ISO/IEC JTC 1/SC 22/WG 23 N 0317

Strawman draft, "Code Signing for Source Code"

Date	2011-03-23
Contributed by	Larry Wagoner
Original file name	Code Signing Strawman_120110.doc
Notes	

Code Signing for Source Code

1 Introduction

Code Signing is a capability that identifies to customers the company responsible 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 and 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 does on packaged software. If an organization or individual wants to use the Internet to distribute software, they should be willing to take responsibility for that software. 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 that it is legitimate.

Multiple signatures for one piece of code would be needed in some cases so as to create a digital trail through the origins of the code. Consider a signed piece of code. Someone should be able to modify some portion of the code, even one character, without assuming responsibility for the integrity of the remainder of the code. Similarly, 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 their driver for a particular use. Company B is not as trusted or has an unknown reputation. The recipient should be able to determine what part of the code originated with and was unaltered by Company B so as to be able to concentrate their evaluation on the sections of code that Company B either added or altered. Therefore, a means is needed to keep track of the modifications made from one signature to the next. Each signature would create another layer on top of the preceding one.

1.1 Scope

This document defines the utility programs 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. Terminology

3. APIs

certcreate

Description

creates the file outCerFile that will contain a certificate that complies with ITU-T X.509.

Syntax

certcreate [options] outputCertificateFile

Options

TBD

Errors

TBD

Examples

createcert certfile
will create the file certfile containing a certificate

certsigncode

Description

Generates a digital signature (encrypted hash) of the source code file *filename* using public certificate *myCertificate* and private key *myPrivateKey*. The default hashing algorithm for signing shall be MD5. Alternatively SHA1 could be specified with the –*s* option. The digital signature and publisher's certificate are stored in file *filename*.ds unless otherwise specified with the –o option.

Syntax

certsigncode [options] myCertificate myPrivateKey filename

Options

-*n* overwrite the current signature with a new signature

-*o filename* put signature in filename instead of the default output filename

-*s* Use SHA1 hash instead of the default MD5

TBD

Errors

If *filename*.ds or the file specified with the –o option already exists, certsigncode will report that the signature operation could not be completed since *filename*.ds or the specified file already exists and that the –n operation should be used.

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.

Example

certsigncode myCertificate.cer myPrivateKey.pvk sourceCode.c
will create the file sourcecode.c.ds containing the digital signature, an encrypted hash computed using the MD5 algorithm, and the public key.
certsigncode –n myCertificate.cer myPrivateKey.pvk sourceCode.c
will overwrite the existing file sourceCode.c.ds with a file containing the digital signature and public key.
certsigncode –o signatureFile.ds myCertificate.cer myPrivateKey.pvk sourceCode.c
will create the file signatureFile.ds containing the digital signature and the public key.
certsigncode –s myCertificate.cer myPrivateKey.pvk sourceCode.c
will create the file sourceCode.c.ds containg the digital signature, an encrypted hash computed using SHA1 algorithm, and the public key.

certsignwrap

Description

Incorporates changes to a previously signed file in such a way that the changes can be unwrapped later on in order to revert to a previously signed version. Generates a digital signature (encrypted hash) of the source code file *filename* using public certificate *myCertificate* and private key *myPrivateKey*. The hashing algorithm for signing shall be MD5 by default, or optionally sha1. The digital signature, publisher's certificate and *diff* output are added to file *outputFile*.ds.

Syntax

certwrap [options] myCertificate myPrivateKey originalFile modifiedFile

Options

-*s* Use sha1 hash instead of the default MD5

-*o filename* Use filename as signature file instead of default originalFile.ds

TBD

Errors

If *originalFile.ds*, or a file specified by the –*o* option, does not exist, *certwrap* will report that the signature wrapping could not be completed because an existing 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 *originalFile* and *modifiedFile*, *certwrap* will report that the signature operation could not be complted since there have not been any changes to the source code files.

If the hash of *originalFile* does not match the encrypted hash stored within *originalFile.ds*, or a file specified by the –*o* option, *certwrap* will report that the *originalFile* differs from the file which was signed and that the signature operation could not be completed.

Example

certwrap myCertificate myPrivateKey file1.c file1_modified.c
will update the file *file1.c.ds* containing the signature of file *file1.c* and the changes necessary to create file1_modified.c
certwrap –s myCertificate myPrivateKey file1.c file1_modified.c
will update the file outputFile containing the signature of file file1.c, an encrypted hash computed using the SHA1 algorithm, and the changes necessary to create file1_modified.c
certwrap –o signatureFile myCertificate myPrivateKey file1.c file1_modified.c
will update the file *signatureFile* containing the signature of file *file1.c* and the changes necessary to create file1_modified.c as well as the signature of *file1_modified.c* and the public key from file *myCertificate*

certhash

Description

Generates a digital finger print (hash) of the source code. The algorithm for computing the hash shall be MD5 by default, or optionally sha1.

Syntax

certhash [options] filename

Options

-*s* -- use sha1 hash instead of the default MD5

TBD

Errors

If more or less than one filename is provided an error shall be signaled and *certhash* will report its proper usage.

Example

certhash sourceCode.c
will compute the hash of sourceCode.c using the MD5 algorithm
certhash –s sourceCode.c
will compute the hash of sourcecode.c using the SHA1 algorithm

certdecryptsignature

Description

Verifies the digital signature of a source code file and returns the decrypted signature.

Syntax

certdecryptsignature [options] filename

Options

-*s signatureFile* Use signature in *signatureFile* instead of default

Errors

If the signature file 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.

Example

certdecryptsignature sourceCode.c

will verify the digital signature contained in sourceCode.c.ds and return the hash decrypted using the public key contained within the signature file.

certdecryptsignature –s signatureFile sourceCode.c

will verify the digital signature contained in the specified signatureFile and return the hash decrypted using the public key contained within signatureFile

certverifysignature

Description

Verifies the latest digital signature of a source code file *filename* compares the hash computed for *filename* and returns either “signature valid” or “signature not valid”. This accomplishes in one step what certhash() and certdecryptsignature() do in multiple steps. Note the hashing algorithm is inferred by the length of the signed hash and thus need not be specified by the user.

Syntax

certverifysignature [options] filename

Options

-s *filename* -- use digital signature contained in file *filename* instead of the default filename

Errors

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

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

Example

certverifysignature sourceCode.c

will compare the signature contained in the file sourceCode.c.ds with hash of sourceCode.c

certverifysignature –s signatureFile.ds sourceCode.c

will compare the signature contained in the file signatureFile.ds with the hash of sourceCode.c

certunwrap

Description

Unwrap a previously signed file to revert to the last previously signed version. Certunwrap will remove the most recent signature from the filename.ds file and the most recent set of changes in order to revert to the next most recent signature and file.

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.

Syntax

certunwrap [options] modifiedFile

Options

-*n newSignatureFile* – places modified signature file in *newSignatuerFile* instead of modifying the one used to unwrap the changes

-*o newFileName* -- sets the name of the output file to "*newfilename*"

-*s signatureFile* -- uses *signatureFile* instead of the default filename

Errors

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

TBD

Example

certunwrap sourceCode.c

will unwrap *sourceCode.c.ds* as well as modify *sourceCode.c* to the previously signed source code file

certunwrap sourceCode.c –o modified_sourceCode.c

will unwrap *sourcecode.c.ds* as well as produce a modified copy of *sourceCode.c* in the file specified by the *–o* option

certunwrap sourceCode.c –o modified_sourceCode.c –n modified_signatureFile

will unwrap *sourcecode.c.ds* by placing the previous version of the signed file in the file specified by the *–n* option, and produce a modified copy of *sourceCode.c* in the file specified by the *–o* option

certunwrap sourceCode.c –o modified_sourceCode.c –n modified_signatureFile –s signedFile

will unwrap *signedFile,* the file specified by the *–s* option, by placing the previous version of the signed file in the file specified by the *–n* option, and produce a modified copy of *sourceCode.c* in the file specified by the *–o* option

Appendix 1:

A Proposed method of operation

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.

Table 1: Criteria for Commercial Code Publishing Certificate

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.

Table 2: Criteria for Individual Code Publishing Certificate

2. Publisher develops code or modifies previously signed code

3. **Calculate a hash of the code and create a new file containing the encrypted hash, the 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*.

Existing techniques currently in use to create and verify a digital signature

Already there exists several different code signing implementations. It would be a major advance to be able to start to unify these under one standard implementation.

- Microsoft® Authenticode®
  - Digitally sign .exe, .cab, .dll, .ocx, .msi, .xpi, and .xap files
  - Microsoft requires all files with the following extensions: exe, dll, ocx, sys, cpl, drv, scr to be signed with an Authenticode certificate to receive Windows Vista Logo Certification.
- Sun Java® (JavaSoft Developer Certificate)
  - Digitally sign .jar files for desktop and midlet mobile Java platforms
- Microsoft® Office and VBA
  (VBA Developer Certificate is identical to Authenticode certificates) (Digitally sign Microsoft VBA Macros for Microsoft Office)
- Adobe® AIR®
  - Digitally sign .air or .airi files for use in Adobe AIR
- Macromedia Shockwave®
  - Digitally sign files created with Macromedia Director 8 Shockwave Studio
- Authentic IDs for BREW®
  - BREW™: Binary Runtime Environment for Wireless
  - Digitally sign BREW applications

- Apple developer certificate
  Digitally sign extensions to be installed on the Safari web browser/platform

References

1. http://msdn.microsoft.com/en-us/library/ms537361(VS.85).aspx
2. https://www.verisign.com/code-signing/information-center/index.html
3. http://www.verisign.com/code-signing/information-center/certificates-faq/index.html
4. http://www.drdobbs.com/web-development/210004209;jsessionid=IFYXVK2HGN0WJQE1GHRSKH4ATMY32JVN?pgno=2
5. http://www.windowsecurity.com/articles/Code-Signing.html?printversion
6. http://www.tech-pro.net/code-signing-for-developers.html
7. http://www.microsoft.com/whdc/driver/install/drvsign/best-practices.mspx