WHITE PAPER: SECURING YOUR PRIVATE KEYS AS BEST PRACTICE FOR CODE SIGNING CERTIFICATES

White Paper

Securing Your Private Keys As Best Practice for Code Signing Certificates

Stuxnet, a high-profile cyber-attack, used malware signed with legitimate code signing certificates. What went wrong and how can you protect your own assets?

By Larry Seltzer

Security Analyst and Writer

Symantec™

Norton™ SECURED

powered by Symantec

Securing Your Private Keys As Best Practice for Code Signing Certificates

Stuxnet, a high-profile cyber-attack, used malware signed with legitimate code signing certificates. What went wrong and how can you protect your own assets?

CONTENTS

Executive Summary

The efficacy of code signing as an authentication mechanism for software depends on the secure storage of code signing private keys used by software publishers. But a series of recent malware attacks using malicious programs signed with legitimate certificates shows that some developers don't take sufficient precaution.

Code signing is a technology that uses digital certificates and the public key infrastructure to sign program files so that users can authoritatively identify the publisher of the file and verify that the file hasn't been tampered with or accidentally modified. As with other PKI technologies, the integrity of the system relies on publishers securing their private keys against outside access.

Nobody who's talking knows what actually happened at the two companies whose private keys were compromised, but it appears that attackers obtained access to their private keys through some form of break-in, probably an electronic one. The damage to these two companies has been considerable and goes beyond the mud through which their names were dragged. Without minimizing the culpability of the actual criminals who broke in to their networks, it's fair to conclude that these two companies did not take sufficient precautions to protect their private keys.

Companies that are diligent and willing to invest in the appropriate security measures can make the compromise of their private keys highly unlikely. Changes in developer processes may be necessary, but these should not impose serious inconvenience. This paper describes recent security breaches and why they may have happened. It discusses best practices, especially for the Windows platform, which can help to safeguard the private keys associated with code signing certificates. Critical factors are:

- Security of the developers' networks and the developers' systems themselves.
- Minimal access to the private keys associated with genuine code signing certificates and the code signing process.

Introduction

The security world is abuzz over Stuxnet, perhaps the most sophisticated malware attack ever. It appears to have targeted certain facilities in Iran, particularly nuclear facilities, and infiltrated networks one would think to be actively secured. Stuxnet used many new and innovative tools to perform this infiltration, and one of them was to use binaries digitally signed with the code signing certificates of two legitimate companies.

A code signing certificate, also known as a software publisher certificate, has special fields and options particular to code signing. An SSL certificate, for example, cannot be used for code signing. Certificate authorities offer different types of code signing certificates for different code types. Symantec ™, for example, offers certificates for Microsoft Authenticode, for Java, for Adobe AIR, for Microsoft Office, and others.

[1]Dataquest Insight: Application Stores; The Revenue Opportunity Beyond the Hype, Gartner, Inc., January 2010

In one sense, an attack like this was inevitable. Code signing places certain responsibilities on users, and not all users are responsible. This paper aims to help you to take measures so that your organization's certificates and good name don't become the tools of malicious hackers.

It is essential that users keep private keys secure and confidential

The Basics of Code Signing

A reputable Certificate Authority (CA) that sells a code signing certificate will not just take the applicant's word for their identity. The CA will perform checks on the company names, phone numbers, and other information that is required to prove identity – a process that can take up to several days. With a Software Publisher Certificate and the associated private key, a programmer can digitally sign files distributed with the software. Code signing is a process that uses Public Key Infrastructure (PKI) technology to create a digital signature based on a private key and the contents of a program file, and packages that signature either with the file or in an associated catalog file. Users combine the file, the certificate and its associated public key to verify the identity of the file signer and the integrity of the file.

Code signing starts with public and private keys created by a user. Users can create their own digital certificate containing the public key using one of the many available free tools, or can purchase one from a trusted CA to whom they provided the public key. The user provides a name for the entity, typically a company, and other identifying information. The CA provides a certificate to the user. The certificate is also signed by theCA.

As the main subject of this paper emphasizes, it is essential that users keep private keys secure and confidential, restricting access only to those who absolutely need them. Anyone who has access to a private key can create software that will appear to be signed by the owner of the certificate. A general description of the process follows. For more detailed information on how developers actually work with the code signing process, see the Development section on page 6.

Digital signing of software begins with the creation of a cryptographic "hash" of the file being signed. A hashing function is a mathematical process that creates a hash value, often called a digest, which has a 1:1 correspondence with the original data. This digest provides no hints of how to recreate the original data, and even a small change in the original data should result in a significant change in the hash value.

The code signing program then uses the private key to sign the digest, meaning it generates a signature in the form of a string of bits. Good digital signature algorithms allow a user with the public key to verify the creator of the signature, but not allow someone who does not have the private key to generate a signature. The next step in the code signing process is to copy the Software Publisher Certificate into a new PKCS #7 (Public Key Cryptography Standards) signed data object.[1] This object is embedded into the signed file or, in some cases, in a separate file. Pay close attention to the signing and the expiration dates of the certificate. The shorter the lifespan of a certificate, the more frequently the identity of the signer is verified by the CA.

[1]RFC 2315 - PKCS #7: Cryptographic Message Syntax (http://tools.ietf.org/html/rfc2315)

A certificate authority can revoke a certificate for a number of reasons. For example, the user may violate legal terms, such as by signing a malicious program, or the user may report to the CA that the private key has become compromised. Client systems can check to see if a certificate has been revoked

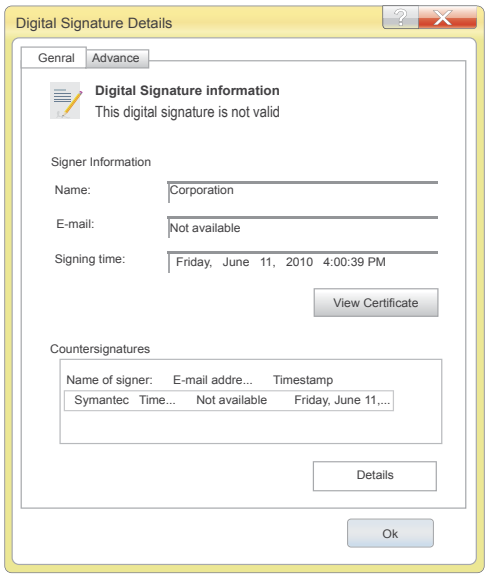


Figure 1. Windows shows a signed file's certificate as being valid

This image shows an example of how Windows provides to users a way to see details of the code signing certificate through the File Properties dialog box. The user can view the actual certificate details, including the issuer and when the certificate expires. Windows takes the signature and certificates that are attached to the program, recalculates the hash, uses the certificate to obtain the public key of the publisher, and uses that public key to verify several characteristics of the file:

- The certificate is valid.
- The digest in the certificate matches the one calculated by Windows.
- The signature is valid, meaning that it was created with the private key associated with the public key.
- The date of the signature is within the valid date range.
- The certificate has not been revoked by the CA.
- The CA is itself trusted or has its own certificate signed by one of the Windows Trusted Root Certificates

In many cases, this information is checked automatically. Apple Software Update, for example, downloads updates and checks the signatures before installing them. Such automated update systems don't usually provide any positive feedback for users on their connections to servers. If the connection is successful, users rarely see anything more informative than "Connected to update server."

In the static case, checking the signature causes the software to recalculate the hash for the file to check it against the stored one. When even a single byte is changed in the file from the above "valid certificate" example, Windows informs the user that the signature is not valid.

Anyone who has access to a private key can create software that appears to be signed by the owner of the certificate

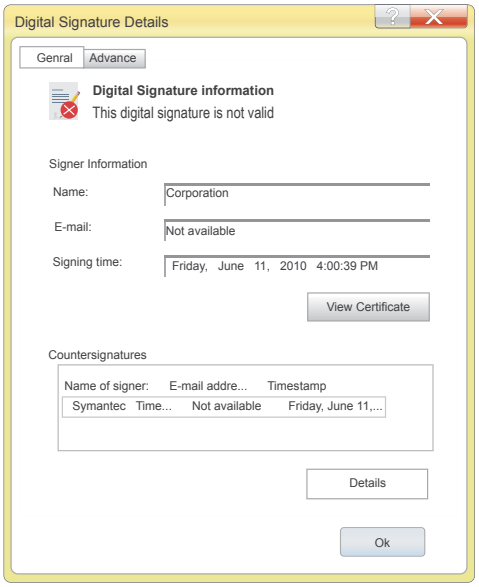


Figure 2. After the file has been tampered with, the signature is no longer valid

It is worth noting that code signing signatures don't check for safety or quality. Code signing confirms who published the file, and that it has not been modified since it was signed. It is important not to trust software merely because it is signed, but to examine the signer and evaluate their trustworthiness.

Development

Signing code isn't hard, but in most environments it is performed with command line tools. Microsoft has a suite of command line tools included with their development platforms for creating and using code signing certificates.[2] They are analogous to the tools offered by other vendors.

Microsoft's makecert.exe creates a digital certificate. This is necessary in order to create an in-house CA to sign in-house software, but most users don't need to touch this tool. cert2spc.exe converts a digital certificate into the Software Publisher Certificate, which is a certificate in code signing format. pvk2pfx.exe, the actual code signing tool, takes the imports, the private key, and software publisher certificate into a .pfx file. signtool.exe, the actual code signing tool, takes the .pfx file as input. Alternatively, signatures may be stored in a separate .CAT file which is created with makecat.exe.

The JDK (Java Development Kit) comes with a similar suite of command line tools. With Java, once the certificate is installed into the Java keystore, the jarsigner tool is run, specifying the JAR file to sign and the certificate to use. The signature is added to the JAR file.[3]

[2]makecert (http://msdn.microsoft.com/en-us/library/bfsktky3%28VS.100%29.aspx). cert2spc (http://msdn.microsoft.com/en-us/library/f657tk8f%28VS.100%29.aspx).pvk2pfx -. (http://msdn.microsoft.com/en-us/library/dd434714.aspx). signtool - (http://msdn.microsoft.com/en-us/library/8s9b9yaz%28VS.100%29.aspx). makecat (http://msdn.microsoft.com/en-us/library/aa386967(VS.85).aspx)

[3]How to Sign Applets Using RSA-Signed Certificates (http://java.sun.com/j2se/1.5.0/docs/guide/plugin/developer_guide/rsa_signing.html)

Apple's code signing tools are part graphical tool, part command line. There are graphical tools for generating certificates, but the code signing utility is a classic UNIX-style command-line tool that performs signing as well as verification of files.[4]

Microsoft's Visual Studio also integrates code signing into the development environment. Using the Solution Explorer, the programmer can select a certificate from the Windows Certificate Store, from a file, or have Visual Studio generate a test certificate. Thenceforth, code builds will automatically sign the software.[5]

Many other integrated development environments are capable of running the command line tools in an automated fashion as part of the build process. Once set up, code signing is easy.

Because self-signing doesn't involve a trusted CA, there is no revocation mechanism – so the compromise of a private key can be disastrous

Revocation

When a certificate has become compromised or for certain other reasons, the CA revokes it. The certificate itself contains links to the CA's Certification Revocation List (CRL) where clients can check to see if it is revoked, and there are two methods.

CRLs are simple lists of certificates, or rather serial numbers of certificates. When a client needs to check a certificate they download the CRL from the CA at the link indicated in the certificate. If the certificate serial number is in the list, the advisable course is not to trust the certificate.

CRLs have a number of problems, including the fact that they can become quite large and cumbersome, especially from a busy CA. A new solution was developed called OCSP or Online Certificate Status Protocol. This is a communication protocol between a client and the CA at a link specified in the certificate to determine whether that particular certificate is revoked. OCSP is preferred now, but clients must be able to check both it and CRL as some CAs don't always support OCSP.

A software publisher who determines that their certificate was compromised doesn't necessarily want to invalidate it for all the software they have signed with it and distributed. This is why revocation records include a date/time. If the publisher can say that the breach occurred no earlier than some point in time, they can revoke as of that date. Thereafter, when a client checks whether a certificate is revoked, they can compare the date/time of revocation to the date/time of the signature.

This scheme depends on a system of timestamps used in signatures. A timestamp is a date/time value and an assertion from a trusted source called a Timestamp Authority (TSA) that the signature was in existence at the time of the timestamp. If the timestamp precedes the revocation date of the certificate, then it is still considered valid for usage. A similar rule follows for expired certificates. If the signature timestamp precedes the certificate expiration date/time then the signature is still valid. Otherwise all code signed with the certificate would be suspect and untrustworthy.

[4]Mac OS X Reference Library – "Code Signing Guide" (http://developer.apple.com/documentation/Security/Conceptual/CodeSigningGuide/)
[5]MSDN – How to: Sign Application and Deployment Manifests (http://msdn.microsoft.com/en-us/library/che5h906%28VS.80%29.aspx)

Even though timestamps are optional, these issues make them highly advisable. Without a timestamp, code signed with an expired or revoked certificate must be considered untrustworthy. There are many public timestamp authorities.

Stuxnet exploited four zero-day vulnerabilities, which is certainly a record

.NET Strong Name Assembly Signatures

Windows .NET programs in the form of "assemblies" have a special code-signing method which involves self-signing, meaning that no trusted CA is involved. The developer generates his own public and private keys, uses the private key to sign the assembly file which contains both the public key and signature embedded. The embedded public key is used to test the embedded signature for validity. The public key plus a few other factors, including the file name, are used to form the "Strong Name" which can be considered unique.

Because there is no trusted CA, there is no revocation mechanism, and so the compromise of a private key can be disastrous. For this reason, some developers combine the Strong Name mechanism with more traditional code signing technologies like Microsoft's Authenticode.

sn.exe is the .NET Framework SDK's Strong Name Tool8, used to apply strong name signatures to assemblies.[6] It has options for key management, signature generation and verification. It supports test signing, release signing, and delay signing. With delay signing, only the public key is included in the file and space is reserved for a signature until later, after testing is complete, when the file is finally signed

Stuxnet – What Happened?

In June of 2010, a new form of malware was uncovered by Minsk anti-virus company – Virus-BlokAda.[7] The immediately interesting part of it was that it exploited a "zero-day" vulnerability, a term which refers to a software vulnerability which is exploited before it is otherwise known to the public. Before all the analysis was over, it turned out that Stuxnet, in fact, exploited four zero-day vulnerabilities, which is certainly a record. The mountain of research about Stuxnet includes a paper from Symantec entitled Win32.Stuxnet Dossier[8]. It summarizes what we know on the matter and adds some interesting new details dug out of the innards of the code.

Some summary characteristics of Stuxnet from the paper:

- It self-replicates through removable drives exploiting a vulnerability allowing auto-execution. Microsoft Windows Shortcut 'LNK/PIF' Files Automatic File Execution Vulnerability (BID 41732).[9]
- It spreads in a LAN through a vulnerability in the Windows Print Spooler. Microsoft Windows Print Spooler Service Remote Code Execution Vulnerability (BID43073).[10]

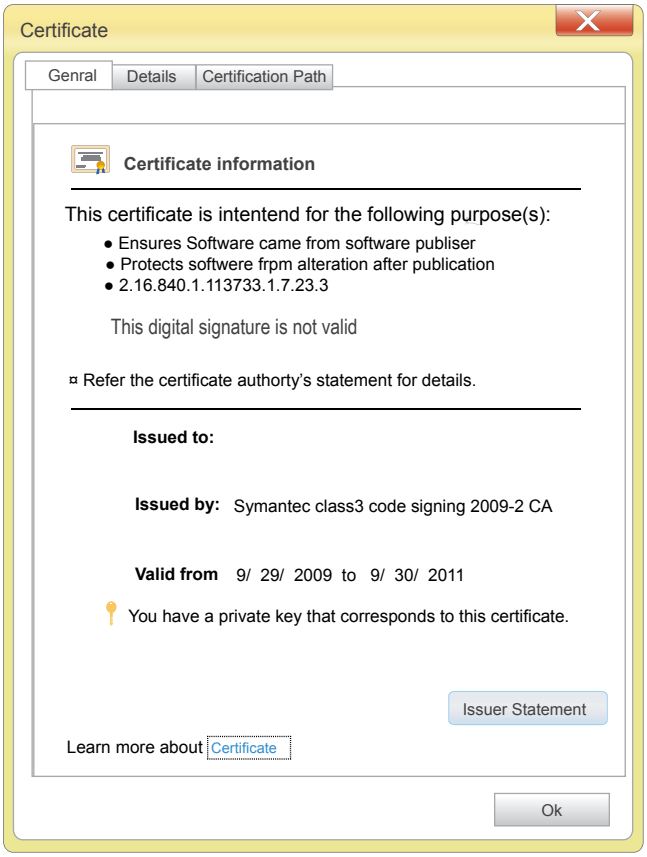
[6] sn.exe (http://msdn.microsoft.com/en-us/library/k5b5tt23(VS.80).aspx)
[7] Rootkit.TmpHider - http://anti-virus.by/en/tempo.shtml
[8] Win32.Stuxnet Dossier - http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf
[9] Microsoft Windows Shortcut 'LNK/PIF' Files Automatic File Execution Vulnerability (BID 41732) - http://www.securityfocus.com/bid/41732
[10] Microsoft Windows Print Spooler Service Remote Code Execution Vulnerability (BID 43073) - http://www.securityfocus.com/bid/43073

- It spreads through SMB (Server Message Block, the application-layer networking protocol for Windows) by exploiting the Microsoft Windows Server Service RPC Handling Remote Code Execution Vulnerability (BID31874).[11]
- It copies and executes itself on remote computers through network shares.
- It copies and executes itself on remote computers running a Siemens WinCC database server.[12]
- It copies itself into SIMATIC Step 7[13] projects in such a way that it automatically executes when the Step 7 project is loaded.
- It updates itself through a peer-to-peer mechanism within a LAN.
- It exploits a total of four unpatched Microsoft vulnerabilities, two of which are previously mentioned vulnerabilities for self-replication and the other two are escalation of privilege vulnerabilities that have yet to be disclosed.
- It contacts a command and control server that allows the hacker to download and execute code, including updated versions.
- It contains a Windows rootkit that hides its binaries.
- It attempts to bypass security products.
- It fingerprints a specific industrial control system and modifies code on the Siemens PLCs to potentially sabotage the system.
- It hides modified code on PLCs (Programmable Logic Controllers), essentially a rootkit for PLC.

Stuxnet was discovered in June but seems to have existed for at least a year prior. Microsoft said recently at a security conference that there is evidence that Stuxnet code dates back to January 2009. This is both impressive in and of itself, and confirmation of the sophistication of the programming in Stuxnet. The distribution pattern of Stuxnet was also unusual. While it eventually crept out to the world at large, the majority of infections for some time were in Iran. Symantec network analysis in July, 2010 showed 58.85% of actively-infected machines in Iran, with a substantial number of the remainder in other south Asian countries.[14]

All this gives the impression of a targeted attack written by an unusually sophisticated group of programmers. Forensic analysis of Stuxnet invariably remarks on the professionalism of the programmers and the quality of the code as compared to most malware. No wonder many suspect it was written by a state actor.

[11]Microsoft Windows Server Service RPC Handling Remote Code Execution Vulnerability (BID 31874) - http://www.securityfocus.com/bid/31874
[12]Siemens Automation Technology - http://www.automation.siemens.com/mcms/automation/en/Pages/automation-technology.aspx
[13]Siemens SIMATIC Step 7 - http://www.automation.siemens.com/mcms/simatic-controller-software/en/step7/Pages/Default.aspx
[14]Symantec Connect, W32.Stuxnet — Network Information - http://www.symantec.com/connect/blogs/w32stuxnet-network-information



But one of the earliest things analysts noted about Stuxnet was that the two binaries used to infect users' systems were digitally signed with code signing certificates from two legitimate companies. These certificates were quickly revoked by Symantec, who had issued them, but the fact that the programs were signed might have helped them to achieve access to legitimate machines and networks.

How could this have happened? The two companies aren't saying, and we can only speculate. But it's well-understood that it's the developers' responsibility to keep the private key secure. Not to excuse the theft, but there's an argument that no matter how it happened, it's the certificate holder's responsibility to prevent it. Clearly Stuxnet's perpetrators were sophisticated and had good resources at hand, but it's also entirely possible that the certificate holders did not go as far as they should have in the protection of their private keys.

A few specific circumstances are possible:

- A successful network intrusion into the company network allowed attackers to steal the certificate and private key.
- A successful network intrusion into the company network allowed attackers to gain access to a system, perhaps a developer system that had access to the certificate and private key.
- A successful physical intrusion allowed attackers access to these resources.
- An inside job: Someone at the companies either made the malware themselves or sold access to their systems or the certificate and private key.

In many cases there is no practical difference between the first two circumstances. If the private key is not properly secured, then any developer system that can use it can also get a full copy of it. Surely the thieves would be better off simply stealing the private key whole. It's possible, but unlikely, that the developers could follow some best practices (see the Best Practices section) but not others, in which case a developer system could sign code, but not necessarily obtain a full copy of the private key.

A compromised developer system is not just an opportunity to lose intellectual property; it's an opportunity for an outsider to plant unwanted code in your program

The inside job seems quite unlikely. First, there were two companies involved and two inside jobs would be hard to believe. And it has been widely reported that both companies have offices in the same office park in Taiwan, perhaps elevating the possibility of a physical intrusion in at least some aspect. For instance, perhaps all tenants in the office park share a common physical computer network.

Given the technical sophistication of Stuxnet's designers a computer-oriented break-in seems more likely. But there's nothing that can get past systems that are properly secured.

The Costs of a Breach

The damage to the reputation of a company that suffers a code signing certificate breach is serious enough. All security decisions reduce to trust decisions at some level, and trust must suffer in the case of such a breach.

The breach almost certainly indicates a severe security weakness, possibly in the building security, and certainly in the network security of the company. If the code signing certificates were stolen, what else was? What was modified? Is it possible that Trojan horse code was inserted into the company's source code? The number of distressing possibilities is large.

Customers must worry about these possibilities and whether any products of the company that they are running may have been tampered with.

Meanwhile, the company will have to have the CA revoke their code signing certificates. If there are other private keys that were stored in a similar manner to the ones that were stolen, all of them need to be revoked. This magnifies the impact of the problem.

The company must decide whether they can make a confident statement as to the date of the intrusion, or if it was definitely after a certain date. If so, they must set that as the date of the certificate revocation. If they can't, they must revoke the certificates, and probably revoke all certificates that were obtained as of the acquisition date of the compromised certificate.

The company must replace any code in the hands of customers that was signed with what is now a revoked certificate. This means contacting customers and explaining what happened, which you probably should do in any event. It's embarrassing, but it's the right thing to do if you hope to regain customer trust.

Best Practices: How to Avoid a Breach

There's an old joke that the only safe computer is one that's completely shut off from the rest of the world, and there's more than a grain of truth to it. That's why the most secure systems, including those with access to genuine code signing certificates, need to have the least connection possible to the outside world.

All protections, which are normally best practice, go double for developer systems. They must be "locked down" to the greatest extent possible, not necessarily because of access to private keys – normally, such systems should have no access to real code signing private keys – but because they have access to source code, your company's intellectual property. A compromised developer system is not just an opportunity to lose intellectual property; it's an opportunity for an outsider to plant unwanted code in your program.

A full accounting of the measures which are advisable for securing such systems are out of the scope of this paper, but they include using the principle of least privilege, multi-factor authentication for access to the system and network, blocking all but the most necessary network ports, installing all available security updates, and running an updated antivirus scanner. Give developers at least two systems to work with; actual development systems should probably be on a separate network with separate credentials from those used for ordinary corporate computing, like e-mail.

None of that specifically relates to the security of code signing private keys. But the measures described below do. A Microsoft paper entitled Code-Signing Best Practices[15] covers these and many other related issues on the subject. This paper will not go into the same level of detail on the implementation of measures which aid in the security of code signing private keys. We strongly recommend referencing it, especially for Windows development environments.

Separate Test Signing and Release Signing

You need to test your code when it's signed, but there's no reason to expose your real private keys and signing mechanisms more often or to more users than is necessary. Therefore it is best practice to set up a parallel code signing infrastructure using test certificates generated by an internal test root certificate authority. In this way, the code will be trusted only on systems with access to the test CA; if any pre-release code escapes the development/test network it won't be trusted. And these practices will minimize the chances that something will be signed which shouldn't be.

Signing your test code is good practice as well because it tests any effects that signing may have on the application (which should be little or none). By using test signing, you can be freer with access to test certificates; in fact, by distributing different certificates to different groups or individuals you can be certain who created a particular binary.

[15]Code Signing Best Practices - http://www.microsoft.com/whdc/driver/install/drvsign/best-practices.mspx

You can set up test signing in two ways: Use self-signed certificates or set up an internal test signing CA. In either case, the signature names should make clear that the signature is a test signature, for instance by making the organization name "TEST SIGNATURE – EXAMPLE CORP." See your legal department for any additional legal disclaimers they may wish to add to the certificate.

The makecert tool described in the Development section above can be used to generate self-signed certificates. The advantages are that the tools are free; they require no public key infrastructure and the developer can operate independently of the development network. But self-signed certificates won't be, by default, trusted and the developer or tester will have to do a little work to make them trusted. It's not the work that matters; it's the artificial circumstance of trusting a self-signed cert. And different developers won't necessarily be working with the same test parameters.

Test Certificate Authority

A better solution, especially as the size and complexity of the development and testing environments increase, is to deploy a test certificate server implementing a test root certificate. Microsoft Certificate Server (a.k.a. Active Directory Certificate Services)[16] can serve this role and works best in an Active Directory domain environment where Group Policy can be used to manage and revoke certificates. But there are other tools including OpenSSL[17], OpenCA[18] and EJBCA[19].

There are several different ways, procedurally, to run a test CA.

- The CA can issue certificates to all developers and testers.
- Clients can be required to make certificate enrollment requests of the server. These can either be fulfilled manually by an administrator or through some policy mechanism or ACLs set up by the administrator.
- Certain users, team leaders for example, can be designated to make certificate enrollment requests on behalf of certain other users.

Whether you use self-signed certificates or a CA, it's important to automate the signing process and integrate it into the development environment. This helps to ensure high quality and avoid problems. This is especially true of complex application environments that may have different signing requirements: device drivers and conventional applications, for example, may use different signature packaging configurations.

Internal or External Timestamp Authority

In most cases, you can use a public timestamp authority[20], but sometimes it is necessary to timestamp code without accessing public networks. In such cases you should consider the use of an internal Timestamp Authority, such as Thales (nCipher) Time Stamp Server[21] or OpenTSA[22].

[16]Active Directory Certificate Services - http://technet.microsoft.com/en-us/windowsserver/dd448615.aspx
[17]The OpenSSL Project - http://www.openssl.org/
[18]OpenCA Research Labs - http://www.openca.org/
[19]EJBCA, J2EE PKI Certificate Authority - http://sourceforge.net/projects/ejbca/
[20]For a list of public timestamp authorities, including some free ones, see Wikipedia - http://en.wikipedia.org/wiki/Trusted_timestamping
[21]THALES TIME STAMP SERVER - http://iss.thalesgroup.com/en/Products/Time%20Stamping/Time%20Stamp%20Server.aspx
[22]OpenTSA - http://opentsa.org/

This approach can be problematic, as the timestamp will be trusted only in the local network where the internal Timestamp Authority is trusted. This may be acceptable, in fact even desirable, for test environments. Refer to the Microsoft/ nCipher white paper, Deploying Authenticode with Cryptographic Hardware for Secure Software Publishing.[23]

Code signing requires access to the private key and thus should not be exposed to the public Internet. But time stamping may require access to public timestamp authorities on the Internet. Therefore, for final external release code it is best practice to use a separate, heavily secured but Internet-facing computer to perform the time stamping separate from the rest of the signing process.

Cryptographic Hardware Modules

Keys stored in software on general-purpose computers are susceptible to compromise. Therefore it is more secure, and best practice, to store keys in secure, tamper-proof, cryptographic hardware device. These devices are less vulnerable to compromise and, in many cases, theft of the key requires theft of the physical device.

Such devices are trusted for the most critical applications. For example, Symantec uses hardware security modules (HSMs) to hold and protect the private keys they use to sign digital certificates.

There are three types of such devices typically used:

- Smart cards.
- Smart card-type devices such as USB tokens.
- Hardware security modules (HSM).

The following table[24] shows the feature comparison between these device types:

Criteria	Smart card	HSM
Certification: FIPS 140-2 Level 3	Generally no	Yes
Key generation in hardware	Maybe	Yes
Key backup	No	Yes
Multifactor authentication	No	Maybe
Speed	Slower	Faster
Separation of roles	No	Yes
Automation	No	Yes
Destruction	Yes	Yes

Some HSMs will also never allow the export of keys, which is a significant security benefit. In such a case, in order to steal the keys you would need to steal the actual HSM, and even then, without further credentials, you may not be able to use the keys in it.

[23]Deploying Authenticode with Cryptographic Hardware for Secure Software Publishing - http://technet.microsoft.com/en-us/library/cc700803.aspx
[24]Microsoft - Code Signing Best Practices, p. 33 - http://www.microsoft.com/whdc/driver/install/drvsign/best-practices.mspx

FIPS (Federal Information Processing Standards) 140-2[25] is a NIST standard for conformance testing of cryptographic modules performed by NIST-certified test labs.

140-2 has four levels of security at which the device may be certified. At level 3, in addition to supporting certain cryptographic functions, the device must be highly-resistant to tampering.

As you can see from the table, smart cards and smart card-type devices are less robust and feature-rich than HSMs.

High-quality FIPS 140-2 Level 3-compliant smart card-type devices can be had for a reasonable price these days.

A recent test, including a torture test, of tamper-resistant USB flash drives[26] included devices for under $100 that take a licking and keep on ticking:

The hardware didn't flinch when thrown off the roof of a four story building, spiked down a flight of stairs, put through the dishwasher and anchored under Barnegat Bay for a month. The body took the blow of a 20-pound weight, although the cap did split open after a direct shot. The USB connector, however, was undamaged.

HSMs, typically in the format of an add-on card or network-attached device, can also perform cryptographic operations internally. Critically, they can generate and operate on keys internally and back them up encrypted externally, so that they need never be stored in plain text outside of the device. Some smart cards support key generation.

If only as a procedural matter, development organizations should require multiple sign-offs for the use of code signing private keys. This is usually implemented using the "k of n" rule, where k authorizations out of a total of n authorizers must approve for the procedure to proceed. Some HSMs implement this method directly with multifactor authentication. The same protection can be secured with smart cards or USB keys by storing them in a safe that has multiple keys or combinations and implements k of n. Smart cards can also require a PIN.

Conclusion

HSMs have dedicated cryptographic processors and operate in form factors, which lend themselves to higher performance. HSMs also support automation of signing operations. If you have high-volume needs for signing, you need an HSM.

HSMs support separation of administrative and operational roles, which increases the overall security of the process by not relying on a single individual or team. Finally, HSMs support key destruction, which is an important task in the event a key must be revoked.

[25]FIPS PUB 140-2 - http://csrc.nist.gov/groups/STM/cmvp/standards.html#02
[26]eWEEK - IronKey USB Flash Drives Prove Their Mettle by Matt Sarrel - http://www.eweek.com/c/a/Data-Storage/IronKey-USB-Flash-Drives-Prove-Their-Mettle-718976/

Physical Security

There is no security without physical security. If it's possible for an outsider, or even a visitor like a contract employee, to gain unnecessary access to code signing keys then all the cryptography measures are for naught.

We won't go into detail on what measures: cameras, guards, fingerprint scanners; are appropriate to protect your critical assets, but you need to take them as seriously as you take the computer security measures.

Sometimes we don't take security measures we know to be necessary until the threat is more appreciable than just a theory in a journal. This is what the case has been for the security of code signing certificates for many development shops. It has long been axiomatic that you must take serious and robust measures to protect your private keys. It's also been inconvenient and expensive to do so.

The usual excuses are unjustifiable in the face of Stuxnet. That Stuxnet appears to have been written by a first-class group of malcoders with excellent espionage capabilities is irrelevant. The two companies whose private keys were appropriated weren't targeted because they were themselves interesting; they were targeted because they were vulnerable and had assets – their code signing certificates – which were of value to Stuxnet's authors. They were in the wrong place at the wrong time, and anyone could get caught there.

Fortunately, if you value your intellectual property and your reputation, there are measures you can take to protect them. By securing your developer networks and facilities, formalizing code signing processes with multiple sign-offs and test signing, placing final code signing in a highly secured environment and using hardware cryptographic devices to implement the signing, you can make yourself too hard a target to bother with.

More Information

Visit our website

http://www.symantec.com/ssl

To speak with a Product Specialist in the U.S.

1-866-893-6565 or 1-650-426-5112

To speak with a Product Specialist outside the U.S.

For specific country offices and contact numbers, please visit our website.

About Symantec

Symantec protects the world's information and is the global leader in security, backup, and availability solutions. Our innovative products and services protect people and information in any environment – from the smallest mobile device to the enterprise data center to cloud-based systems. Our industry-leading expertise in protecting data, identities, and interactions gives our customers confidence in a connected world. More information is available at www.symantec.com or by connecting with Symantec at: go.symantec.com/socialmedia.

Symantec World Headquarters

350 Ellis Street
Mountain View, CA 94043 USA
1-866-893-6565
www.symantec.com

Symantec

Norton SECURED
powered by Symantec

UID: 072/10/13