ISO/IEC JTC 1/SC 22/WG 23 N 0399

Date: 2012-05-25

ISO/IEC IS 17960

Secretariat: ANSI

Information Technology—Programming languages, their environments and system software interfaces—Code Signing for Source Code

Warning

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: International standard
Document subtype: if applicable
Document stage: (20) development stage
Document language: E

Table of Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2. Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. ISO/IEC IS 17960, which is an International Standard, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

Introduction

Source code is written and is used in many critical applications. Knowing that the source code being relied upon is the same as that which was used in testing is vital to ensuring the safety and security of a particular application. Given the ease with which source code can be modified, some method of protection of the source code is necessary. Sequestration of the source code is one method, but ensuring protection in that way is impractical and unreliable. Virtual protection through the use of a digital signature offers a practical solution and provides integrity even though the source code may traverse an insecure supply chain.

Modifications to source code are frequently made to correct the software or to adapt it for other purposes. Rarely are the modifications made by the original author. Revision control software allows for a tracking of the software changes, but those changes can be easily spoofed. Digital code signing provides a means to assign a responsible party to each of the modifications as they are made.

This International Standard specifies the APIs necessary for signing source code in a manner that allows signatures to be applied to ensure the integrity and a means for reversing the application of the signatures to unwrap the source code. Annex A provides a step by step description of a typical application of source code signing and the APIs necessary for each step in the process. A bibliography lists documents that were referred to during the preparation of this standard.

Information Technology — Programming Languages — Code Signing for Source Code

1. Scope

This document uses a language-neutral and environment-neutral description to define the application program interfaces (APIs) and supporting data structures necessary to support the signing of code and executables. It is intended to be used by both applications developers and library implementers.
The following areas are outside the scope of this specification:

- Graphics interfaces
- Object or binary code portability
- System configuration and resource availability

2. Normative References

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 10118–2:2000, *Information technology — Security techniques — Hash-functions — Part 2: Hash-functions using an n-bit block cipher*

ISO/IEC 10118–3:2003, *Information technology — Security techniques — Hash-functions — Part 3: Dedicated hash functions*

ISO/IEC 14750:1999, *Information technology -- Open Distributed Processing -- Interface Definition Language*

ITU-T, "Information Technology - Open Systems Interconnection - The Directory: Public-Key and Attribute Certificate Frameworks", Recommendation X.509, August 2005, http://www.itu.int/rec/T-REC-X.509/en.

3. Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

3.1 digital certificate

electronic block of data received from a trusted certificate authority that certifies the authenticity of the sender’s public key, identifies the creator of the sender’s public/private key, and contains the sender’s public key

3.2 digital certificate

electronic block of data received from a trusted certificate authority that certifies the authenticity of the sender's public key, identifies the creator of the sender's public/private key, and contains the sender's public key

3.3 digital signature

data appended to, or a cryptographic transformation of, a data unit that allows the recipient of the data unit to prove the source and integrity of the data unit and protect against forgery

3.4 private key

key of an entity's asymmetric key pair which should only be used by that entity and shall not normally be disclosed

3.5 public key

key of an entity's asymmetric key pair which can be made public

3.6 public key encryption

a cryptographic technique which enables users to securely communicate on an insecure public network and reliably verify the identity of a user via digital signatures

4. Conformance

An implementation of code signing conforms to this International Standard if it provides the interfaces specified in Clause 6.

Clause 5 is informative, providing an overview of the concepts of code signing. Annex A, also informative, provides a possible scenario of usage for the interfaces specified in Clause 6.

5. Concepts

Code signing is a technique for providing a digital signature for scripts and source code supporting the verification of the origin and supporting the verification that it has not been altered since it was signed.

Code signing can provide several valuable functions such as:

- knowledge of where the code originated
- confidence that the code has not been accidentally or maliciously altered
- verification of the identity of the responsible party for the code
- accountability for the code
- non-repudiation of the source of the code

Code Signing identifies to customers the responsible party for the code and confirms that it has not been modified since the signature was applied. In traditional software sales where a buyer can physically touch a package containing the software, the buyer can confirm the source of the application and its integrity by examining the packaging. However, most software is now procured via the Internet. Software procurement is not limited to complete applications as code snippets, plug-ins, add-ins, libraries, methods, drivers, etc. are all downloaded over the Internet. Verification of the source of the software is extremely important since the security and integrity of the receiving systems can be compromised by faulty or malicious code. In addition to protecting the security and integrity of the software, code signing provides authentication of the author, publisher or distributor of the code, and protects the brand and the intellectual property of the developer of the software by making applications uniquely identifiable and more difficult to falsify or alter.

When software source code is associated with a publisher's unique signature, distributing software on the Internet is no longer an anonymous activity. Digital signatures ensure accountability, just as a manufacturer's brand name ensures accountability with packaged software. Distributions on the Internet lack this accountability and code signing provides a means to offer accountability. Accountability can be a strong deterrent to the distribution of harmful code. Even though software may be acquired or distributed from an untrusted site or a site that is unfamiliar, the fact that it is signed by a known and trusted entity allows the software to be used with confidence that it has not been changed.

In addition to the valuable functions that code signing offers, this International Standard will specifically facilitate the following capabilities:

- a tracking mechanism to show what has been altered in the code and by whom
- multiple signatures to allow an audit trail of the signed object
- versioning information
- storage of other meta data about an object

The capability for a tracking mechanism and multiple signatures for one piece of code would be needed in some cases in order to create a digital trail through the origins of the code. Consider a signed piece of code. Someone should be able to modify a portion of the code, even if just one line or even one character, without assuming responsibility for the remainder of the code. A recipient of the code should be able to identify the responsible party for each portion of the code. For instance, a very trustworthy company A produces a driver. Company B modifies company A's driver for a particular use. Company B is not as trusted or has an unknown reputation. The recipient should be able to determine exactly what part of the code originated with company A and what was added or altered by company B so as to be able to concentrate their evaluation on the sections of code that company B either added or altered. This necessitates a means to keep track of the modifications made from one digital signature to the next.

An alternative scenario is software offered by company B that contains software from company A. Company B does not alter company A's software, but incorporates it into a package or suite of software. It would be useful to a customer to be able to identify the origin of each portion of the software.

6. Structures and APIs

6.1 General

The structures and APIs described below are intended to be language and platform independent. A particular language implementation will need to specify, for instance, an appropriate convention for specifying options and determine how error reporting will be done.

The structures and APIs are described with a syntax that is independent of any particular programming language, using the Interface Description Language (IDL) provided by ISO/IEC 14750:1999.
The identifiers used within the APIs are expressed using camel case (e.g. *isIntTrue* instead of underscores *is_int_true*). Particular language implementations may prefer to implement the APIs using underscores. Either is acceptable as long as the implementation is consistent throughout the language implementation and with ISO/IEC 14750:1999 Section 4.1.3.

6.2 Source Code File Format

For the initial signing of a source code file, a hash is generated for the source code. The signature block for the source code file consisting of the hash followed by the developer's digital certificate is then added to the beginning of the file. When modifications are made to previously signed code, the changes are recorded and appended to the beginning of the source code file and a new hash of this file is generated. The new signature block is then added to the beginning of the file. This allows the series of modifications, which can be thought of as encapsulations, to be reversed one at a time. The newest signature can be removed from the code and the changes that were made to the code from the next newest signature can be reversed to identify the previously signed version.

The initial signing of source code, by default, stores the changes in snapshot format. Subsequent encapsulations can use either snapshot or changeset format.

6.3 Structures

Additional descriptions of the fields used in these structures are available in ITU-T Recommendation X.509.

```
struct algorithmIdentifierStruct {
  unsigned short algorithm;      // used to identify the cryptographic
                                 // algorithm
  string parameters;             // optional parameters associated
                                 // with the algorithm
}

struct certStruct {              // structure for an X.509 certificate
  unsigned short version;        // certificate format version
  unsigned long serialNumber;    // unique identifier generated by
                                 // the certificate issuer
  algorithmIdentifierStruct algorithmID;    // the algorithm used by
```



```
                                  // the issuer to sign the certificate
  string issuerName;              // representation of its issuer's
                                  // identity in the form of a
                                  // Distinguished Name
  int validNotBeforeDate;         // the start of the time period in
                                  // which a certificate is intended
                                  // to be used
  int validNotAfterDate;          // the end of the time period in
                                  // which a certificate is intended
                                  // to be used
  string subjectName;             // a representation of its
                                  // subject's identity in the form
                                  // of a Distinguished Name
  unsigned short publicKeyAlgorithm;  // public key algorithm to be
                                  // used
  string subjectPublicKey;        // public key component of its
                                  // associated subject
  string issuerUniqueIdentifier; // optional issuer unique
                                  // identifier
  string subjectUniqueIdentifier; // optional subject unique
                                  // identifier
  string extensions;              // optional extensions
  algorithmIdentifierStruct certificateSignatureAlgorithm;
                                  // specifies the algorithm
                                  // used by the issuer to sign the
                                  // certificate
  string certificateSignature;    // signature of the certificate
}

struct keyStruct {                // structure for X.509 private key
  string privateKey;
}
```

6.4 certCreate

Notional Syntax

```
boolean certCreate (
     string certificateFile
     )
```

Description

CertCreate creates the file *certificateFile* that shall contain a certificate that conforms with ITU-T X.509.

Returns

CertCreate returns TRUE if the certificate was successfully created and FALSE otherwise.

6.5 certSignCode

Notional Syntax

```
boolean certSignCode (
      certStruct myCertificate,
      keyStruct myPrivateKey,
      string sourceFilename,
      unsigned int signatureAlgorithm,
      string signFilename,
      boolean overwriteCurrentSignatureFile
      )
```

Description

CertSignCode generates a digital signature (encrypted hash) of the source code file *sourceFilename* using public certificate *myCertificate* and private key *myPrivateKey*. The default hashing algorithm for signing shall be SHA-1. Alternative hashing functions that are specified in ISO/IEC 10118-3:2004 may be used instead and would be indicated through the value contained in *signatureAlgorithm*. The digital signature and publisher's certificate are stored in the file *signFilename*. If permitted by the implementation, the extension for the filename shall be "**.**ds". If *signFilename* already exists, then *overwriteCurrentSignatureFile* must be set to TRUE or *certSignCode* will return an error that the file could not be created since it already exists.

Returns

CertSignCode returns TRUE if the digital signature was successfully created and FALSE otherwise.

6.6 certSignWrap

Notional Syntax

```
boolean certSignWrap (
      certStruct myCertificate,
      keyStruct myPrivateKey,
      string originalSourceFilename,
      string modifiedSourceFilename,
      unsigned int signatureAlgorithm,
      string signFilename
      )
```

Description

Incorporates changes to the previously signed file *originalSourceFilename* in such a way that the changes can be unwrapped at a later date in order to revert to a previously signed version. *CertSignWrap* generates a digital signature (encrypted hash) of the source code file *modifiedSourceFilename* using public certificate *myCertificate* and private key *myPrivateKey*. The default hashing algorithm for signing shall be SHA-1. Alternative hashing functions that are specified in ISO/IEC 10118:2004 may be used instead and would be indicated through the value contained in *signatureAlgorithm*. The digital signature, publisher's certificate and changes between the current version and the previous version are appended to the end of the file *signFilename*.

Returns

CertSignWrap returns TRUE if the signature was successfully created and FALSE otherwise.

6.7 certHash

Notional Syntax

```
boolean certHash (
      string sourceFilename,
      unsigned int signatureAlgorithm
      )
```

Description

CertHash generates a digital finger print (hash) of the source code contained in file *sourceFilename*. The default hashing algorithm for signing shall be SHA-1. Alternative hashing functions that are specified in ISO/IEC 10118:2004 may be used instead and would be indicated through the value contained in *signatureAlgorithm*.

Returns

CertHash returns TRUE if the hash was successfully generated and FALSE otherwise.

6.8 certDecryptSignature

Notional Syntax

```
boolean certDecryptSignature (
      certStruct myCertificate,
      keyStruct myPrivateKey,
      string signFilename
      )
```

Description

CertDecryptSignature decrypts the digital signature of the source code file contained in *signFilename* using *myCertificate* and *myPrivateKey*.

Returns

CertDecryptSignature returns TRUE if the digital signature was successfully decrypted and FALSE otherwise.

6.9 certVerifySignature

Notional Syntax

```
boolean certVerifySignature (
      certStruct myCertificate,
      keyStruct myPrivateKey,
      string signFilename
      )
```

Description

CertVerifySIgnature verifies that the most recent digital signature of the source code file *signFilename* is valid and returns either an indication that the "signature is valid" or that the "signature is not valid". This accomplishes in one step what *certHash()* and *certDecryptSignature()* do in multiple steps. Note that the hashing algorithm is inferred by the length of the signed hash and thus need not be specified by the user.

Returns
CertVerifySignature returns TRUE if the signature is valid and FALSE otherwise.

6.10 certUnwrap

Notional Syntax

```
boolean certUnwrap (
      string signFilename,
      string sourceFilename
      )
```

Description

CertUnwrap reverts the previously signed file *signFilename* to the last previously signed version. *CertUnwrap* will remove the most recent signature from *signFilename* and the most recent set of changes in order to revert to the next most recent signature and source code. If *newSignFilename* is non-Null, then the unwrapped file contents are placed in *newSignFilename*.
After the operation is complete, *certVerifySignature* can be run to ensure the file they are viewing is the previous version of source code and has a valid signature.

Returns

CertUnwrap returns TRUE if the unwrapping was successful and FALSE otherwise.

Annex A
(*informative*)
A possible method of operation

This annex describes one possible way of using the interfaces specified in Clause 6 of this International Standard.

1. Publisher obtains a Code Signing Digital ID (Software Publishing Certificate) from a global certificate authority

The acquisition of a Code Signing Digital ID is outside the scope of this international standard. Once the Code Signing Digital ID has been acquired, the publisher uses `certCreate` to create a certificate file.

2. Publisher develops code or modifies previously signed code

This international standard places no requirements on the activities of code development.

3. Calculate a hash of the code and create a new file containing the encrypted hash, the publisher's certificate and the code

There are possible cases. The first is signing code which does not have a signature. The second is signing code received from others and which already contains a signature.

For new code that does not have a signature, a one-way hash of the code is produced using *`certsigncode`*, thereby signing the code.

For previously signed code, a one-way hash of the code is produced using *`certSignWrap`* to sign the code.

The hash and publisher's certificate are appended to the beginning of the file containing the code. Therefore, for previously signed code, the newest signature will appear first in the file containing the code.

4. The digitally signed file is transmitted to the recipient

The means of transmission of the digitally signed file is beyond the scope of this international standard.

5. The recipient produces a one-way hash of the code

`CertHash` is used to produce a one-way hash of the code.

6. Using the publisher's public key contained within the publisher's Digital ID and the digital signature algorithm, the recipient browser decrypts the signed hash with the sender's public key

CertDecryptSignature is used to decrypt the signed hash with the sender's public key. Steps 5 and 6 can be accomplished in one step using *certVerifySignature*.

7. The recipient compares the two hashes

If the signed hash matches the recipient's hash, the signature is valid and the document is intact and hasn't been altered since it was signed.

Software that has multiple signings must be able to be "unwrapped" using *certUnwrap* in order to recreate previously signed versions.

Bibliography

1. *Code-Signing Best Practices*, http://msdn.microsoft.com/en-us/windows/hardware/gg487309.aspxJuly 25, 2007

2. *Code Signing Certificate FAQ*, http://www.verisign.com/code-signing/information-center/certificates-faq/index.html, 2011

3. *Code Signing for Developers - An Authenticode How-To*, Tech-Pro.net, http://www.tech-pro.net/code-signing-for-developers.html, 2011.

4. Oliver Goldman, *Code Signing in Adobe AIR*, Dr. Dobb's, September 1, 2008.

5. *How Code Signing Works,* https://www.verisign.com/code-signing/information-center/how-code-signing-works/index.html, 2011.

6. *Introduction to Code Signing*, http://msdn.microsoft.com/en-us/library/ms537361(VS.85).aspx, June 21, 2011.

7. ISO/IEC 10118-3:2004, Information technology -- Security techniques -- Hash-functions -- Part 3: Dedicated hash-functions.

8. ISO/IEC 14750:1999, Information technology -- Open Distributed Processing -- Interface Definition Language, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=25486.

9. ITU-T Recommendation X.509:2008, Information Technology - Open Systems Interconnection - The Directory: Authentication Framework, http://www.itu.int/rec/T-REC-X.509/en.

10. Steve Mansfield-Devine, *A Matter of Trust*, Network Security, Vol 2009, Issue 6, June 2009.

11. Regina Gehne, Chris Jesshope, Jenny Zhang, *Technology Integrated Learning Environment: A Web-based Distance Learning System*, AI-ED'95, 7th World Conference on Artificial Intelligence in Education, 2001.

12. Justin Samuel, Nick Mathewson, Justin Cappos, and Roger Dingledine, *Survivable Key Compromise in Software Update Systems*, The 17th ACM Conference on Computer and Communications Security, 2010.

13. Deb Shinder, *Code Signing: Is it a Security Feature?*, WindowSecurity.com, http://www.windowsecurity.com/articles/Code-Signing.html?printversion , June 9, 2005.

