamp is generated by a timestamping authority (TSA) and asserts that a publisher’s signature existed before the specified time. The timestamp extends the lifetime of the signature when a signing certificate expires or is later revoked.

Authenticode signatures can be “embedded” in a Windows PE file, in a location specified by the **Certificate Table** entry in **Optional Header Data Directories**. When Authenticode is used to sign a Windows PE file, the algorithm that calculates the file's Authenticode hash value excludes certain PE fields. When embedding the signature in the file, the signing process can modify these fields without affecting the file's hash value.

Figure 1 provides a simplified overview of how an Authenticode signature is included in a Windows PE file. It includes the location of the embedded Authenticode signature and specifies which PE fields are excluded when calculating the PE file's hash value.

For details about the PE file structure, see ”Microsoft Portable Executable and Common Object File Format Specification” (PE/COFF specification).

For details on the PKCS #7 portion of the Authenticode signature see the Abstract Syntax Notation version 1 (ASN.1) structure definitions later in this paper.

For details on how the Authenticode PE hash value is calculated, see “Calculating the PE Image Hash” later in this paper.

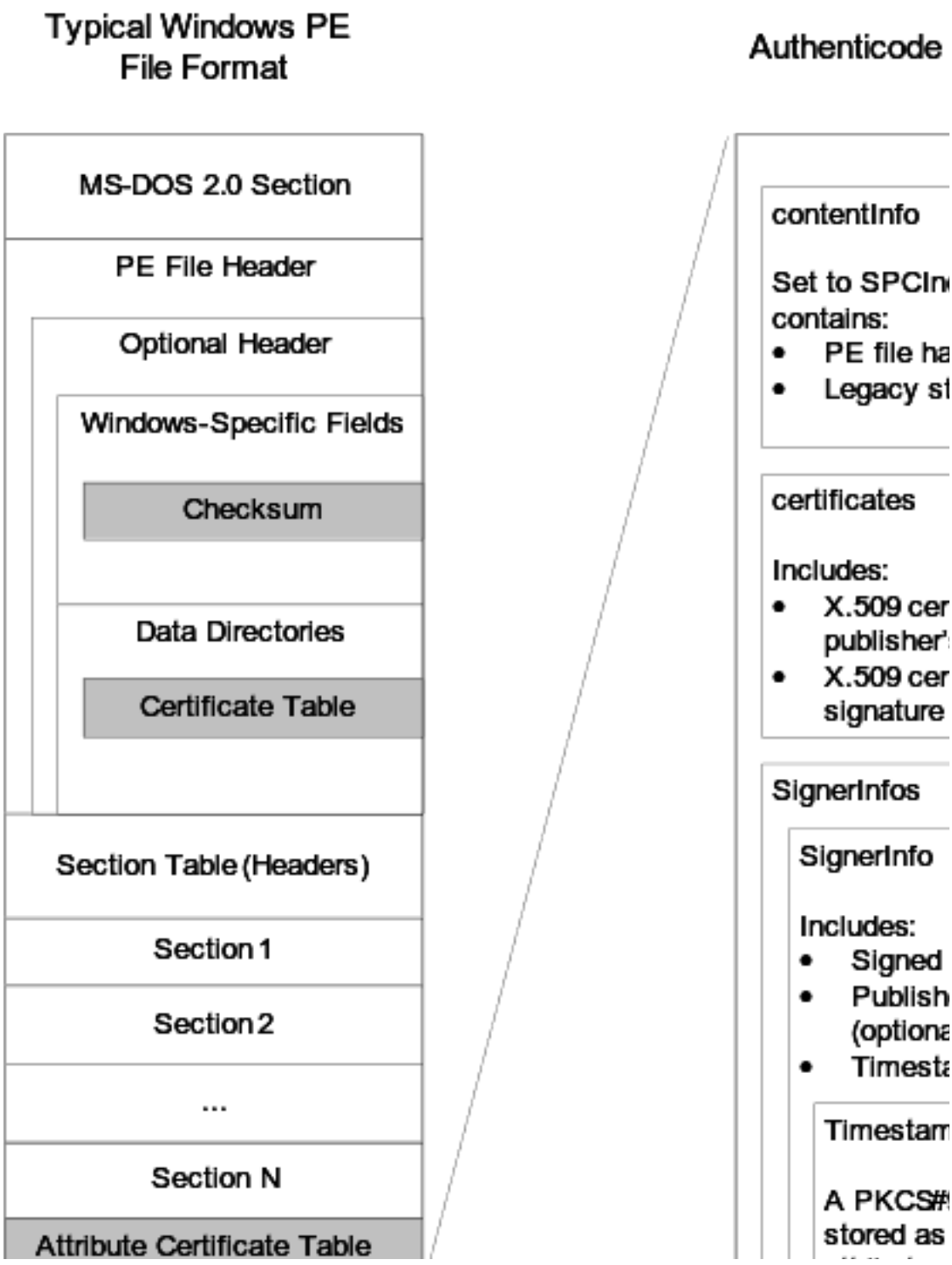

Figure 1. Overview of the Windows PE file format and the Authenticode signature format

Authenticode Profile of PKCS #7 SignedData

This profile describes which PKCS #7 structures and values are used in the Authenticode signature. For more details on the PKCS #7 standard, see ”PKCS #7: Cryptographic Message Syntax Standard.”

SignedData

The PKCS #7 v1.5 specification defines the following ASN.1 structure for **SignedData**:

```
SignedData ::= SEQUENCE {
  version Version,
  digestAlgorithms DigestAlgorithmIdentifiers,
  contentInfo ContentInfo,
  certificates
       [0] IMPLICIT ExtendedCertificatesAndCertificates
       OPTIONAL,
  Crls
    [1] IMPLICIT CertificateRevocationLists OPTIONAL,
  signerInfos SignerInfos }

DigestAlgorithmIdentifiers ::=
  SET OF DigestAlgorithmIdentifier

ContentInfo ::= SEQUENCE {
  contentType ContentType,
  content
    [0] EXPLICIT ANY DEFINED BY contentType OPTIONAL }

ContentType ::= OBJECT IDENTIFIER

SignerInfos ::= SET OF SignerInfo
```

The Authenticode profile of **SignedData** assigns the following values:

version
This field must be set to 1.

digestAlgorithms
This field contains the object identifiers (OIDs) of the digest algorithms that are used to sign the contents of the **ContentInfo** type, as defined by ”PKCS #7: Cryptographic Message Syntax Standard.” Because Authenticode signatures support only one signer, **digestAlgorithms** must contain only one **digestAlgorithmIdentifier** structure and the structure must match the value set in the **SignerInfo** structure's **digestAlgorithm** field. If not, the signature has been tampered with.

contentInfo
This field contains two fields:

- **contentType** must be set to SPC_INDIRECT_DATA_OBJID (1.3.6.1.4.1.311.2.1.4).
- **content** must be set to an **SpcIndirectDataContent** structure, which is described later.

certificates
This field contains a set of certificates. For Authenticode signatures, **certificates** contains the signer certificate and any intermediate certificates, but typically does not contain the root certificate. If the Authenticode signature is timestamped, **certificates** contains certificates that are used to verify the timestamp, which may include the root certificate. Authenticode certificate processing rules are described in “Authenticode Signature Verification” later in this paper.

Note: This paper specifies only the signature format of PE files that are signed with X.509 v3 certificates. For more information on processing X.509 v3 certificate chains, see ”Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.”

crls
This field is not used.

signerInfos
This field contains a set of **SignerInfo** structures, which contains information about the signatures. Because Authenticode supports only one signer, only one **SignerInfo** structure is in **signerInfos**. For details, see ”SignerInfo” later in this paper.

SignerInfo

For Authenticode signatures, **SignerInfos** contains one **SignerInfo** structure. The PKCS #7 v1.5 specification defines the following ASN.1 structure for **SignerInfo**:

```
SignerInfo ::= SEQUENCE {
  version Version,
  issuerAndSerialNumber IssuerAndSerialNumber,
  digestAlgorithm DigestAlgorithmIdentifier,
  authenticatedAttributes
    [0] IMPLICIT Attributes OPTIONAL,
  digestEncryptionAlgorithm
    DigestEncryptionAlgorithmIdentifier,
  encryptedDigest EncryptedDigest,
  unauthenticatedAttributes
    [1] IMPLICIT Attributes OPTIONAL }
IssuerAndSerialNumber ::= SEQUENCE {
  issuer Name,
  serialNumber CertificateSerialNumber }
EncryptedDigest ::= OCTET STRING
```

The Authenticode profile for **SignerInfo** assigns the following values:

version
This field must be set to 1.

issuerAndSerialNumber
This field contains an **issuerAndSerialNumber** structure, which contains the issuer name and serial number of the signing certificate, as defined by ”PKCS #7: Cryptographic Message Syntax Standard**.”**

digestAlgorithm
This field contains the OID of the digest algorithm that is used to sign the contents of **ContentInfo**, as defined by ”PKCS #7: Cryptographic Message Syntax Standard.” The **digestAlgorithm** value in the parent **SignedData** structure must match the **digestAlgorithm** value assigned in **signerInfo**. Algorithms supported by Authenticode include:

- SHA1 (1.3.14.3.2.26)
- MD5 (1.2.840.113549.2.5)

This algorithm is supported only for backwards-compatibility requirements and should not be used to sign new content.

authenticatedAttributes
This field contains a set of signed attributes. The following attributes are always present:

- **contentType** (1.2.840.113549.1.9.3)

This attribute contains a **messageDigest** OID (1.2.840.113549.1.9.4) as defined in ”PKCS #9: Selected Attribute Types.”

- **messageDigest** (1.2.840.113549.1.9.4)

This attribute contains an octet string with a hash value that is calculated as defined in ”PKCS #7: Cryptographic Message Syntax Standard.”

The follow signed attribute is always present in an Authenticode signature:

- **SPC_SP_OPUS_INFO_OBJID** (1.3.6.1.4.1.311.2.1.12)

  This attribute contains an **SpcSpOpusInfo** structure. For details, see "Authenticode-Specific Structures" later in this paper.

digestEncryptionAlgorithm
This field contains an OID that specifies the signature algorithm. Supported algorithms include:

- RSA (1.2.840.113549.1.1.1)
- DSA (1.2.840.10040.4.1)

encryptedDigest
This field contains the signature created by the signing certificate's private key, calculated as defined by the PKCS #7 specification.

unauthenticatedAttributes
If present, this field contains an **Attributes** object that in turn contains a set of **Attribute** objects. In Authenticode, this set contains only one **Attribute** object, which contains an Authenticode timestamp. The Authenticode timestamp is described in "Authenticode-Specific Structures" later in this paper.

Authenticode-Specific Structures

This part of the paper describes the Authenticode-specific structures in an Authenticode signature's PKCS #7 **SignedData** structure.

Authenticode-Specific Structures in ContentInfo

An Authenticode signature's **ContentInfo** structure contains several structures that in turn contain the file's hash value, page hash values (if present), the file description, and various optional or legacy ASN.1 fields. The root structure is **SpcIndirectDataContent**.

SpcIndirectDataContent

The following is the ASN.1 definition of **SpcIndirectDataContent**:

```
SpcIndirectDataContent ::= SEQUENCE {
    data                    SpcAttributeTypeAndOptionalValue,
    messageDigest           DigestInfo
} --#public—

SpcAttributeTypeAndOptionalValue ::= SEQUENCE {
    type                    ObjectID,
    value                   [0] EXPLICIT ANY OPTIONAL
}

DigestInfo ::= SEQUENCE {
    digestAlgorithm     AlgorithmIdentifier,
    digest              OCTETSTRING
}

AlgorithmIdentifier    ::=    SEQUENCE {
    algorithm           ObjectID,
    parameters          [0] EXPLICIT ANY OPTIONAL
}
```

The **SpcIndirectDataContent** structure has two members:

data
This field is set to an **SpcAttributeTypeAndOptionalValue** structure.

messageDigest
This field is set to a **DigestInfo** structure.

These structures are defined later.

The **SpcAttributeTypeAndOptionalValue** structure has two fields, which are set as follows for an Authenticode-signed PE file:

type
This field is set to SPC_PE_IMAGE_DATAOBJ OID (1.3.6.1.4.1.311.2.1.15).

value
This field is set to an **SpcPeImageData** structure, which is defined later.

The **DigestInfo** structure has two fields:

digestAlgorithm
This field specifies the digest algorithm that is used to hash the file. The value must match the **digestAlgorithm** value specified in **SignerInfo** and the parent PKCS #7 **digestAlgorithms** fields.

digest
This field is set to the message digest value of the file. For details, see “Calculating the PE Image Hash” later in the paper.

SpcPeImageData

The following is the ASN.1 definition of **SpcPeImageData**:

```
SpcPeImageData ::= SEQUENCE {
   flags                   SpcPeImageFlags DEFAULT { includeResources },
   file                    SpcLink
} --#public--

SpcPeImageFlags ::= BIT STRING {
    includeResources            (0),
    includeDebugInfo            (1),
    includeImportAddressTable   (2)
}

SpcLink ::= CHOICE {
    url                     [0] IMPLICIT IA5STRING,
    moniker                 [1] IMPLICIT SpcSerializedObject,
    file                    [2] EXPLICIT SpcString
} --#public--

SpcString ::= CHOICE {
    unicode                 [0] IMPLICIT BMPSTRING,
    ascii                   [1] IMPLICIT IA5STRING
}
```

The **SpcPeImageData** structure has two fields:

flags
This field specifies which portions of the Windows PE file are hashed. It is a 2-bit value that is set to one of the **SpcPeImageData** flags. Although **flags** is always present, it is ignored when calculating the file hash for both signing and verification purposes.

file
This field is always set to an **SPCLink** structure, even though the ASN.1 definitions designate **file** as optional.

SPCLink originally contained information that describes the software publisher, but it now has the following choices:

url [0]
This choice is not supported, but it does not affect signature verification if present.

moniker [1]
This choice is set to an **SpcSerializedObject** structure, which is described later.

file [2]
This is the default choice. It is set to an **SpcString** structure, which contains a Unicode string set to "<<<Obsolete>>>".

Warning to Implementers: There is an optional instance of **SpcString** in the **SignerInfo** structure that, if present, contains an ASCII string set to the publisher's URL. Do not confuse these instances of **SPCString**.

SpcSerializedObject

The following is the ASN.1 definition of **SpcSerializedObject**:

```
SpcSerializedObject ::= SEQUENCE {
    classId             SpcUuid,
    serializedData      OCTETSTRING
}

SpcUuid ::= OCTETSTRING
```

SpcUuid
The **SpcUuid** field is set to the following 10-byte octet string (a globally unique identifier—GUID) if **SpcSerializedObject** is present:

```
a6 b5 86 d5 b4 a1 24 66  ae 05 a2 17 da 8e 60 d6
```

serializedData
The **serializedData** field contains a binary structure. When present in an Authenticode signature generated in Windows Vista, **serializedData** contains a binary structure that contains page hashes. However, the definition of this binary structure is outside the scope of this paper.

Authenticode-Specific SignerInfo UnauthenticatedAttributes Structures

The following Authenticode-specific data structures are present in **SignerInfo** authenticated attributes.

SpcSpOpusInfo

SpcSpOpusInfo is identified by SPC_SP_OPUS_INFO_OBJID (1.3.6.1.4.1.311.2.1.12) and is defined as follows:

```
    SpcSpOpusInfo ::= SEQUENCE {
        programName             [0] EXPLICIT SpcString OPTIONAL,
        moreInfo                [1] EXPLICIT SpcLink OPTIONAL,
    } --#public--
```

SpcSpOpusInfo has two fields:

programName
This field contains the program description:

- If publisher chooses not to specify a description, the **SpcString** structure contains a zero-length program name.
- If the publisher chooses to specify a description, the **SpcString** structure contains a Unicode string.

moreInfo
This field is set to an **SPCLink** structure that contains a URL for a Web site with more information about the signer. The URL is an ASCII string.

Authenticode-Specific SignerInfo UnsignedAttrs Structures

The following Authenticode-specific data structures are present in **SignerInfo** unsigned attributes.

Authenticode Timestamp

The Authenticode timestamp is a PKCS #9 v1 countersignature located in the software publisher's **SignerInfo unauthenticatedAttributes**. The timestamp is generated by a TSA and signs both the hash value of the **SignerInfo** structure's **encryptedDigest** field and the Coordinated Universal Time (UTC) time at which the timestamp was generated. The timestamp asserts that the signature existed before the UTC time specified by the timestamp.

The timestamp certificate chain—including the root certificate—is added to the PKCS #7 **SignedData certificates** structure, although the root certificate is not required.

The following is the timestamp attribute's OID type:

```
szOID_RSA_counterSign 1.2.840.113549.1.9.6
```

The timestamp attribute content contains a PKCS #9 countersignature. The values in the countersignature are set by the TSA in accordance with ”PKCS #9: Selected Attribute Types.” The ASN.1 definition of **SignerInfo** that the countersignature uses is the same as that discussed in ”SignerInfo” earlier in this paper.

The Authenticode timestamp **SignerInfo** structure contains the following **authenticatedAttributes** values:

- ContentType (1.2.840.113549.1.9.3) is set to PKCS #7 Data (1.2.840.113549.1.7.1).
- Signing Time (1.2.840.113549.1.9.5) is set to the UTC time of timestamp generation time.
- Message Digest (1.2.840.113549.1.9.4) is set to the hash value of the **SignerInfo** structure's **encryptedDigest** value. The hash algorithm that is used to calculate the hash value is the same as that specified in the **SignerInfo** structure’s **digestAlgorithm** value of the timestamp.

Authenticode Signature Verification

The most common Authenticode verification policy is implemented by the Win32® **WinVerifyTrust** function with *pgActionID* set to WINTRUST_ACTION_GENERIC_VERIFY_V2. This section describes how Authenticode signatures are verified against this policy.

Note: This portion of the paper refers to many PE structures described in the PE/COFF specification. You will find it helpful to have that document available for reference.

Extracting and Verifying PKCS #7

The Authenticode signature is in a location that is specified by the **Certificates Table** entry in **Optional Header Data Directories** and the associated **Attribute Certificate Table**.

Note: “Attribute Certificate” as used by the PE/COFF specification does not refer to X.509 attribute certificates, as used in a PKI context. This is an unfortunate name collision.

The Authenticode signature is in a WIN_CERTIFICATE structure, which is declared in Wintrust.h as follows:

```
typedef struct _WIN_CERTIFICATE
{
    DWORD       dwLength;
    WORD        wRevision;
```

```
    WORD        wCertificateType;
    BYTE        bCertificate[ANYSIZE_ARRAY];
} WIN_CERTIFICATE, *LPWIN_CERTIFICATE;
```

The fields in WIN_CERTIFICATE are set to the following values:

- **dwLength** is set to the length of **bCertificate**.
- **wRevision** is set to the WIN_CERTIFICATE version number.

Value	Name	Notes
0x0100	WIN_CERT_REVISION_1_0	Version 1 is the legacy version of WIN_CERTIFICATE. It is supported only for verifying legacy Authenticode signatures.
0x0200	WIN_CERT_REVISION_2_0	Version 2 is the current version of WIN_CERTIFICATE.

- **wCertificateType** is set to 0x0002 for Authenticode signatures. This value is defined in Wintrust.h as WIN_CERT_TYPE_PKCS_SIGNED_DATA.
- **bCertificate** is set to a variable-length binary array that contains the Authenticode PKCS #7 **signedData**.

The PKCS #7 integrity is verified as described in ”PKCS #7: Cryptographic Message Syntax Standard.”

Certificate Processing

The software publisher's signing certificate and certificate chain are verified against the following criteria:

- The certificate chain is built to a trusted root certificate by using X.509 chain-building rules, as specified by IETF RFC 3280 “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.”

The trusted root certificate is configured in the Trusted Root Certification Authorities certificate store. For more information on certificate stores, see “Certificates Stores.”

- The signing certificate must contain either the extended key usage (EKU) value for code signing, or the entire certificate chain must contain no EKUs. The following is the EKU value for code signing:

```
szOID_PKIX_KP_CODE_SIGNING 1.3.6.1.5.5.7.3.3
```

- In Windows XP and Windows 2003, the signing certificate must not be in the Untrusted Certificates certificate store. In Windows Vista, none of the certificates in the chain, including the root certificate, can be in the Untrusted Certificates certificate store.
- The certificate must be within its validity period or the signature must have been timestamped, as discussed in ”Timestamp Processing” later in this paper.
- Revocation checking—to check certificate validity—is optional, but is used by many Windows components and by applications that call **WinVerifyTrust**.

Timestamp Processing

By default, timestamping an Authenticode signature extends the lifetime of the signature indefinitely, as long as that signature was timestamped, both:

- During the validity period of the signing certificate.
- Before the certificate revocation date, if applicable.

The signature lifetime is not extended if the “lifetime signer OID” (szOID_KP_LIFETIME_SIGNING) is present in the signing certificate or if WTD_LIFETIME_SIGNING_FLAG is set in the WINTRUST_DATA structure when calling **WinVerifyTrust**. For details, see “Timestamp Processing with Lifetime Signing Semantics.”

The certificates associated with the timestamp are in the PKCS #7 **SignedData** structure's **certificates** field.

Timestamp chains are compared against the following criteria:

- The certificate chain is built to a trusted root certificate by using X.509 chain-building rules.

  The trusted root certificate is configured in the Trusted Root Certification Authorities certificate store. For more information on certificate stores, see “Certificates Stores.”

- The TSA certificate that is used to sign the timestamp contains the following EKU:

```
szOID_PKIX_KP_TIMESTAMP_SIGNING 1.3.6.1.5.5.7.3.8
```

- The signing certificate must not be in the Untrusted Certificates certificate store.

  Note: If the certificate that is used to sign the timestamp is in the Untrusted Certificates certificate store, then the signature is not verified even if the software publisher certificate is still within its validity period.

- Revocation checking is turned off by default for checking the validity of the timestamping certificate.

Timestamp Processing with Lifetime Signing Semantics

Applications or certification authorities that do not want timestamped signatures to verify successfully for an indefinite period of time have two options:

- Set the lifetime signer OID in the publisher’s signing certificate.

  If the publisher’s signing certificate contains the lifetime signer OID in addition to the PKIX code signing OID, the signature becomes invalid when the publisher’s signing certificate expires, even if the signature is timestamped. The lifetime signer OID is defined as follows:

```
szOID_KP_LIFETIME_SIGNING 1.3.6.1.4.1.311.10.3.13
```

- Set the WTD_LIFETIME_SIGNING_FLAG in the WINTRUST_DATA structure when calling **WinVerifyTrust**.

  If a **WinVerifyTrust** caller sets WTD_LIFETIME_SIGNING_FLAG in the WINTRUST_DATA structure and the publisher’s signing certificate has expired, **WinVerifyTrust** reports the signature as invalid even if the signature is timestamped.

If a publisher revokes a code signing certificate that contains the lifetime signer OID or a **WinVerifyTrust** caller sets WTD_LIFETIME_SIGNING_FLAG in the WINTRUST_DATA structure, **WinVerifyTrust** reports the signature as valid if both of the following conditions are met:

- The signature was timestamped before the revocation date.
- The signing certificate is still within its validity period. After the validity period expires, the signature becomes invalid.

Calculating the PE Image Hash

After the integrity and identity of the Authenticode signature is verified, the final step of the signature verification process compares the original file hash value—which was calculated when the file was signed—to the hash value calculated from the current PE file. If the two hash values do not match, the file has been modified since it was signed and the signature is invalid. The file's original hash value is in the **Digest** field of the **SpcIndirectDataContent** structure's **DigestInfo** structure.

The Win32 **ImageGetDigestStream** function supports several methods to compute a PE file's hash value. Authenticode uses only one of these methods. The procedure for calculating the image hash value is described later in this section. It is a simplified version of the procedure performed by **ImageGetDigestStream** and calculates the correct hash value for almost all Authenticode-signed PE files.

To summarize, the hash calculation procedure includes:

- Hashing the PE Header, omitting the file's **checksum** and the **Certificate Table** entry in **Optional Header Data Directories** (Steps 3 through 7).
- Sorting and hashing the PE sections (steps 8 through 13).
- Omitting **Attribute Certificate Table** from the hash calculation and hashing any remaining data (steps 14 and 15).

Figure 1, in "Introduction," provides a visual representation of the PE format and shows which PE objects are excluded from the Authenticode hash value calculation. The hash value calculation omits these parts of the PE file because they are modified by the act of adding an Authenticode signature to the file. If the hash calculation did not omit these parts of the file, signing the file would change the file's hash value and invalidate the Authenticode signature.

To calculate the hash value

1. Load the image header into memory.
2. Initialize a hash algorithm context.
3. Hash the image header from its base to immediately before the start of the checksum address, as specified in **Optional Header Windows-Specific Fields**.
4. Skip over the **checksum**, which is a 4-byte field.
5. Hash everything from the end of the **checksum** field to immediately before the start of the **Certificate Table** entry, as specified in **Optional Header Data Directories**.
6. Get the **Attribute Certificate Table** address and size from the **Certificate Table** entry. For details, see section 5.7 of the PE/COFF specification.
7. Exclude the **Certificate Table** entry from the calculation and hash everything from the end of the **Certificate Table** entry to the end of image header, including **Section Table** (headers).The **Certificate Table** entry is 8 bytes long, as specified in **Optional Header Data Directories**.
8. Create a counter called SUM_OF_BYTES_HASHED, which is not part of the signature. Set this counter to the **SizeOfHeaders** field, as specified in **Optional Header Windows-Specific Field**.
9. Build a temporary table of pointers to all of the section headers in the image. The **NumberOfSections** field of **COFF File Header** indicates how big the table should be. Do not include any section headers in the table whose **SizeOfRawData** field is zero.
10. Using the **PointerToRawData** field (offset 20) in the referenced **SectionHeader** structure as a key, arrange the table's elements in ascending order. In other words,

sort the section headers in ascending order according to the disk-file offset of the sections.

11. Walk through the sorted table, load the corresponding section into memory, and hash the entire section. Use the **SizeOfRawData** field in the **SectionHeader** structure to determine the amount of data to hash.
12. Add the section's **SizeOfRawData** value to SUM_OF_BYTES_HASHED.
13. Repeat steps 11 and 12 for all of the sections in the sorted table.
14. Create a value called FILE_SIZE, which is not part of the signature. Set this value to the image's file size, acquired from the underlying file system. If FILE_SIZE is greater than SUM_OF_BYTES_HASHED, the file contains extra data that must be added to the hash. This data begins at the SUM_OF_BYTES_HASHED file offset, and its length is:

```
(File Size) - ((Size of AttributeCertificateTable) + SUM_OF_BYTES_HASHED)
```

    Note: The size of **Attribute Certificate Table** is specified in the second ULONG value in the **Certificate Table** entry (32 bit: offset 132, 64 bit: offset 148) in **Optional Header Data Directories.**

15. Finalize the hash algorithm context.

Note: This procedure uses offset values from the PE/COFF specification, version 8.1 . For authoritative offset values, refer to the most recent version of the PE/COFF specification.

Resources

Applicable Standards

The following industry standards are used as a basis for Authenticode:

Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
http://www.ietf.org/rfc/rfc3280.txt

PKCS #7: Cryptographic Message Syntax Standard
http://www.rsa.com/rsalabs/node.asp?id=2129

PKCS #9: Selected Attribute Types
http://www.rsa.com/rsalabs/node.asp?id=2131

Authenticode PE Signature Format References

The following links contain information that is directly related to this paper:

Microsoft Portable Executable and Common Object File Format Specification
http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx

Object IDs associated with Microsoft cryptography
http://support.microsoft.com/kb/287547

WinVerifyTrust Function
http://msdn2.microsoft.com/en-us/library/aa388208.aspx

ImageGetDigestStream Function
http://msdn2.microsoft.com/en-us/library/ms680160.aspx

General Code Signing References

Code Signing Best Practices
http://www.microsoft.com/whdc/winlogo/drvsign/best_practices.mspx

Kernel-Mode Code Signing Walkthrough
http://www.microsoft.com/whdc/winlogo/drvsign/kmcs_walkthrough.mspx

Windows SDK Tools to Sign Files and Check Signatures (SignTool, MakeCat, and SetReg)

http://msdn2.microsoft.com/en-us/library/aa388151(VS.85).aspx

Windows Driver Kit (WDK) (contains Authenticode signing tools)

http://www.microsoft.com/whdc/devtools/WDK/default.mspx

Microsoft Windows Software Development Kit for Windows Vista and .NET Framework 3.0 Runtime Components

http://www.microsoft.com/downloads/details.aspx?FamilyID=4377F86D-C913-4B5C-B87E-EF72E5B4E065&displaylang=en

Certificate stores

http://technet2.microsoft.com/windowsserver/en/library/1c4d3c02-e996-450a-bf4f-9a12d245a7eb1033.mspx?mfr=true