Realtime publishers

The Essentials Series:
Code-Signing Certificates

What Are Certificates?

sponsored by

VeriSign®

by Don Jones

Copyright Statement

What Are Certificates?

Whether you realize it or not, digital certificates are *everywhere.* Once considered the domain of Web servers used by online banking and e-commerce sites, digital certificates continue to expand into every aspect of information technology, thanks to the significant security and privacy benefits they offer. Code-signing certificates in particular have been around for a long time, but they're starting to see much broader use. In this Essential Series, we'll look at the many ways in which code-signing certificates are being used, how they can benefit you and your business, how they work, and how you actually go about using them.

On its surface, a *certificate* is really nothing more than a delivery vehicle for a pair of encryption keys. However, when you dig a bit deeper, you find that encryption isn't necessarily a certificate's first and foremost business purpose but rather a means to an end.

Digital Certificates and Asymmetric Encryption

Everything a certificate is able to achieve is achieved through encryption, so it's worth a bit of time to explain how that works. Most people are familiar with *symmetric* encryption, which is simply encryption that uses the same encryption key to both encrypt and decrypt data—a password, if you will, or a combination lock. The biggest problem with symmetric encryption is that you have to somehow make sure everyone involved has the password; insecurely transmitting the password can, of course, result in it being divulged to someone you didn't intend to have it.

To alleviate the problem of sharing the encryption key, math geniuses have created *asymmetric* encryption. In this style of encryption, a pair of complementary keys is used. Anything encrypted by one key can only be decrypted by the other. For example, if Key A is used to encrypt something, Key A cannot be used to decrypt it; only Key B can be used to decrypt it, and vice-versa. Typically, these keys are referred to as the *public key* and the *private key*. As their names imply, you're intended to keep one of them to yourself and to share the other one freely with the world.

How you use these keys depends on your goals. If you want to keep something a secret—say, an email message—you obtain the recipient's public key, and use that key to encrypt the data. The recipient then uses their private key—which only they have—to decrypt the data. Code signing, however, is less about privacy than it is about *identity.* In code signing, a software publisher or developer uses their private key to encrypt a *digital signature* (more on what that is later); the rest of the world can then use the corresponding public key to decrypt and analyze that signature. If the decryption is successful, they've successfully verified the identity of the software publisher or developer.

Certificates as a Form of ID

That, in fact, is what certificates are really all about: Identity. Let's take email as an easy-to-understand example: If someone sends me an email purportedly from my boss, how can I verify that it's really from my boss? Obviously, I could pick up the phone and call the number that I know goes to my boss' phone, but that kind of defeats the purpose of email and would get pretty tedious. Instead, my boss could start using a digital certificate to sign his email. In the signature process, his email software analyzes the email content and creates a *hash* from it.

> **Note**
>
> A *hash* is the result of running clear-text information, such as the content of an email, through a well-known encryption algorithm that uses a fixed encryption key—one that everyone in the world knows. A portion of the result is discarded, meaning it's impossible to take the hash and re-construct the clear-text original, even though you know the encryption key.

His email software then uses his private key to encrypt the hash, and appends the result—called a *digital signature*—to the body of the email. It's then sent on its way, with the clear-text body of the email still in clear-text and still readable by anyone who sees it. Signatures aren't intended for privacy, they're intended for *identity.* When I receive the email, my email software obtains my boss' public key (in reality, this is usually sent along with the email itself), and uses it to decrypt the signature. My email software then creates a hash from the clear-text portion of the email and compares it with the hash that was in the formerly encrypted signature. If the two hashes match, I know a couple of things:

- I know the message came from my boss, because I could use his public key to decrypt the signature. If I couldn't decrypt the message using his public key, the only reason would be that it wasn't encrypted with his private key in the first place—meaning the message didn't come from him.
- I know that the message content hasn't changed because my boss sent the email. If the content had changed, the hash my email software calculated would be different from the one contained in the digital signature.

If you're sharp, you'll notice the potential flaw in this logic: My boss' identity, and the validity of the message, can only be trusted if I know for a fact that *only my boss* has a copy of my boss' private encryption key. In other words, if someone had issued a certificate with my boss' name on it *to someone other than my boss,* the whole process falls apart.

Code signing works the same way: A software publisher (say, ACME Software Publishing) uses their private key to digitally sign their code. This signature includes a hash, which allows users of the software to verify that (a) the software really did come from the publisher as claimed, and (b) the software hasn't been modified since the publisher released it. Again, however, this assumes you can trust that whoever has the private key for ACME Software Publishing really *is* ACME Software Publishing.

> **Note**
>
> In fact, some commercial Certification Authorities (CAs) now refer to their products as *Digital IDs* rather than merely *certificates,* to better convey the identity aspect of the technology. Anyone can sell you a couple of encryption keys; only a good, trusted CA can vouch for your *identity.*

Certificates, Publishers, and Trust

So where do certificates come from? From a CA—either a commercial one or one that is internal to a company. In the case of code-signing certificates, commercial CAs are normally used because they're *trusted* by the world at large.

What does trust have to do with it? Let's talk about driver's licenses. Any bar in any state in the US will accept a driver's license from any other state as proof that you're old enough to enjoy an alcoholic beverage. Why? Think about it: Driver's licenses are a form of certificate in that they attest to your identity, and certain facts about you, such as your age. These certificates are issued by a CA in each state—the Department of Motor Vehicles, in most cases. Each state trusts the other states' CAs because we all know that each state has roughly the same procedures for verifying identity. In other words, before being handed a license that says you're Bill Gates, each state is going to perform the same basic checks to make sure you really *are* Bill Gates. The states trust not so much each other but rather the *identity verification process* that they all share.

Digital certificates work the same way. Before a CA hands out a digital certificate to ACME Software Publishing, that CA needs to do something to make sure that the folks receiving the certificate *really are* ACME Software Publishing. If a CA is doing nothing more than, say, making sure that the recipient's email address ends in "@acmesoftware.com," that's not a very good check—there is no guarantee that the company actually owns that email domain, and email addresses aren't impossible to fake. A good CA will dig deeper, perhaps checking public records, telephone books, and other information to verify the recipient's identity. When we talk of *trusting a CA,* what we mean is that we trust the identity verification process they use. If you don't *know* what process a CA uses, you shouldn't trust them!

When you examine the digital certificate attached to a piece of signed code, you're not only looking at the name of the publisher who signed the code but also the complete chain of everyone who issued and authorized that certificate—all the way back to the CA. If you trust the CA who issued the certificate, you can trust the signature on the code; if you don't trust the CA, the signature is worthless.

How Code-Signing Certificates Affect Your Code

Signing an application lets users know that the application came from you, and that it hasn't been modified in any way since you signed it. Different operating systems (OSs) and platforms behave differently with regard to signed code. For example, in Microsoft Windows, signed code has an additional Properties tab that displays information about the signature. As Figure 1 shows, the user can see who the application came from and be reminded that the application hasn't been changed since it was signed.

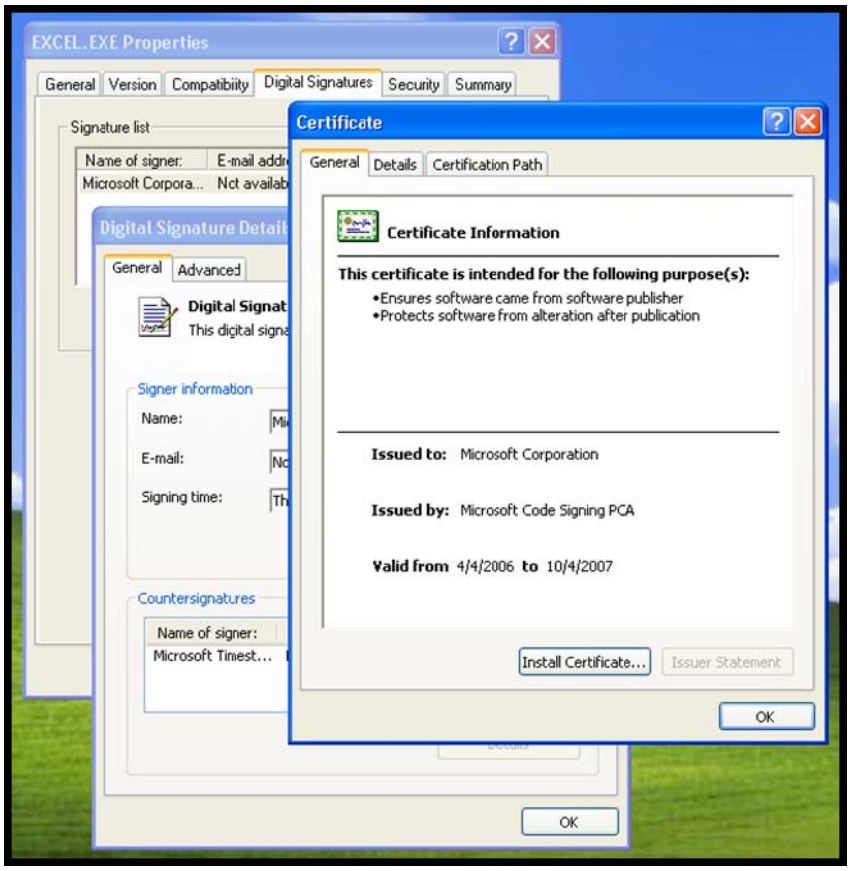

Figure 1: Viewing the certificate on a signed application.

A user can also view the *certification path,* as Figure 2 shows. Notice in Figure 1 that the certificate used to sign this application was issued by "Microsoft Code Signing PCA," but that doesn't necessarily mean we trust that issuer. The certification path shown in Figure 2 reveals that the Microsoft Code Signing PCA was itself authorized by the Microsoft Root Authority—and Windows ships from the factory preconfigured to trust that top-level CA.

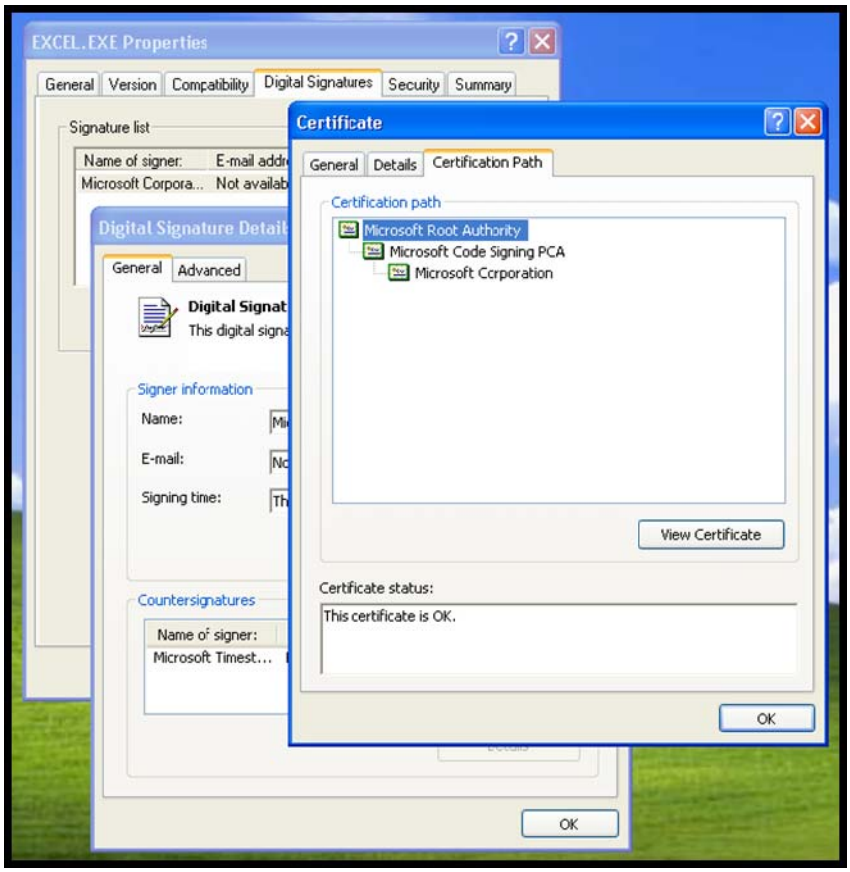

Figure 2: Reviewing the certification path to find the top-level CA.

Of course, as only Microsoft has access to certificates issued by this trusted-by-default CA, the rest of us have to obtain certificates from elsewhere. We could just deploy our own CA, as Microsoft has done, but we would have to convince all of our users to trust our CA, which is likely to be an uphill battle. Instead, software publishers typically rely on code-signing certificates issued by a commercial CA because many OSs are preconfigured to trust the most common commercial CAs, and because the world at large is accustomed to trusting the process these CAs use to verify identities.

Why Unsigned Code Is Bad Code

Unsigned code is unknown and unsafe. Users don't know who wrote it, and even if they trust the person that they *think* wrote it, they have no way to verify that the software they're about to run actually came from that trusted person. Further, even if they get over that issue of trust, they don't know whether the software has been modified since their trusted author released it into the world. There is nothing easier for a hacker than to modify a piece of unsigned code and send it out into the world to sow confusion and damage. Frankly, the only reason most users are at all willing to run unsigned code is because they haven't the slightest idea what signatures are! That's why many OSs—some mobile ones are a great example—are configured to never run untrusted (unsigned) code. The users don't get a choice; in order to protect them from harm they don't even understand, the platform manufacturer has made the decision to simply not trust code that isn't signed.

The Certificate Lifecycle: Issue, Renew, Revoke

Of course, we have to acknowledge that things can go wrong with certificates. The biggest worry is that they can be stolen or lost—meaning an attacker could commit the software equivalent of identity theft and start signing malicious code as if they were a legitimate, trusted software publisher. For that reason, certificates—not unlike driver's licenses—have a lifecycle:

- **Issue.** This is when the certificate is produced by the CA, after verifying the identity of the recipient.
- **Renew.** Certificates expire, so recipients must periodically go back to the CA for a renewed version. This helps ensure that a stolen or lost certificate can't be used indefinitely.
- **Revoke.** A stolen or lost certificate can be revoked. The CA publishes a list of revoked certificates, and Internet protocols exist that allow computers to consult this list. Anything signed by a revoked certificate is considered suspect.

Commercial CAs help manage this process and maintain the not-inconsiderable infrastructure necessary to make revocation work properly when it's needed.