ISO/IEC JTC 1/SC 22/WG 23 N 0379

Submitted New Work Item Proposal and Preliminary Working Draft for Code Signing

Date	12 December 2011
Contributed by	SC 22 Secretariat
Original file name	ISO-IECJTC1-SC22_N4698_Information_technology--Programming_l.pdf
Notes	This is the New Work Item Proposal and attached preliminary working draft being balloted in SC22.

ISO/IEC JTC 1/SC 22

Programming languages, their environments and system software interfaces

Secretariat: ANSI

Document type:	Text for NP ballot
Title:	Information technology--Programming languages, their environments and system software interfaces--Code signing for source code
Status:	Please vote using the Balloting Portal.
Date of document:	2011-11-30
Source:	SC 22/WG 23
Expected action:	VOTE
Action due date:	2012-03-01
Email of secretary:	mpeacock@ansi.org
Committee URL:	http://isotc.iso.org/livelink/livelink/open/jtc1sc22

WG 23 Document: N0377

G3 New Work Item Proposal

March 2007

[Form downloaded from the JTC 1 Templates web site, 5 November 2011]

PROPOSAL FOR A NEW WORK ITEM

Date of presentation of proposal: [YYYY-MM-DD]	Proposer: ISO/IEC JTC 1/SC 22/WG 23
Secretariat: ANSI (United States)	**ISO/IEC JTC 1 N** [XXXX] ISO/IEC JTC 1/SC 22 N [XXX]

A proposal for a new work item shall be submitted to the secretariat of the ISO/IEC joint technical committee concerned with a copy to the ISO Central Secretariat.

Presentation of the proposal - to be completed by the proposer.

Title Information technology--Programming languages, their environments and system software interfaces--Code signing for source code
Scope This International Standard uses a language 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 application developers and systems implementers. The following areas are outside the scope of this specification: • Graphics interfaces • Object or binary code portability • System configuration and resource availability
Purpose and justification - The extended supply chains used in modern software development make it very difficult to ascertain the origin of source code and to ensure that no intentional or unintentional modifications were made to the code. Code signing applies the existing technology of digital signatures so that receivers' of source code can be assured that the received code is identical to the code signed by the originator of the code. Traceability to the originator cannot guarantee that the code is correct, but it can guarantee that the code being used is the same as the code that was tested by its developer.
Programme of work If the proposed new work item is approved, which of the following document(s) is (are) expected to be developed? __x__ a single International Standard ___ more than one International Standard (expected number: .....) _____ a multi-part International Standard consisting of ..... parts _____ an amendment or amendments to the following International Standard(s) ..... _____ a technical report , type .....

And which standard development track is recommended for the approved new work item?

_x___a. Default Timeframe
____b. Accelerated Timeframe
____c. Extended Timeframe

Relevant documents to be considered

- ISO/IEC 14750:1999 Information technology--Open distributed processing--Interface definition language: This document will be considered as the source of a notation for describing the APIs.
- ITU-T Recommendation X.509 (2008), Information technology--Open systems interconnection--The directory: Authentication framework
- Open literature concerning code signing and digital signature technology
- The programming language standards of JTC 1/SC 22 and the IT security standards of JTC 1/SC 27

Co-operation and liaison

- Liaison with ISO/IEC JTC 1/SC 27 (IT Security Techniques), will be pursued with the hope of applying available specifications and expertise from the IT security standards.
- Liaison with ISO/IEC JTC 1/SC 7/WG 21 (Software Asset Management), will be pursued with the hope of applying technology for "software asset tags".

Preparatory work offered with target date(s)

A preliminary working draft is circulated with this New Work Item Proposal

Signature:

Will the service of a maintenance agency or registration authority be required? .....No.....
- If yes, have you identified a potential candidate? .....
- If yes, indicate name .....

Are there any known requirements for coding? ....No.....
-If yes, please specify on a separate page

Does the proposed standard concern known patented items? .....No.....
- If yes, please provide full information in an annex

Are there any known accessibility requirements and/or dependencies (see: http://www.jtc1access.org)?.....No.....

-If yes, please specify on a separate page

Are there any known requirements for cultural and linguistic adaptability?.....No.....

-If yes, please specify on a separate page

Comments and recommendations of the JTC 1 or SC XXSecretariat - attach a separate page as an

annex, if necessary

Comments with respect to the proposal in general, and recommendations thereon:
It is proposed to assign this new item to JTC 1/SC 22/WG 23

Voting on the proposal - Each P-member of the ISO/IEC joint technical committee has an obligation to vote within the time limits laid down (normally three months after the date of circulation).

Date of circulation: [YYYY-MM-DD]	Closing date for voting: [YYYY-MM-DD]	Signature of Secretary:

NEW WORK ITEM PROPOSAL - PROJECT ACCEPTANCE CRITERIA		
Criterion	**Validity**	**Explanation**
A. Business Requirement		
A.1 Market Requirement	Essential _x__ Desirable ___ Supportive ___	Security and safety of software is an increasingly important problem. The ability to assure that the code has not been altered supports accountability for security in supply chains.
B. Related Work		
B.1 Completion/Maintenance of current standards	Yes ___ No__x_	
B.2 Commitment to other organisation	Yes ___ No__x_	
B.3 Other Source of standards	Yes ___ No_x__	
C. Technical Status		
C.1 Mature Technology	Yes _x__ No___	The underlying technology is mature. The application of the technology in this context is not yet mature.
C.2 Prospective Technology	Yes _x__ No___	See above.
C.3 Models/Tools	Yes ___ No__x_	
D. Conformity Assessment and Interoperability		
D.1 Conformity Assessment	Yes ___	

	No_x__	
D.2 Interoperability	Yes ___ No_x__	
E. Adaptability to Culture, Language, Human Functioning and Context of Use		
E.1 Cultural and Linguistic Adaptability	Yes ___ No_x__	We believe that the existing technology for digital signatures already supports cultural and linguistic adaptability.
E.2 Adaptability to Human Functioning and Context of Use	Yes ___ No_x__	
F. Other Justification		

Notes to Proforma

A. Business Relevance. That which identifies market place relevance in terms of what problem is being solved and or need being addressed.

A.1 Market Requirement. When submitting a NP, the proposer shall identify the nature of the Market Requirement, assessing the extent to which it is essential, desirable or merely supportive of some other project.

A.2 Technical Regulation. If a Regulatory requirement is deemed to exist - e.g. for an area of public concern e.g. Information Security, Data protection, potentially leading to regulatory/public interest action based on the use of this voluntary international standard - the proposer shall identify this here.

B. Related Work. Aspects of the relationship of this NP to other areas of standardisation work shall be identified in this section.

B.1 Competition/Maintenance. If this NP is concerned with completing or maintaining existing standards, those concerned shall be identified here.

B.2 External Commitment. Groups, bodies, or for a external to JTC 1 to which a commitment has been made by JTC for Co-operation and or collaboration on this NP shall be identified here.

B.3 External Std/Specification. If other activities creating standards or specifications in this topic area are known to exist or be planned, and which might be available to JTC 1 as PAS, they shall be identified here.

C. Technical Status. The proposer shall indicate here an assessment of the extent to which the proposed standard is supported by current technology.

C.1 Mature Technology. Indicate here the extent to which the technology is reasonably stable and ripe for standardisation.

C.2 Prospective Technology. If the NP is anticipatory in nature based on expected or forecasted need, this shall be indicated here.

C.3 Models/Tools. If the NP relates to the creation of supportive reference models or tools, this shall be indicated here.

D. Conformity Assessment and Interoperability Any other aspects of background information justifying this NP shall be indicated here.

D.1 Indicate here if Conformity Assessment is relevant to your project. If so, indicate how it is addressed in your project plan.

D.2 Indicate here if Interoperability is relevant to your project. If so, indicate how it is addressed in your project plan

E. Adaptability to Culture, Language, Human Functioning and Context of Use
NOTE: The following criteria do not mandate any feature for adaptability to culture, language, human functioning or context of use. The following criteria require that if any features are provided for adapting to culture, language, human functioning or context of use by the new Work Item proposal, then the proposer is required to identify these features.

E.1 Cultural and Linguistic Adaptability. Indicate here if cultural and natural language adaptability is applicable to your project. If so, indicate how it is addressed in your project plan.

ISO/IEC TR 19764 (Guidelines, methodology, and reference criteria for cultural and linguistic adaptability in information technology products) now defines it in a simplified way:

- "ability for a product, while keeping its portability and interoperability properties, to:
- be internationalized, that is, be adapted to the special characteristics of natural languages and the commonly accepted rules for their se, or of cultures in a given geographical region;
- take into account the usual needs of any category of users, with the exception of specific needs related to physical constraints

Examples of characteristics of natural languages are: national characters and associated elements (such as hyphens, dashes, and punctuation marks), writing systems, correct transformation of characters, dates and measures, sorting and searching rules, coding of national entities (such as country and currency codes), presentation of telephone numbers and keyboard layouts. Related terms are localization, jurisdiction and multilingualism.

E.2 Adaptability to Human Functioning and Context of Use. Indicate here whether the proposed standard takes into account diverse human functioning and diverse contexts of use. If so, indicate how it is addressed in your project plan.
NOTE:
1. Human functioning is defined by the World Health Organization at http://www3.who.int/icf/beginners/bg.pdf as: << In ICF (International Classification of Functioning, Disability and Health), the term functioning refers to all body functions, activities and participation. >>
2. Content of use is defined in ISO 9241-11:1998 (Ergonomic requirements for office work with visual display terminals (VDTs) Part 11: Guidance on usability) as: << Users, tasks, equipment (hardware, software and materials), and the physical and societal environments in which a product is used.>>
3. Guidance for Standard Developers to address the needs of older persons and persons with disabilities).

F. Other Justification Any other aspects of background information justifying this NP shall be indicated here.

ISO/IEC JTC 1/SC 22/WG 23 N 0359

Revised preliminary working draft, "Code Signing for Source Code"

Date	9 September 2011
Contributed by	Larry Wagoner
Original file name	Prelim_WD_code_signing_090811.doc
Notes	Replaces N0357

The following is a preliminary working draft related to a New Work Item Proposal which has not yet been approved. It is offered as an illustration of what the proposed project might produce.

Strawman INTERNATIONAL STANDARD

ISO/IEC xxxxx

Information technology—Programming languages, their environments and system software interfaces—Code signing for source code

1. Scope

This document uses a language 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 systems 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 14750:1999, Information technology -- Open Distributed Processing -- Interface Definition Language

3. Terms and Definitions

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

[TBD]

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 the process of digitally signing scripts and executable objects that verifies the author or origin and guarantees that the signed code has not been tampered with or corrupted since it was signed by use of a cryptographic hash.

Code signing provides several valuable functions,

- code signing can provide security when deploying,
- code signing can provide a digital signature mechanism to verify the identity of the author or build system,
- code signing can provide multi signatures, allowing an audit trail of the signed object,
- code signing will provide a checksum to verify that the object has not been modified,
- code signing can provide versioning information, and
- code signing can store other meta data about an object.

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 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. This 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 (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 written and signed by someone known and trusted allows the software to be used with confidence.

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 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 independent of any particular programming language, using the Interface Description Language (IDL) provided by ISO/IEC 14750:1999.

Note: 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 within the language implementation.

6.2 Structures

Additional descriptions of the fields used in these structures are available at 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;                          // a representation of its issuer's identity in
                                                    // the form of a Distinguished Name
```

```
        string int validNotBeforeDate;              // the start of the time period in which a
                                                    // certificate is intended to be used
        string 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;          // the public key algorithm to be used with
                                                    // the subjectPublicKey
        string subjectPublicKey;                    // the 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 a X.509 private key
        string privateKey;
}
```

6.3 certCreate

Notional Syntax

boolean certCreate (string certificateFile, string certificateDirPath)

Description

CertCreate creates in the directory *certificateDirPath* the file *certificateFile* that contains a certificate that complies with ITU-T X.509.

Returns

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

Errors

If the *certificateFile* cannot be created, *CertCreate* will report an error.

If *certificateDirPath* is an invalid path, *CertCreate* will report an error.

6.4 certSignCode

Notional Syntax

boolean certSignCode (certStruct myCertificate, keyStruct myPrivateKey, string sourceFilename, string sourceDirPath, boolean overwriteCurrentSignature, enum hashType signatureAlgorithm, string signFilename, string signDirPath)

Description

CertSignCode generates a digital signature (encrypted hash) of the source code file *sourceFilename* in directory *sourceDirPath* 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 could be used instead and would be indicated through the enumerated type *signatureAlgorithm*. The digital signature and publisher's certificate are stored in the directory *signDirPath* in the file *signFilename*. By convention, the signature filename *signFilename* should be of the form "filename.ds". If *signFilename* already exists in the directory *signDirPath*, then *overwrite* 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.

Errors

If *signFilename* exists and *overwrite* is FALSE, *certSignCode* will report that the signature operation could not be completed since sign*Filename* already exists.

If *myCertificate* or *myPrivateKey* are in an unknown format or do not contain proper keys, *certSignCode* will report that the signature operation could not be completed since a key could not be read or used.

6.5 certSignWrap

Notional Syntax

boolean certSignWrap (certStruct myCertificate, keyStruct myPrivateKey, string originalSourceFilename, string originalSourceDirPath, string modifiedSourceFilename, string modifiedSourceDirPath, enum hashType signatureAlgorithm, string signFilename, string signDirPath)

Description

Incorporates changes to the previously signed file *originalSourceFilename* in directory *originalSourceDirPath* 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* in directory modifiedSourceDirPath 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 could be used instead and would be indicated through the enumerated type *signatureAlgorithm*. The digital signature, publisher's certificate and changes between the current version and the previous version are added to the file *signFilename* in directory *signDirPath*.

Returns

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

Errors

If a signature for *originalSourceFilename* does not exist, *certSignWrap* will report that the signature wrapping could not be completed because a signature does not exist and that a signature file would need to be created before the operation could be completed.

If there are no differences between the contents of *originalSourceFilename* and *modifiedSourceFilename*, *certWrap* will report that the signature operation could not be completed since there have not been any changes to the source code file.

If the hash of *originalSourceFilename* does not match the encrypted hash stored within *originalFile.ds*, *certSignWrap* will report that the *originalFile* differs from the file which was signed and that the signature operation could not be completed.

6.6 certHash

Notional Syntax

boolean certHash (string sourceFilename, string sourceDirPath, enum hashType signatureAlgorithm)

Description

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

Returns

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

Errors

TBD

6.7 certDecryptSignature

Notional Syntax

boolean certdecryptsignature (certStruct myCertificate, keyStruct myPrivateKey, string signFilename, string signDirPath)

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.

Errors

If the signature file *signFilename* does not exist, *certDecryptSignature* will report that the signature could not be verified because the signature file is missing.

If the signature file exists yet does not contain the properly formatted signature and public key components, *certDecryptSignature* will report that the signature file is corrupt.

6.8 certVerifySignature

Notional Syntax

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

Description

CertVerifySIgnature verifies the latest digital signature of the source code file sign*Filename* in directory *signDirPath* is valid and returns either an indication that the "signature is valid" or "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.

Errors

If the signature file does not exist, *certVerifySignature* will report that the signature file is missing.

If the signature file exists but does not contain the properly formatted signature and public key components, *certVerifySignature* will report that the signature file is corrupt.

6.9 certUnwrap

Notional Syntax

boolean certUnwrap (string signatureFile, string signatureFileDirPath, string sourceFilename, string sourceDirPath, string newSignatureFile, string newSignatureDirPath, string newSourceFilename, string newSourceDirPath)

Description

CertUnwrap reverts a previously signed file to the last previously signed version. *CertUnwrap* will remove the most recent signature for *sourceFilename* in *sourceDirPath* from the file *signatureFile* in directory *signatureFileDirPath* and the most recent set of changes in order to revert to the next most recent signature and file. If *newSignatureFile* and newSignatureFileDirPath are non-NULL, certUnwrap places modified the signature file in *newSignatureFile* inside directory newSignatureDirPath instead of modifying the contents of *signatureFile*. If *sourceFilename* and sourceDirPath non-Null, then the unwrapped file contents are placed in *sourceFilename* in *sourceDirPath*.

After the operation is complete, the user should run *certverifysignature* to ensure the files 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.

Errors

If the signature file does not contain a valid signature or is missing any components such as certificates or file differences, *cerUnwrap* will report that the unwrap operation could not be completed.

If only one of *newSignatureFile* and *newSignatureFileDirPath* is NULL, an error is generated.

If only one of *sourceFilename* and *sourceDirPath* is NULL, an error is generated.

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**

   (how one obtains a Code Signing Digital ID may be out of scope and might be better left to other standards bodies such as the World Wide Web Consortium (W3C))

   A software publisher's request for certification is sent to the Certification Authority (CA). It is expected that the CAs will have Web sites that walk the applicant through the application process. Applicants will be able to look at the entire policy and practices statements of the CA. The utilities that an applicant needs to generate signatures should also be available.

   Digital IDs can be either issued to a company or an individual. In either case, the global certificate authority must validate the identification of the company and applicant. Validation for applicants would be in the form of a federally issued identification for applicants and a Dun & Bradstreet number. Tables 1 and 2, respectively, contain the criteria for a commercial and individual code signer.

   Proof of identification of an applicant must be made. Simply trusting the applicant's ID via a web site is insufficient. Additional verification of the applicant's ID should be commensurate with the application process for a federally issued ID, such as a passport. Sending in a federally issued ID, such as a passport, to the CA would be sufficient for proof of identification.

   The applicant must generate a key pair using either hardware or software encryption technology. The public key is sent to the CA during the application process. Due to the identity requirements, the private key must be sent by mail or courier to the applicant.

Identification	Applicants must submit their name, address, and other material along with a copy of their federally issued id that proves their identity as corporate representatives. Proof of identify requires either personal presence or registered credentials.

Agreement	Applicants must agree to not distribute software that they know, or should have known, contains viruses or would otherwise harm a user's computer or code.
Dun & Bradstreet Rating	Applicants must achieve a level of financial standing as indicated by a D-U-N-S number (which indicates a company's financial stability) and any additional information provided by this service. This rating identifies the applicant as a corporation that is still in business. (Other financial rating services are being investigated.) Corporations that do not have a D-U-N-S number at the time of application (usually because of recent incorporation) can apply for one and expect a response in less than two weeks.

312 **Table 1: Criteria for Commercial Code Publishing Certificate**

313

Identification	Applicants must submit their name, address, and other material along with a copy of their federally issued id that proves their identity as citizens of the country where they reside. Information provided will be checked against an independent authority to validate their credentials.
Agreement	Applicants must agree that they cannot and will not distribute software that they know, or should have known contains viruses or would otherwise maliciously harm the user's computer or code.

314 **Table 2: Criteria for Individual Code Publishing Certificate**

315

316 **2. Publisher develops code or modifies previously signed code**

317

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

A one-way hash of the code is produced using *certsigncode*, thereby signing the code. The hash and publisher's certificate are inserted stored in a separate file.

In order to be able to verify the integrity of previously signed code, it must be possible to identify the responsible party for each section of code. When new code modifies or in some way encapsulates previously signed code, the original code must be able to be identified so that its signature can be checked. Therefore, iterative changes to code must be able to be reversed to identify previously signed versions.

4. **The digitally signed file is transmitted to the recipient**

5. **The recipient produces 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**

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" in order to recreate previously signed versions. Iterative changes to code can be reversed to identify previously signed versions through the use of *certunwrap*.

Bibliography

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

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

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

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

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

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

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.

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.

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

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.

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.

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