Code Signing

Code signing is the process of digitally signing executables and scripts to confirm the identity of the software author and guarantee that the code has not been altered or corrupted since it was signed.

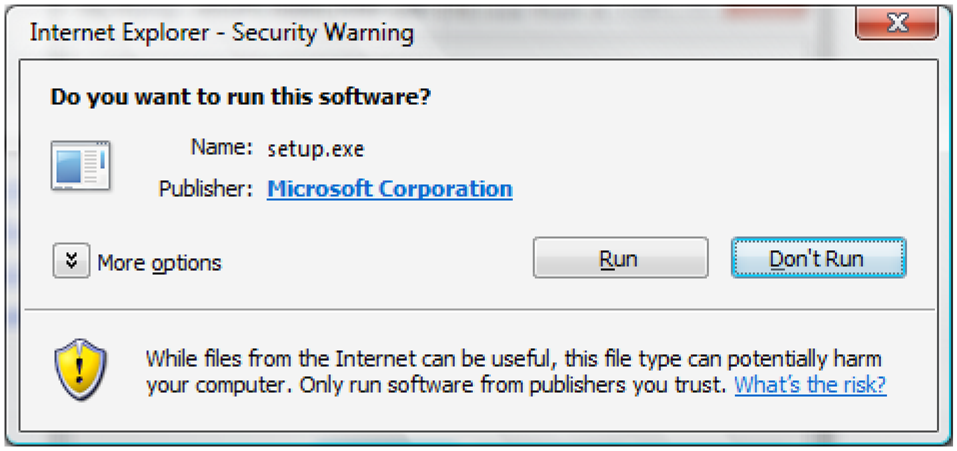
Publicly trusted certification authorities (CAs) confirm signers' identities and bind their public key to a code signing certificate.

This paper discusses how code signing works and the best practices to perform code signing.

Contents

Why Code Sign?

Most mass-market computing devices sold today come with pre-loaded software, but the software that comes with the device "out of the box" is never all that will be needed for the full life of the device. Whether for a personal computer or a mobile device, users will frequently run into situations where they need to download software or applications. In other cases, a user might not be planning to download software. Users are advised by an application on their device, or the site they are visiting, that in order to experience or use the offered they need to upgrade, patch or augment their current software. They are asked to make a spot decision: "Run" or "Don't Run."


Internet Explorer - Security Warning

Do you want to run this software?

Name: setup.exe

Publisher: Microsoft Corporation

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

In these situations, “run/don’t run” asks the user whether or not to run the downloaded code. How does a user decide? How does a user or user agent (usually a “browser”) know whether or not to trust the software? The answer is code signing.

To assist in the trust-decision process, the software publisher can digitally sign their code. The digital signature answers the questions of authentication and integrity, that is:

- Who signed the code?
- Has the code been tampered with since it was signed?

Armed with this information, the user can now make the “run/don’t run” decision.

Even though the digital signature does not answer whether you can “trust” the software not to harm your computer, unsigned code does not provide any evidence of origin or file integrity. The publisher is not identified and, therefore, cannot be held accountable. In addition, the code is subject to tampering. Digitally signed code, which is backed by a certificate issued by a CA acting as a trusted third party, is granted greater reliability than unsigned code, which should generally not be trusted.

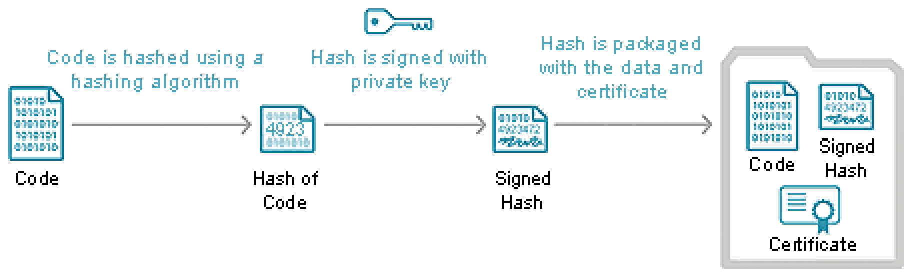
What is Code Signing?

From Wikipedia, “Code signing is the process of digitally signing executables and scripts to confirm the software author and guarantee that the code has not been altered or corrupted since it was signed by use of a cryptographic hash.”

In order to sign the code, the publisher needs to generate a private-public key pair and submit the public key to a CA along with a request to issue a code signing certificate. The CA verifies the identity of the publisher and authenticates the publisher’s digitally signed certificate request. If this vetting and key-verification process is successful, the CA bundles the identity of the publisher with the public key and signs the bundle, creating the code signing certificate.

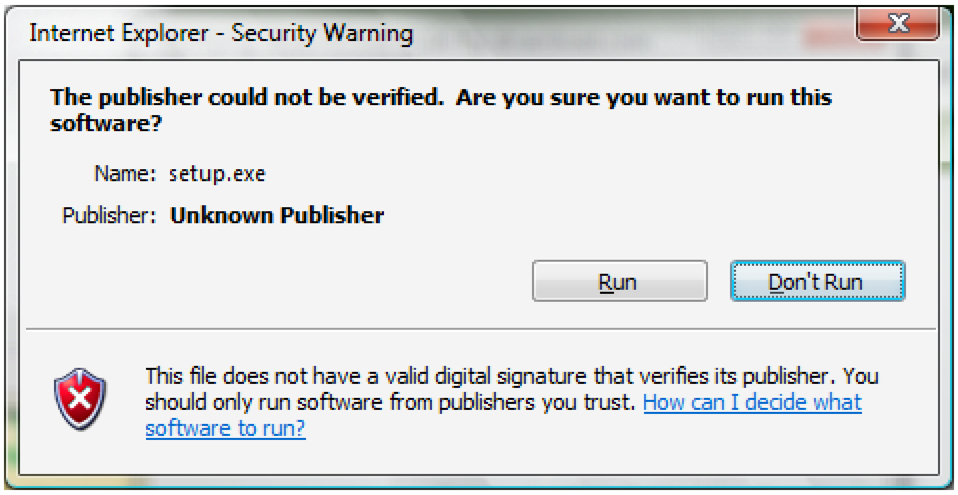
Armed with the code signing certificate, the publisher is ready to sign the code. When the code is signed, several pieces of information are added to the original file holding the executable code. This bundled information is used by the recipient's user agent to authenticate the publisher and check for code-tampering. The entire sequence for bundling the digitally signed code takes place as follows:

- **A hash of the code is produced**
    - Public-key algorithms are inefficient for signing large objects, so the code is passed through a hashing algorithm, creating a fixed-length digest of the file
    - The hash is a cryptographically unique representation of the file
    - The hash is only reproducible using the unaltered file and the hashing algorithm that was used to create the hash
- **The hash is signed using the publisher's private key**
    - The hash is passed through a signing algorithm using the publisher's private key as an input
    - Information about the publisher and the CA is drawn from the code signing certificate and incorporated into the signature
- **The original code, signature and code signing certificate are bundled together**
    - The code signing certificate key is added to the bundle (as the public key is required to authenticate the code when it is verified)


Code is hashed using a hashing algorithm
Hash is signed with private key
Hash is packaged with the data and certificate
Code
Hash of Code
Signed Hash
Code
Signed Hash
Certificate

Verifying Code Authenticity

When an end-user's user agent loads the code, it checks the authenticity of the software using the signer's public key, signature and the hash of the file. If the signature is verified successfully, the user agent accepts the code as valid. If the signature is not successfully verified, the user agent will react by warning the user or rejecting the code, according to the level of security being used.


Internet Explorer - Security Warning

The publisher could not be verified. Are you sure you want to run this software?

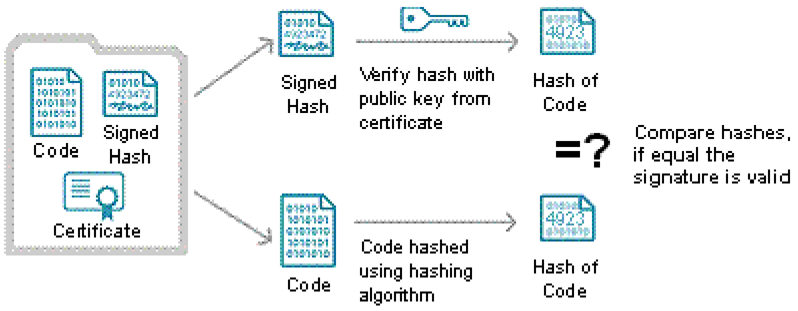
Name: setup.exe

Publisher: **Unknown Publisher**

This file does not have a valid digital signature that verifies its publisher. You should only run software from publishers you trust. How can I decide what software to run?

The signature is verified as follows:

- The original code is passed through the hashing algorithm to create a hash
- The public key of the publisher is extracted from the bundle and applied to the signature information; applying the public key reveals the hash that was calculated when the file was signed
- The two hashes are compared; if equal, then the code has not changed and the signature is considered valid
- The code signing certificate is checked to make sure that it was signed by a trusted CA
- The expiry date of the code signing certificate is checked
- The code signing certificate is checked against the revocation lists to be sure that it is valid
- If the file is considered valid, it is accepted by the user agent; if the file is not considered valid, the user agent displays a trust dialogue like the one above


Code
Signed Hash
Certificate
Signed Hash
Verify hash with public key from certificate
Hash of Code
=?
Compare hashes, if equal the signature is valid
Code
Code hashed using hashing algorithm
Hash of Code

How to Digitally Sign Code

Various application platforms support code signing and provide different tools to perform the signing. Here is a list of the more common code signing types and references as to where you can find guides for the given application.

Adobe AIR – Digitally signing an AIR file

Firefox XPI – Signing an XPI

Java – How to Sign Applets Using RSA-Signed Certificates and Signing Code and Granting it Permissions

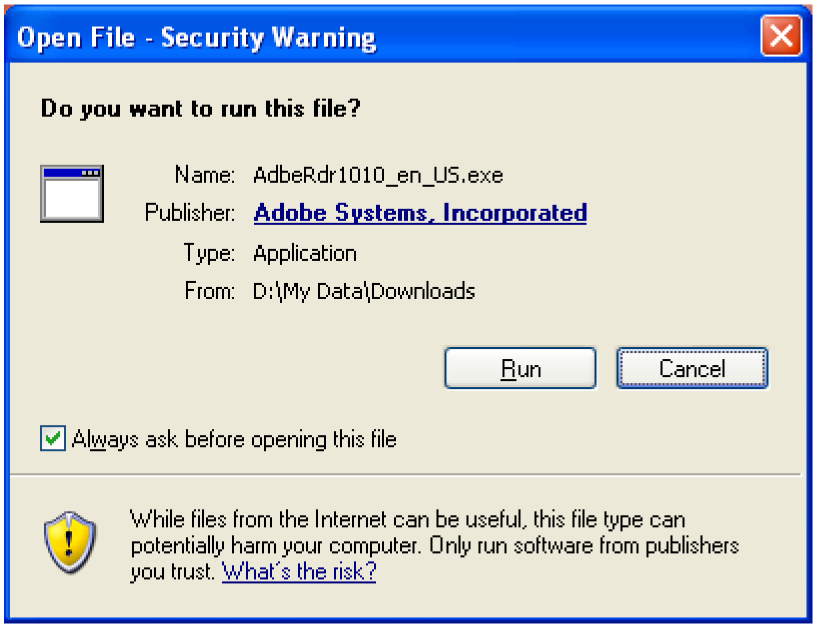
Microsoft Authenticode – Signing and Checking Code with Authenticode

Microsoft Windows Macro and Visual Basic Signing – Signing a VBA Project

Code Installation Trust Decision

The code has been signed, the user has started installation and verification has taken place. How does the user know whether or not to accept the code?

Here is a typical code verification security warning:


Open File - Security Warning

Do you want to run this file?

Name: AdbeRdr1010_en_US.exe

Publisher: Adobe Systems, Incorporated

Type: Application

From: D:\My Data\Downloads

✓ Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

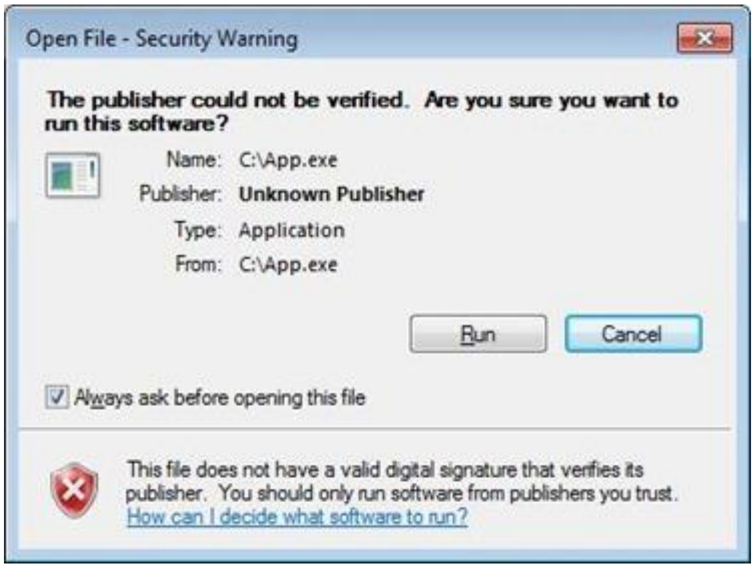
The user must make their trust decision based on the above. The statement provides the following:

1. **File Name:** In this case, it is "AdbeRdr1010_en_US.exe"
2. **Publisher Name:** Adobe Systems, Incorporated
3. **Code Signing Certificate:** The user would need to click on the publisher name

How to make the installation trust decision:

- Were you planning to install software? If so, proceed to the next step.
- Check the file name and see if it indicates the software you were planning to install. In this case, it is Adobe Reader 10, which the name seems to indicate.
- Check the publisher name and see if this matches who you think wrote the software. This may be hard as the software download site may be different than the publisher's site.
- Check the code signing certificate and see if the publisher's name is in the certificate. Also, the user can trust the certificate based on the issuing CA.

Here is a trust dialogue for code that you might not trust:


Open File - Security Warning

The publisher could not be verified. Are you sure you want to run this software?

Name: C:\App.exe
Publisher: **Unknown Publisher**
Type: Application
From: C:\App.exe

This file does not have a valid digital signature that verifies its publisher. You should only run software from publishers you trust.
How can I decide what software to run?

The file name is “App.exe,” which is not likely specific enough. The publisher’s name is “Unknown Publisher,” which means that a public CA did not verify the code signing certificate. The code may not be harmful, but it was likely signed with a self-issued code signing certificate. This means you cannot trust who signed the code. Thus, you should not trust the code.

What is Time-Stamping?

What happens to signed code when the code signing certificate expires? In many cases, an expired certificate means that the signature validation will fail and a trust warning will appear in the user agent.

Time-stamping was designed to alleviate this problem. The idea is that if you know the time when the code was signed and the certificate was confirmed to be valid, then you know the signature was valid at the time the software was published. This is much the same as a notarized handwritten signature where a third party is able to affix evidence of time.

The main benefit of time-stamping is that it extends code trust beyond the validity period of the certificate. The code stays good as long as you can run it. Also, the certificate may be revoked or expire in the future, but the code will still be trusted.

Time-stamping the signature is implemented as follows:

- The signature is sent to the time-stamping authority (TSA).
- The TSA adds a time-stamp to the bundled information and computes a new hash.
- The TSA signs the new hash with its private key creating a new bundle of information.
- The time-stamped bundle, original bundle (that was sent to the TSA) and the time-stamp are re-bundled with the original code.

Upon receipt of a time-stamped signature, the following is done for verification (in addition to verification of the signature on the code itself):

- The TSA certificate is checked to ensure it was issued from a trusted root certificate and that its status is valid
- The time-stamping authority's public key is applied to the time-stamped signature block, revealing the hash calculated by the TSA.
- The validity of the TSA's public key is verified by checking its expiry date and consulting the revocation lists to be sure that it has not been revoked.
- The two hashes are compared. If the hashes are equal, the time-stamp is considered to be valid.

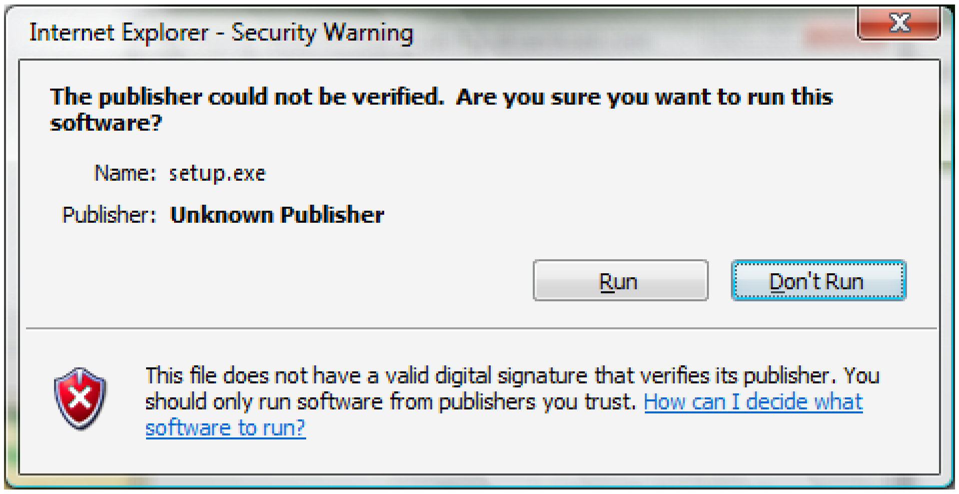
In the event that the code signing certificate must be revoked due to a compromise, the revocation will be made dependent on a specific date. The idea is that code signing, up until the compromise took place, was good (i.e., widely distributed and still in good working order). In other words, signatures with time stamps before the revocation date will remain valid and the software should still work.

Self-Signed Versus Publicly Trusted Code Signing Certificates

In most cases, you have to sign your code in order to get it installed on an operating system. You can sign your code using a self-signed certificate or using a certificate issued by a publicly trusted CA.

Due to the costs of buying a code signing certificate from a publicly trusted CA, some users might decide to try a self-signed certificate, but here is what you need to consider.

Self-Signed Certificate	**Certificates Issued from Publicly Trusted CA**
Issuer provides their own identity, which is not published in the trust dialogue	CA performs identity verification, which is displayed in the trust dialogue
Issuer provides their own policy and quality	CA issues certificates in accordance with the industry policy and quality
Signatures will provide a trust warning indicating that there was an un-verified publisher and will display "Unknown Publisher"	Signatures will provide positive trust dialogue
Compromised certificates cannot be revoked and could harm the users of your software	Compromised certificates can be revoked and if time-stamping was used, code signed before revocation will remain trusted


Internet Explorer - Security Warning

The publisher could not be verified. Are you sure you want to run this software?

Name: setup.exe

Publisher: **Unknown Publisher**

This file does not have a valid digital signature that verifies its publisher. You should only run software from publishers you trust. How can I decide what software to run?

For user trust and the longevity of your code, it is recommended that you use a certificate issued from a publicly trusted CA.

Code Signing: Best Practices

The biggest issue with code signing is the protection of the private signing key associated with the code signing certificate. If a key is compromised, the certificate loses trust and value, jeopardizing the software that you have already signed.

Here are some best practices for code signing:

1. **Minimize access to private keys**
    - Computers with keys should have minimal connections
    - Keep the users who have key access to a minimum
    - Use physical security to minimize access

2. **Protect private keys with cryptographic hardware products**
    - Keys should be protected by FIPS 140 Level 2-certified product (or better)
    - Cryptographic hardware does not allow export of the private key to software where it could be attacked

3. **Time-stamp your code**
    - Time-stamping allows your code to be verified after the certificate has expired or been revoked

4. **Test-signing versus release-signing**
    - Test-signing private keys and certificates require less security access controls than production code signing private keys and certificates
    - Test-signing certificates can be self-signed or come from an internal test CA
    - Establish a separate test code signing infrastructure to test-sign prerelease builds of software
    - Test certificates must chain to a completely different root certificate than the root certificate that is used to sign publicly released products; this precaution helps ensure that test certificates are trusted only within the intended test environment

5. **Authenticate code to be signed**
    - Any code that is submitted for signing should be strongly authenticated, so you know that it is permissible to sign and release

- Implement a code signing submission and approval process to prevent the signing of unapproved or malicious code
- Log all code signing activities for auditing and/or incident-response purposes

6. **Virus scan code before signing**
    - Code signing does not confirm the safety or quality of the code; it confirms the publisher and whether or not the code has been changed
    - Take care when incorporating code from other sources
    - Virus-scanning will help to improve the quality of the released code

7. **Do not over-use any one key (distribute risk with multiple certificates)**
    - If code is found with a security flaw, then you might want create a trust dialogue when the code is installed in the future; this can be done by revoking the code signing certificate and a revoked prompt will occur
    - If the code with the security flaw was issued before more good code was issued, then revoking the certificate will impact the good code as well
    - Changing keys and certificates often will help to avoid this conflict

Extended Validation (EV) Code Signing Certificates

EV code signing certificates have two distinct advantages over the normal issuance and management of code signing certificates.

First, the EV verification process of the identity and authorization of the publisher must be completed in accordance with the CA/Browser Forum EV Code Signing Guidelines. Second, the private keys to the certificates must be managed in hardware meeting the requirements of FIPS 140 Level 2 or equivalent.

The upside of EV code signing certificates is you know who the publisher is and reasonable protection has been provided to the private key to mitigate unauthorized signing. Since EV code signing certificates are more trusted, this allows developers of verification products to raise the reputation level of the publisher or the signed code.

For instance, Microsoft provides application reputation for EV-signed code through SmartScreen for IE9 and 10 and in Windows 8. A higher reputation will reduce the necessity to present a trust dialogue box to the user. They simply choose “Save” or “Run.”

Conclusion

Code signing is required to install code on many platforms because it provides assurances of authenticity and origin. When signing code you have to make a few decisions to protect your deployed software. The most important decision when establishing end-user trust is that the signed code is backed by a code signing certificate issued by a trusted Certification Authority. Self-signed certificates should only be used for testing, not for production releases.

The second most important step is to timestamp your code.   In the event of a compromised key, your time-stamp may ensure that your code is protected even if your certificate needs to be revoked.

The best practices section provides additional important tips for protecting the code signing private key and the quality of your signed code.

An Extended Validation Code Signing Certificate is the best tool available to establish trust in the security of the private key used to sign code and provides a higher assurance of the identity of the software publisher. Because EV code signing provides better information about the source of software, some platforms with malware security filters give EV-signed software better treatment in user dialog boxes during installation.

References

CA/Browser Forum EV Code Signing Guidelines, https://www.cabforum.org/documents.html

Microsoft Developer Network – Introduction to Code Signing, http://msdn.microsoft.com/en-us/library/ms537361.aspx

Microsoft Windows Code-Signing Best Practices, http://msdn.microsoft.com/en-us/windows/hardware/gg487309.aspx

Microsoft Technet- Deploying Authenticode with Cryptographic Hardware for Secure Software Publishing, http://technet.microsoft.com/en-us/library/cc700803.aspx

Microsoft Technet – Kill Bits, http://blogs.technet.com/b/srd/archive/2008/02/06/the-kill_2d00_bit-faq_3a00_-part-1-of-3.aspx

Microsoft SmartScreen and Extended Validation (EV) Code Signing Certificates, https://blogs.msdn.com/b/ie/archive/2012/08/14/microsoft-smartscreen-amp-extended-validation-ev-code-signing-certificates.aspx