CODE SIGNING

Why Developers Need to Digitally Sign Code and Applications

Table of contents

Introduction

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

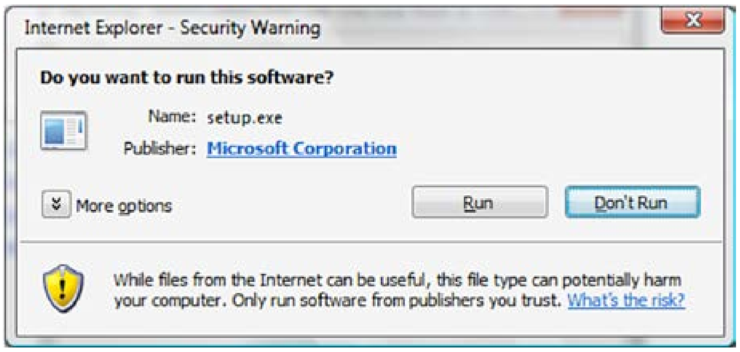
Publicly trusted certification authorities (CAs) confirm signers' identities and bind their public key to a code signing certificate. The certificate is used to support validation of code signatures to a trusted root certificate in widely distributed applications such as Windows or Java.

CAs and browsers have developed standards to manage and issue code signing certificates. The standards ensure applications are verified and code signing specifications meet the latest cryptographic requirements.

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

Why Code Sign?

Most mass-market computing devices sold today come with pre-loaded software, but the software that comes "out of the box" with the device is not all that will be needed for the full life of the device. Whether for a personal computer or a mobile device, users will frequently need to download additional software or applications. In other cases users are often advised by an application on their device, or the site they are visiting, that in order to experience or use the offered service they need to upgrade, patch or augment their current software. Users are asked to make a spot decision: "Run or Don't Run," "Install or Don't Install" or "Run or Cancel."


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 help users determine whether or not they can trust software before they install it, software publishers can digitally sign their code. A digital signature verifies who signed the code and that the code has not been 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. Generally, unsigned code should not be trusted, as it does not provide any evidence of origin or file integrity, which means the publisher cannot be held accountable for errors and the code is subject to tampering.

Armed with the information provided by a digital signature, users can make a more informed "Run/Don't Run" decision.

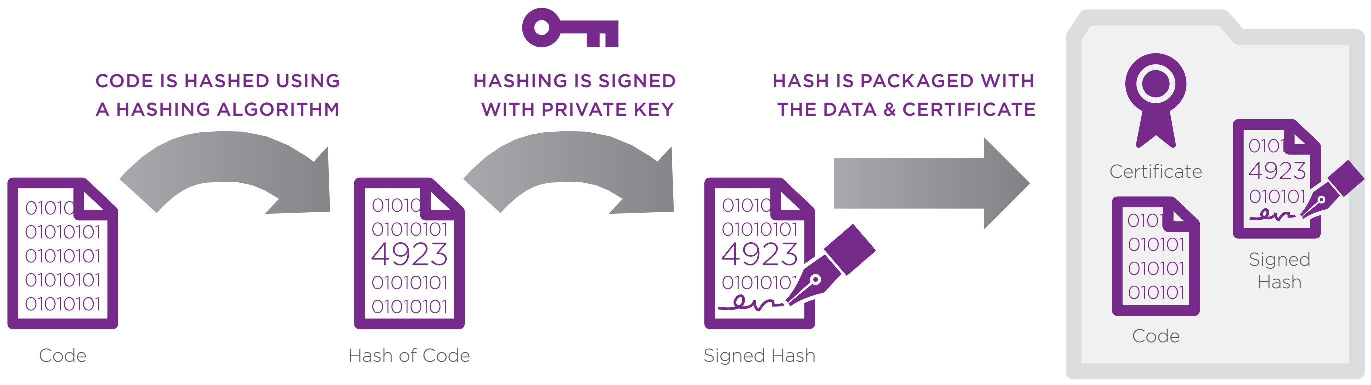
What is Code Signing?

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. In order to sign the code, a software 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, thus 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 software publisher's users 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 can be reproduced only by 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)

The code is now ready for distribution and is packaged in a form that will allow the user to verify for authenticity.



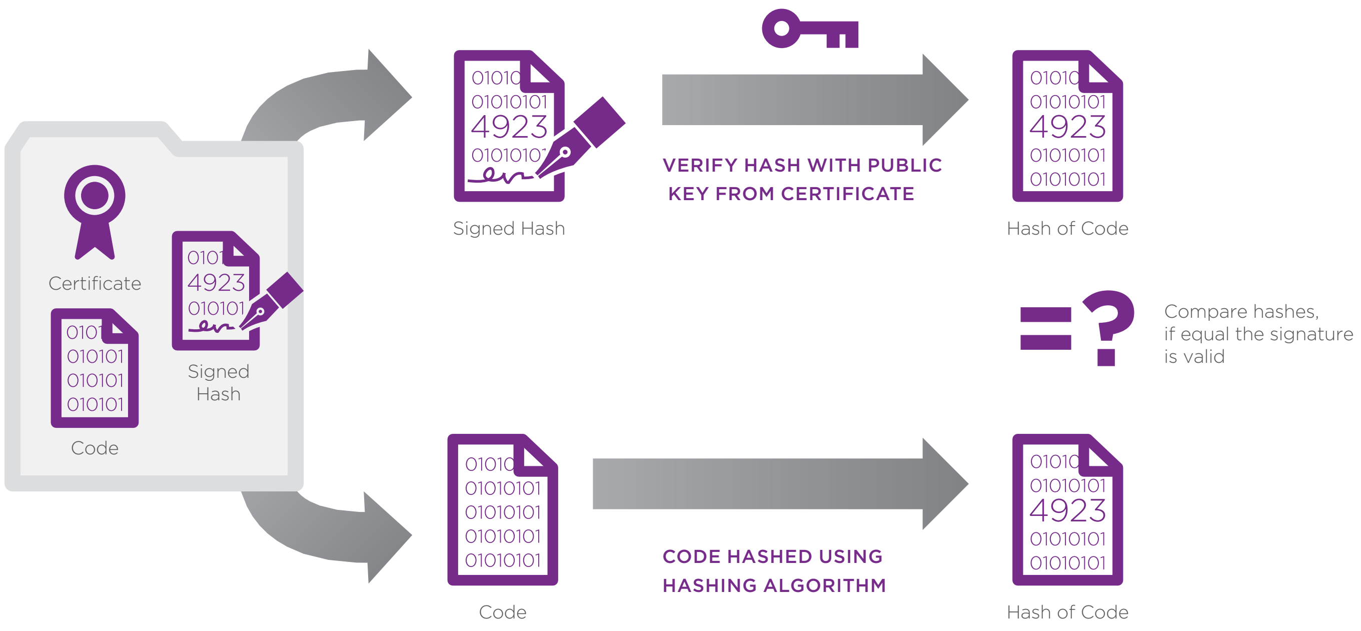
Verifying Code Authenticity

When a user agent loads the code, it checks the authenticity of the software using the packaged 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 either warning the user or rejecting the code, according to the level of security being used.

The signature is verified as follows:

- **Hash is Verified**
  - 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
- **Code Signing Certificate is Verified**
  - The code signing certificate is checked to ensure 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 ensure it is valid

If the hash and the certificate are valid, then the code is considered valid. As such, it is accepted by the user agent and presented for installation. If the file is not considered valid, the user agent displays a warning message.



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 to where guides can be found for each given application.

AdobeAIR

- **Digitally signing an AIR file**

Apple Mac OS X Developer Library

- **Code Signing and Application Sandboxing Guide**

Firefox XPI

- **Signing an XPI**

Java

- **Java Code Signing User Guide (Entrust)**
- **How to Sign Applets Using RSA-Signed Certificates (Oracle)**
- **Signing Code and Granting it Permissions (Oracle)**

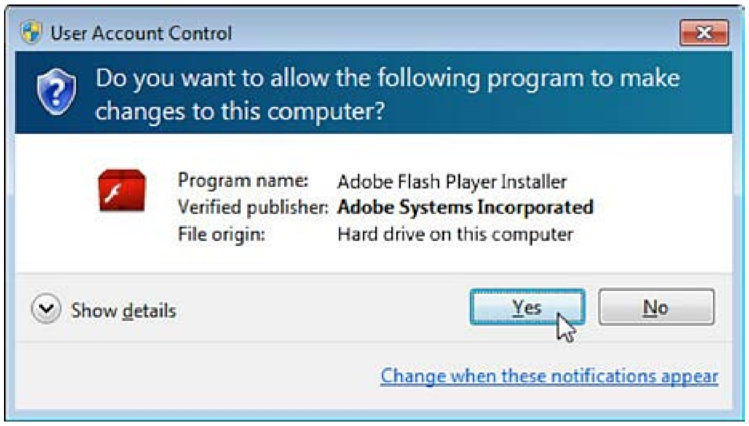
Microsoft

- Authenticode – **Entrust Signing Guide**
- Authenticode – **Signing and Checking Code with Authenticode**
- Windows Macro and Visual Basic Code Signing – **Entrust Signing Guide**
- Microsoft Windows Macro and Visual Basic Signing – **Signing a VBA Project**
- Microsoft Windows Driver Signing – **Driver Signing**

Code Installation Decisions

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 message:



User Account Control

Do you want to allow the following program to make changes to this computer?

Program name: Adobe Flash Player Installer
Verified publisher: **Adobe Systems Incorporated**
File origin: Hard drive on this computer

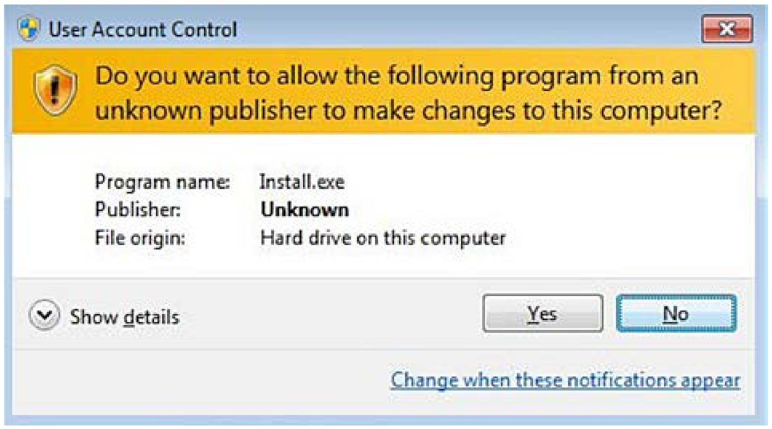
The user must decide if they trust the software based on the messages above. The statement provides the following:

- **Program Name:** "Adobe Flash Player Installer"
- **Publisher Name:** Adobe Systems, Incorporated
- **Code Signing Certificate:** The user would need to click on the "Show Details" drop down button, which will display a link to review the certificate

There are five simple steps users should take to determine whether software can be trusted:

1. Check to see if you were planning to install the software.
2. Check the file name to see if it indicates the software you were planning to install. In this case, the user is installing Adobe Reader 10, which the name seems to indicate.
3. Check the publisher name to see if it matches who you think wrote the software. This may be difficult as the software download site may be different than the publisher's site.
4. Check the code signing certificate to see if the publisher's name is in the certificate.
5. Check to see if the certificate was issued by a publicly trusted CA.

Conversely, here is a dialogue for code that may be untrustworthy:



Do you want to allow the following program from an unknown publisher to make changes to this computer?

Program name: Install.exe
Publisher: **Unknown**
File origin: Hard drive on this computer

The program name is "Install.exe," which is not specific enough to determine what code is being installed. The publisher's name is "Unknown," 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 the user cannot trust who signed 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 a user knows the time when the code was signed and the certificate was confirmed to be valid, then the user will also know the signature was valid at the time the software was published. Put another way, time-stamping is similar to a notarized handwritten signature which includes a third-party's confirmation of when the document was signed.

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

Please note that with some client software, the code verification may not be valid after the time-stamp certificate has expired. The new Minimum Requirements for Code Signing, discussed below, will require time-stamping authorities (TSAs) to use a time-stamping certificate with a maximum validity of 135 months that will be renewed every 15 months. As such, expect time-stamp certificates to have a lifetime of at least 10 years.

Time-stamping the signature is implemented as follows:

- The signature is sent to the 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, the 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 steps are completed by the user agent 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 TSA'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 revocation lists to ensure that it has not been revoked.
- The two hashes are compared. If the hashes are equal, the time-stamp is considered to be valid.

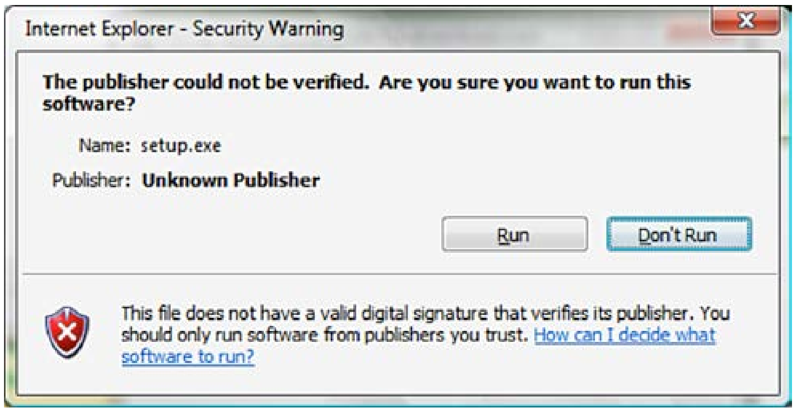
In the event that the code signing certificate must be revoked due to a compromise, the revocation will be made dependent on a specific date. The idea is that code signatures issued before the revocation date will remain valid and the software should still work.

Self-Signed Versus Publicly Trusted Code Signing Certificates

In most cases, software publishers have to sign their code in order to get it installed on an operating system. Publishers can sign their 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 publishers may decide to try a self-signed certificate, but there are differences between the two types of certificates that should be considered.

Self-Signed Certificate
Issuer provides their own identity, which is not published as part of the code verification security message
Issuer provides their own policy and quality
Signatures will provide a warning indicating that the software was created by an "Unknown Publisher"
Compromised certificates cannot be revoked and could harm software users



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?

To ensure user trust and code longevity, it is recommended that software publishers use a certificate issued from a publicly trusted CA.

Certificates Issued from Publicly Trusted CA
CA performs identity verification, which is displayed in a code verification security message
CA issues certificates in accordance with the industry policy and quality
Signatures will clearly identify the publisher's name
Compromised certificates can be revoked, and if time-stamping was used, code signed before revocation will remain trusted

Code Signing Certificate Standards

Through 2016, there are only requirements for the management and issuance of EV code signing certificates. A new standard for non-EV code signing certificates were implemented in February 2017. This standard is required by Microsoft, so all publicly trusted CAs will issue certificates to meet the requirements.

Minimum Requirements for Code Signing

The minimum requirements for code signing were not published by the CA/Browser Forum, but many of the requirements are from their Baseline Requirements. For code signing the minimum requirements also address other items such as:

- Publisher identity: Common name is the publisher's legal name, where the Organization name can also be the publisher's legal name or a DBA name.
- Minimum key size: 2048-bit RSA or ECC curves P-256, P-384 or P-521
- Validity period: Maximum validity period is 39 months.
- High Risk Requests: CAs should check databases to ensure known publishers of suspect code are not issued a code signing certificate.
- Private Key protection: Due to suspect code being signed with compromised keys, private keys are required to be encrypted on hardware or kept on a device separate from the host of the signing software function.
- Takeover Attack: Publishers' with history of a takeover attack will require a higher level of private key protection
- Certificate revocation: Specific revocation and processes have been established which include revocation requests from an application software supplier (e.g., Microsoft)
- Time-stamping: Certification authority, time-stamp certificates and time-stamp authority (TSA) requirements have been defined.

Extended Validation (EV) Code Signing Certificates

EV code signing certificates have two distinct advantages over certificates issued to the Minimum Requirements for Code Signing standard:

- Identity and authorization of the publisher must be completed in accordance with the CA/Browser Forum EV Code Signing Guidelines.
- Private keys must only be managed in hardware meeting the requirements of FIPS 140-2 Level 2 or equivalent.

The upside of EV code signing certificates is users 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.

Please note that Windows 10 requires drivers submitted for kernel mode signing to have their submission signed with an EV code signing certificate.

Application Reputation

Social-engineering attacks are more common than attacks on security vulnerabilities. The traditional defense against malware is a URL-based filter to screen out known malware websites. Microsoft also introduced a new defense called Application Reputation that is available starting with **SmartScreen Application Reputation**.

Application Reputation allows publishers and their applications to build a positive reputation over time through these principles:

- Well-known “good” applications have a better reputation than new applications
- Well-known “good” publishers have a better reputation than unknown publishers
- New applications signed by known “good” publishers can have a relatively high reputation from first release

Reputation can be built for unsigned and signed applications. Signed applications can build reputation at twice the rate of those that are unsigned. Reputation based on signing relies on the identification of the publisher by a trusted certification authority and the issuance of a code-signing certificate. Reputation is built by signing ‘good’ applications, but can be easily lost if the certificate is used to sign malware.

Traditionally, browsers have presented a User Account Control dialogue box for each application download. SmartScreen® Filter does not present a User Account Control dialogue if the application has built a good reputation. The benefit is that applications with good reputations will be installed without requiring the user to decide if they trust the software — they simply choose “Save” or “Run.” This update prevents users from becoming de-sensitized to User Account Control dialog boxes, and encourages users to make better decisions when these dialog boxes appear from applications with unknown reputations.

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.

Seven best practices for code signing include:

1. **Minimize access to private keys**
   - Allow minimal connections to computers with keys
   - Minimize the number of users who have key access
   - Use physical security controls to reduce access to keys
2. **Protect private keys with cryptographic hardware products**
   - Cryptographic hardware does not allow export of the private key to software where it could be attacked
   - Use a FIPS 140 Level 2-certified product (or better)
3. **Time-stamp code**
   - Time-stamping allows code to be verified after the certificate has expired or been revoked
4. **Understand the difference between test-signing and release-signing**
   - Test-signing private keys and certificates requires 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
   - 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
   - Establish a separate test code signing infrastructure to test-sign pre-release builds of software

5 Authenticate code to be signed

- Any code that is submitted for signing should be strongly authenticated before it is signed and released
- 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
- Implement virus-scanning to help 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 publishers may want to prompt a User Account Control dialogue box to appear when the code is installed in the future; this can be done by revoking the code signing certificate so 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

Conclusion

Code signing is required to install code on many platforms because it provides assurances of authenticity and origin. When signing code software publishers have to make decisions to protect their deployed products, the most important decision one can make is whether or not to use a trusted CA. The backing of a code signing certificate issued by a trusted CA is the best way to ensure end-user trust. Self-signed certificates should only be used for testing, not for production releases.

The second most important decision is whether or not to time-stamp code. In the event of a compromised key, a time-stamp may ensure that code is protected even if a certificate needs to be revoked.

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

Minimum Requirements for Code Signing will increase Internet security by setting a new bar to protect private keys from compromise. These requirements also provide a better mechanism to have code signing certificates revoked to limit the proliferation of malware.

Extended Validation code signing certificates are 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**

Minimum Requirements for the Issuance and Management of Publicly Trusted Code Signing Certificates, **https://casecurity.org/resources/**

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**

About Entrust Datacard

Consumers, citizens and employees increasingly expect anywhere-anytime experiences — whether they are making purchases, crossing borders, accessing e-gov services or logging onto corporate networks. Entrust Datacard offers the trusted identity and secure transaction technologies that make those experiences reliable and secure. Solutions range from the physical world of financial cards, passports and ID cards to the digital realm of authentication, certificates and secure communications. With more than 2,000 Entrust Datacard colleagues around the world, and a network of strong global partners, the company serves customers in 150 countries worldwide.

For more information about Entrust products and services, call **888-690-2424**, email **entrust@entrust.com** or visit **www.entrust.com**.

Headquarters
Entrust Datacard
1187 Park Place
Shakopee, MN 55379
USA

Twitter Facebook Youtube LinkedIn Google+ RSS

Entrust Datacard™