Realtime publishers

The Shortcut Guide™ To

Certificates in the Enterprise

sponsored by

thawte™ it's a trust thing™

Don Jones

Introduction to Realtime Publishers

by Don Jones, Series Editor

For several years now, Realtime has produced dozens and dozens of high-quality books that just happen to be delivered in electronic format—at no cost to you, the reader. We've made this unique publishing model work through the generous support and cooperation of our sponsors, who agree to bear each book's production expenses for the benefit of our readers.

Although we've always offered our publications to you for free, don't think for a moment that quality is anything less than our top priority. My job is to make sure that our books are as good as—and in most cases better than—any printed book that would cost you $40 or more. Our electronic publishing model offers several advantages over printed books: You receive chapters literally as fast as our authors produce them (hence the "realtime" aspect of our model), and we can update chapters to reflect the latest changes in technology.

I want to point out that our books are by no means paid advertisements or white papers. We're an independent publishing company, and an important aspect of my job is to make sure that our authors are free to voice their expertise and opinions without reservation or restriction. We maintain complete editorial control of our publications, and I'm proud that we've produced so many quality books over the past years.

I want to extend an invitation to visit us at http://nexus.realtimepublishers.com, especially if you've received this publication from a friend or colleague. We have a wide variety of additional books on a range of topics, and you're sure to find something that's of interest to you—and it won't cost you a thing. We hope you'll continue to come to Realtime for your educational needs far into the future.

Until then, enjoy.

Don Jones

Copyright Statement

Chapter 1: Digital Certificates Crash Course

Digital certificates are becoming more and more prevalent throughout today's enterprises, and for good reason: In many cases, they offer the opportunity for better security, less complexity, and an overall more stable and trustworthy IT environment. In many ways, they're a magic bullet for some of IT's longest-standing and trickiest problems, although certificates sometimes seem like one of the most-overlooked and often-ignored technology solutions out there. If you've never looked at digital certificates before, now is definitely the time: More technologies are using them in more ways, and the sooner you start taking advantage of them, the better off you'll be.

In this guide, I'll be introducing you to digital certificates and their place within the enterprise. In this chapter, I'll begin with an introduction to digital certificates and how they work; if you thought that they were just for encrypting email and Web server traffic, then you may be in for a bit of surprise. In Chapter 2, we'll look at the many ways in which digital certificates are used within a modern enterprise, including some familiar ways and some that might surprise you. In Chapter 3, we'll dive deep into the issue of certificate trust and explore the real value of a certificate (it isn't encryption, believe it or not) as well as the responsibilities of someone who issues certificates (whether it's someone internal to your enterprise or a commercial partner). Finally, in Chapter 4, we'll cover some of the "gotchas" surrounding certificates—the things that *can* come back to bite you if you don't know about them.

Now, let's jump in and see what certificates are, and how they work.

Encryption

It all comes down to math. A basic encryption algorithm—that is, a set of procedures used to encrypt or decrypt data—typically uses some complicated mathematical formulas in conjunction with an *encryption key.* In classic encryption, you use the same key to encrypt the data and to decrypt the data; this is called *symmetric encryption.*

Symmetric Encryption

One of the simplest forms of symmetric encryption, used by Julius Caesar (and involving no real math), is called ROT-3. It simply rotates the alphabet three characters, substituting each character in the clear-text message with the equivalent letter from the rotated alphabet. Figure 1.1 shows an example.

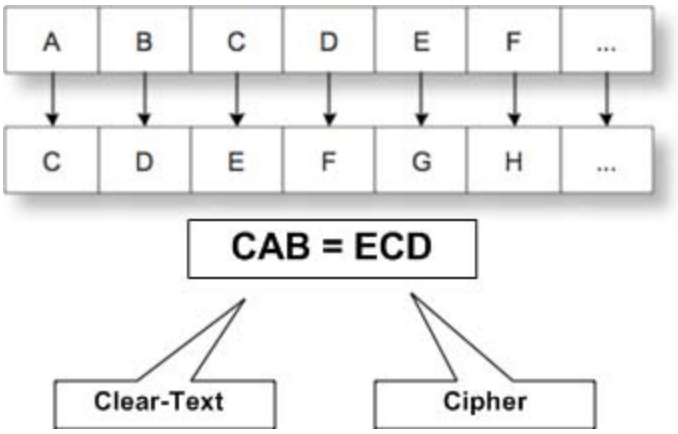

Figure 1.1: Using a ROT-3 encryption algorithm.

The "encryption key" in this case is simply the knowledge that the alphabet was rotated *three* characters to the *left*; obviously, you could choose any rotation direction and amount. The trick is in making sure both parties—the person sending the message and the person receiving the message—are using the same rotation. Therein lies the trick: Communicating the encryption key to both parties without it being accidentally divulged to someone else in doing so.

Take another set of examples: ZIP files or Microsoft Office documents. These are commonly used to contain sensitive information, and are often protected by passwords. If sent by email, the password-protected file is somewhat safer from accidental disclosure than a non-protected file. However, you're still stuck with the task of somehow alerting the recipient to the password without that password being compromised. Folks usually handle this by sending the password in a separate email from the file itself, as if two emails were somehow more difficult to intercept than one.

Although symmetric encryption has its place—it's what Windows Active Directory (AD) uses, by default, at the heart of its Kerberos authentication protocol—its utility and security is sharply limited by the necessity to share a common encryption/decryption key.

Asymmetric Encryption

The password-sharing problem is solved by using *asymmetric encryption,* which uses two sets of keys. One of the keys, referred to as the *private key,* is intended to be held in private (as the name suggests) by a single person or entity. The other key, called the *public key*, is made freely available to anyone who wants it. The trick is that anything encrypted with the public key can only be decrypted by using the private key; likewise, anything encrypted with the private key can only be decrypted by the public key.

This arrangement solves the problem of password-sharing and raises subtle and exciting possibilities that go beyond the privacy and security granted by encryption. In fact, encryption itself simply becomes a means to more valuable ends. Let's take email as an example.

It's easy to think of occasions when you want the contents of an email to remain private. However, what most folks don't think about is that you also usually want to know *who* is sending emails to you. We all know that email addresses are notoriously easy to fake, and even to fake *slightly.* For example, you might trust myfriend@mycompany.com, but when you get an email from the similarly-named myfriend@mycmpany.com, it's easy to mistake it for your trusted contact. Email addresses, in other words, are a poor form of identification.

So let's bring asymmetric encryption into the game. Your trusted friend obtains a set of asymmetric encryption keys. Without diving too much into the infrastructure that makes this happen (we'll do that later in this chapter), your friend makes it possible for you to obtain her public key from a trusted third-party. So you're holding a public encryption key for your friend, and you *know* that it's your friend's public key, not someone else's. Your friend has the corresponding private key safely tucked away on her computer.

Your friend writes you an email—one she's not concerned about keeping private but that you *must* trust in having come directly from her. Her email software takes the contents of the email and creates a *hash.* It then encrypts the hash by using her private key, and appends the entire result to the email. That result, by the way, is called a *signature*. Figure 1.2 illustrates the process.

Note

A *hash* is created by taking some data—such as the body of an email—and running it through a well-known encryption algorithm. *Well-known* means that both the encryption methodology *and* the encryption key are known to the world at large; there are several standardized hashing routines that folks use. After encrypting the data, a portion of the encrypted result is discarded, resulting in the final *hash.* Because the hash is missing part of the encrypted data, the hash itself can never be successfully decrypted; that's not its purpose.

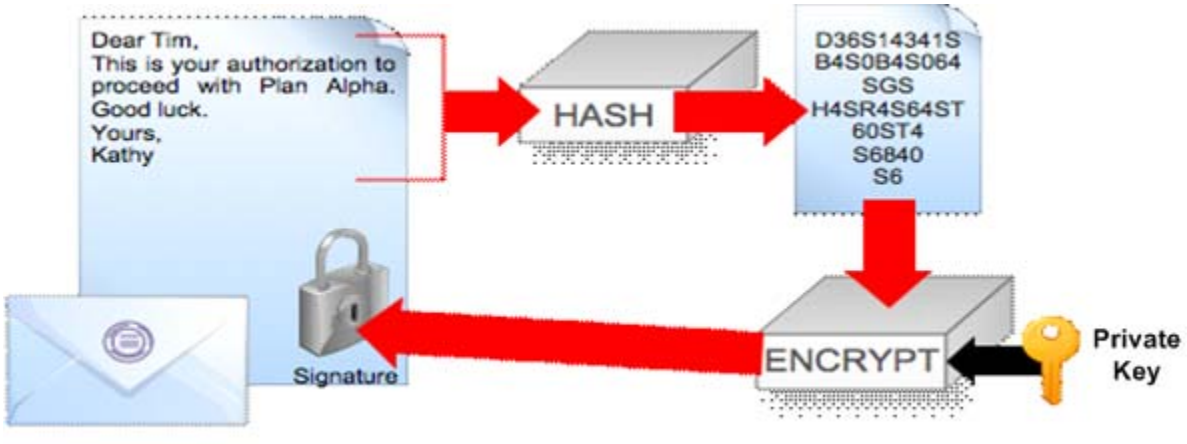

Figure 1.2: Signing an email.

Your friend hits "Send" and the email wings its way to you. Your email software sees who the email is from, and grabs that person's public key. Keep in mind that the signature portion of the email can *only* be decrypted using your friend's public key, and that's what your email software tries to do next. If someone *pretending* to be your friend sent the email, they wouldn't have her private key, and so decrypting the signature with your friend's public key will fail. That means the signature is *broken*, and your email software will display some kind of appropriate icon or message to that effect.

If, however, your email software is able to successfully decrypt the signature, you *know* that the message came from your friend—that's the only way the decryption would have worked. Your email software then takes the clear-text body of the email and uses the same algorithm to create a hash from it. Your email software compares the hash it created to the hash that was in the signature; if they match, you know that the body of the email hasn't been tampered with since your friend sent it. If the hashes don't match, the signature is again considered broken, and you should suspect that the email's contents have been tampered with. Figure 1.3 shows how it comes together.

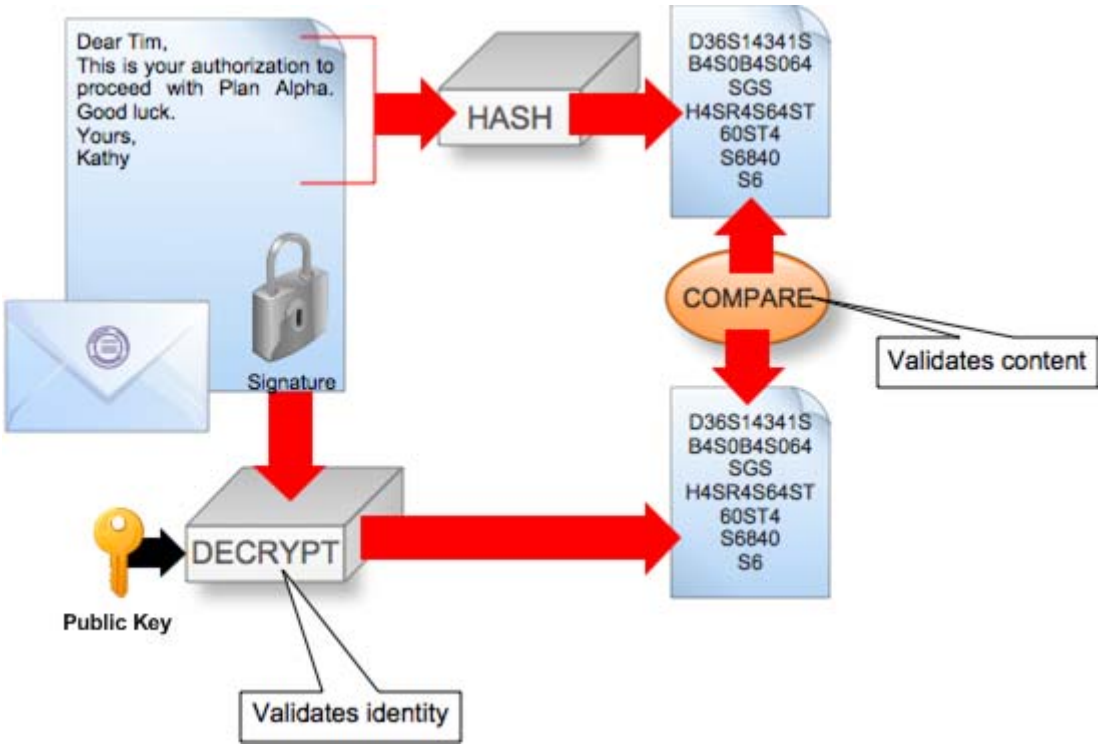

Figure 1.3: Verifying an email signature.

In this example, the encryption was used to prove *identity*, not to provide privacy. Providing privacy is straightforward: You already have your friend's public key, so you can use it to encrypt things and send them to her. Only she will be able to view those things, because only she has the private key needed to decrypt them. You can see that *key exchange* is an important part of this process: In order for your friend to encrypt something to you, she'll need to have your public key.

Note

This was meant to be a simple illustration, but this *same* process—either for signing or encryption—essentially encapsulates everything that you can do with a digital certificate. Certificates' power is in the many and subtle ways in which these two simple techniques can be employed, and in the *trust* aspect of a certificate.

Digital Certificates: A Place for Your Keys

A digital certificate consists of two main elements, the first of which is a place to store a public and private key pair. In practice, a certificate is typically used only to hold the public key so that the private key can remain, well, private.

But a certificate holds something else of great importance: The certificate's *provenance*, which is more commonly called the *certification path.* This is essentially the name of the entity that issued the certificate, the name of the entity that authorized the issuing entity, and so forth. For example, Figure 1.4 shows a software application that has a digital signature applied. The signature actually includes the entire certificate used to create the signature; this allows anyone to access the public key needed to decrypt the remainder of the signature and check the hash contained within. But the certificate also shows that the certificate was issued by the "Microsoft Code Signing PCA," which was in turn authorized by the "Microsoft Root Authority."

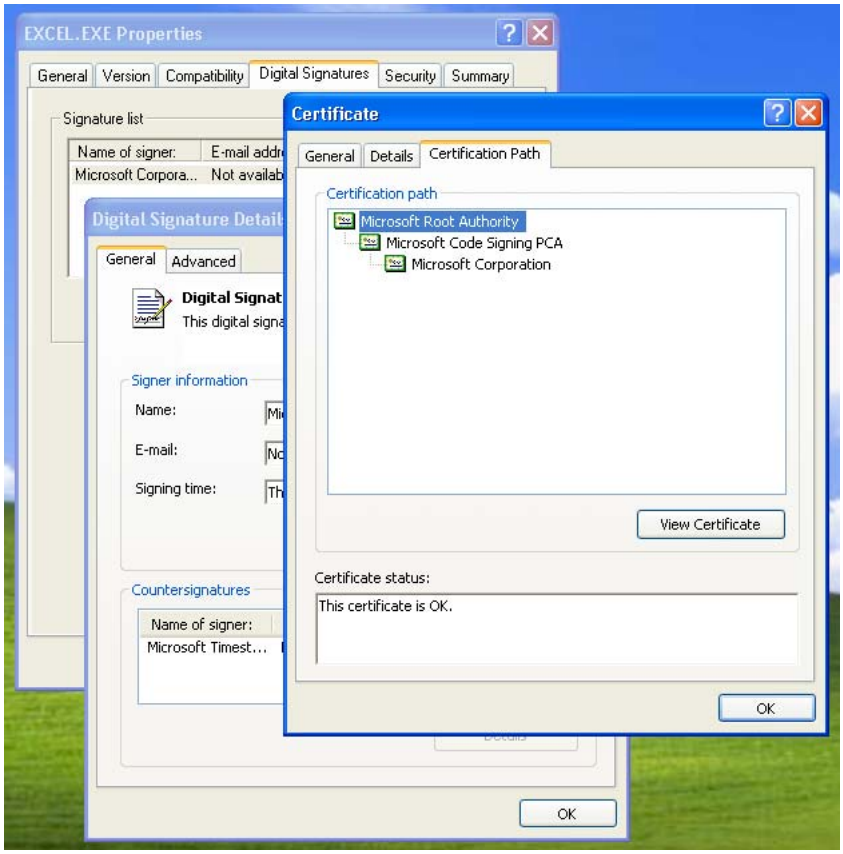

Figure 1.4: Checking the certification path of a signed application.

So let's quickly review:

- A certificate contains the public key part of the asymmetric key pair. The private key portion is held only by the entity to which the certificate was issued.
- A certificate lists its entire certification path. This lets anyone not only access the signature (by means of the enclosed public key) but also see where this certificate came from.

This *certification path* business is actually pretty complicated, and it plays directly into the certificate's value as a form of identity verification.

Understanding the Chain of Trust

Most US citizens have a driver's license, typically issued to them by their home state's Department of Motor Vehicles or similar authority. The main purpose of the driver's license is to serve as a form of identity verification: By matching your face to the picture on the license, someone can determine that the license is indeed yours. They can then review the other information on the license, such as your date of birth, and be reasonably confident that the information applies to you. Businesses that serve or sell alcoholic beverages, for example, can use this information to determine whether they're legally allowed to serve you.

Licenses as Certificates

In many regards, a driver's license serves the same purpose as a certificate. In fact, many of the companies who sell certificates now refer to them as "digital IDs," reflecting the certificate's role in identity verification. Any entity that provides certificates is referred to as a Certification Authority (CA), and they play a similar role to the state Department of Motor Vehicles.

When someone obtains their first driver's license, the Department of Motor Vehicles needs to verify all of the information that they're about to attest to on that person's new license, such as the person's hair color, height, and eye color, which can be easily verified visually by the person issuing the license; the person's date of birth, however, can't be. That's why most states require the applicant to furnish some proof, such as a birth certificate.

Almost every state in the US follows a similar verification procedure when issuing licenses. For that reason, bars and restaurants in a state such as Nevada trust not only Nevada driver's licenses but also the licenses issued by all the other states. In digital certificate terms, we'd say that Nevada *trusts* the Nevada Department of Motor Vehicles CA and trusts the CAs from the other 49 states. Of course, it's not quite that straightforward: In reality, driver's licenses are obtained from one of many different Department of Motor Vehicle offices throughout each state. These offices are authorized by each state's central department, but the offices each act as their own CA. The department sets down the verification rules, and trusts in each office to follow them. This setup creates a certification path not unlike that for a piece of signed software, as Figure 1.5 shows.

Figure 1.5: The certification path for a driver's license.

Businesses don't have to explicitly trust each Department of Motor Vehicles office; instead, they trust the top-level CA—the states' Department of Motor Vehicles. Although the central department doesn't actually issue any licenses, they do authorize *subordinate CAs*—the Motor Vehicles offices—to do the issuing for them.

This concept of trust is *hugely* important for digital certificates. For example, if Nevada's Department of Motor Vehicles suddenly started issuing new licenses *without* demanding proof of birth dates, businesses might not want to accept Nevada licenses as proof of age. Businesses throughout the US might stop trusting the Nevada CA, making Nevada's certificates—that is, their licenses—useless in bars and restaurants across the nation. That loss of trust wouldn't happen because businesses didn't trust the individual offices or the individual license holders; the loss of trust would come because businesses would no longer trust the *process* that Nevada was using to verify identity and information prior to issuing a certificate. I mean, a license.

Proving Trust

But let's say Nevada keeps doing things the way that they have been, and that businesses everywhere continue to trust them and the licenses they issue. When you go to an airport to fly, do you notice the type of scrutiny given to your driver's license (if you've used that as your photo ID to proceed beyond the security checkpoint)? The security officer isn't concerned about your age; they want to match your face to the photo ID to verify that it's *your* ID, and they want to make sure the name on the ID matches the name on your airline boarding pass. They're using the ID in much the same way that a bar or restaurant would, although for different purposes.

But the officer at the security checkpoint needs to be aware of the fact that forged IDs exist: ones claiming to be issued by (to continue the example) the state of Nevada, but which in reality were made up illegally in some dark alley somewhere. The officer needs not only to trust the Nevada Department of Motor Vehicles but also to verify that your license is in fact one that the department has authorized. The officer does so by shining an ultraviolet light on your license, checking for the presence of a difficult-to-forge holographic watermark that is applied to genuine licenses.

Digital certificates have a similar way to check for authenticity. First, you need to trust the top-level CA, called the *root CA,* in the certification path (in Figure 1.4's example, the Microsoft Root CA). Although the decision to trust a CA is one made by a human being, the way a computer is instructed to trust a CA is by installing that CA's certificate, which contains the CA's public key. You can see these in Windows' Internet Options Control Panel, for example, as Figure 1.6 shows.

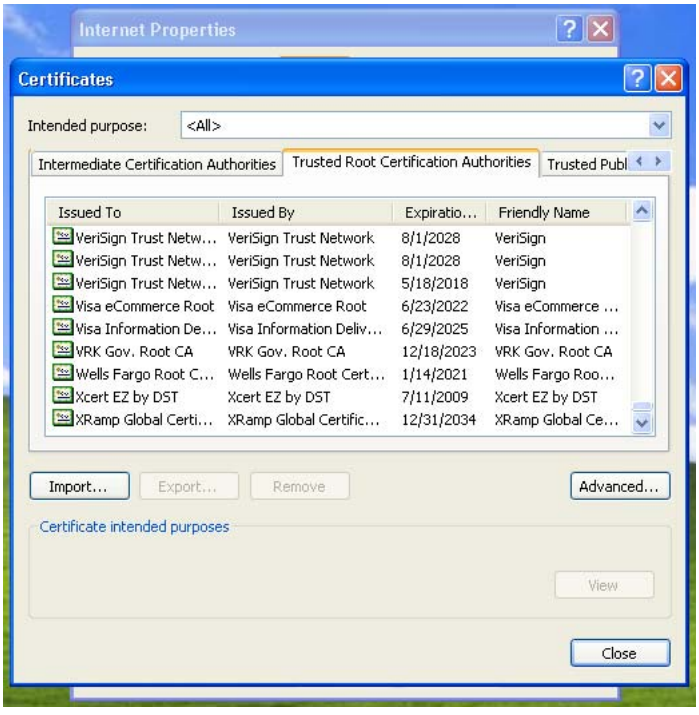

Figure 1.6: Viewing the CAs trusted by Windows.

Each of these "Trusted Root Certification Authorities" has a certificate, including a public key, installed in Windows; the Microsoft Root CA is one that is pre-installed for you, Microsoft having made the decision—to trust Microsoft—on your behalf. The availability of this root CA public key serves as the equivalent to the holographic watermark on a driver's license. Take a look at Figure 1.7, then read the following explanation of how it works.

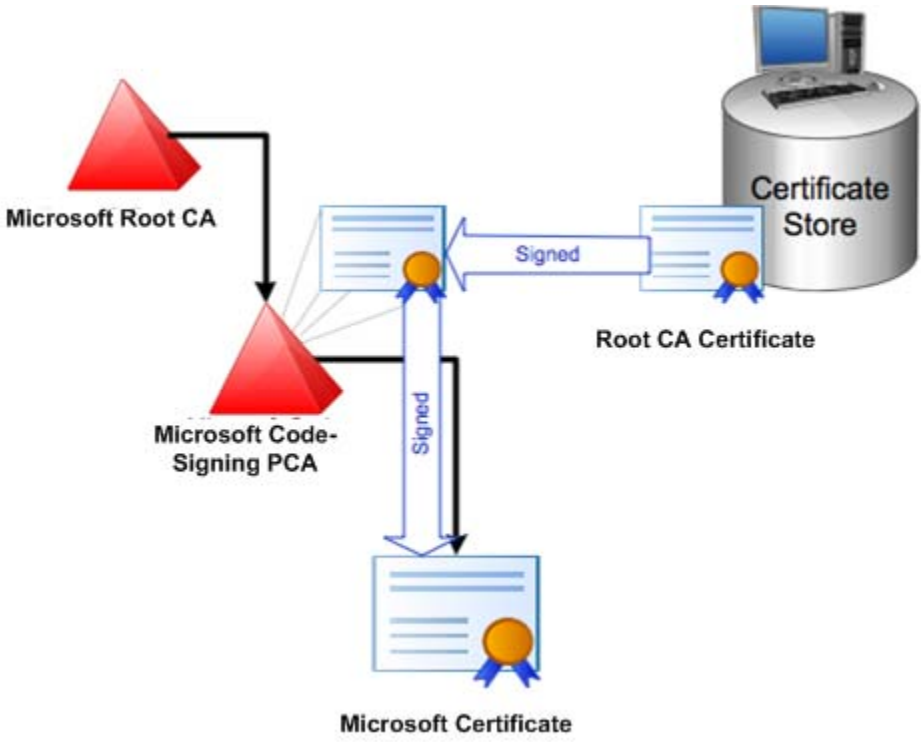

Figure 1.7: Verifying the certification path.

To begin, there's the actual certificate that was used to create the digital signature on the piece of software (signing an email message or any other type of data works exactly the same way). This certificate contains a public key, which enables the computer to verify the signature; but how can the computer tell whether the certificate was *really* issued by the CA claimed?

That CA applied its own digital signature to the certificate, using the CA's own private key. The certificate also includes the CA's certificate so that the CA's public key is available and can be used to decrypt and verify the signature on the certificate. In Figure 1.7, the Microsoft Code Signing PCA has signed the Microsoft Certificate, shown by the downward-pointing "Signed" arrow. But how do you know that the CA's certificate is valid?

The Microsoft Code Signing PCA was authorized by the Microsoft Root CA, which used its private key to digitally sign the Microsoft Code Signing PCA's certificate. Because your computer trusts the Microsoft Root CA, it can obtain the CA's certificate and public key from its local certificate store. It uses the root CA's public key to decrypt and validate the signature on the Microsoft Code Signing PCA certificate, validating its identity. This all worked because ultimately your computer had one piece of independently-provided information—the root CA's certificate.

This is a *really* important set of concepts, so let's run through the example again, substituting numbers for names to perhaps make things easier to follow. Start with Figure 1.8.

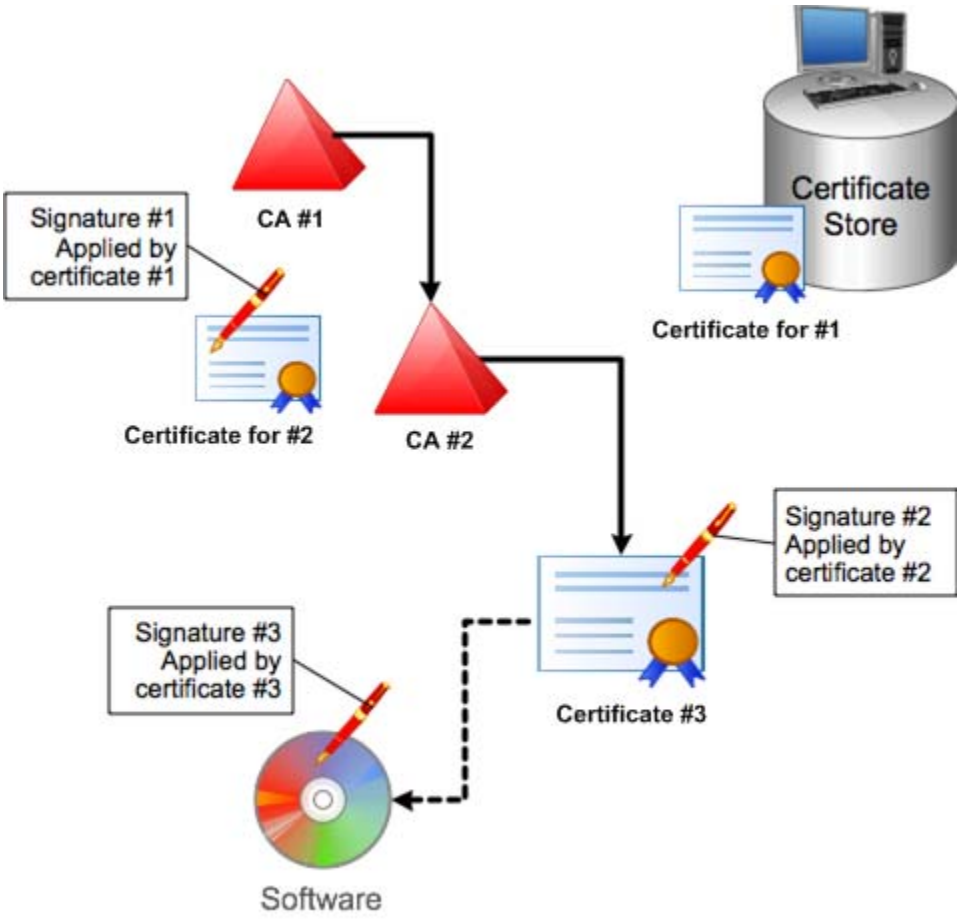

Figure 1.8: Reviewing the certification path, redux.

1. We start with our piece of software at the bottom. It has a digital signature, which we verify by using the public key from certificate #3. But how do we know that certificate #3 is valid?
2. Certificate #3 contains a signature from its issuing CA, CA #2. We verify that signature by using the public key from certificate #2, which belongs to CA #2. But how do we know that certificate #2 is valid?
3. Certificate #2 contains a signature from *its* issuing CA, CA #1. We verify that signature by using the public key from certificate #1, which we already had on file locally because we trust CA #1. That's how we know certificate #1 is valid.

So the validity of certificate #1—which is a *self-signed* certificate, meaning it attests to its own authenticity and it's up to us whether to trust it—leads us to the validity of certificate #2, which leads us to the validity of certificate #3, which leads us to accept (and trust) the signature on the software. In other words, our trust in CA #1 leads us to ultimately trust the piece of software—because we trust its developer, and the certificate chain lets us know that our trusted developer did indeed provide this software.

But What Do We Trust?

When we say that we *trust* a CA, what we're saying is that—like the Department of Motor Vehicles example—we trust the process that the CA uses to verify people's identities. In other words, if we get a piece of software that's signed by the ACME Software Company, and the certificate was issued by the Super CA Company, we're saying that we believe the Super CA Company did a good job of verifying the identity of ACME before giving them a certificate that attests to that identity. Because we trust the CA, we also trust that the software did indeed come from ACME—and if we think ACME is a good company, the software is safe to run. The signature guarantees that the software is exactly as ACME intended, so if it *is* malicious, we know that it's ACME's fault, and we can take appropriate actions.

> We'll spend more time in Chapter 3 examining the issue of trust, what goes into it, and what kinds of decisions you should be making for your business with regard to trust.

Certificate Life Cycles: Issue, Renew, Revoke

Certificates—not unlike driver's licenses—are living things. By that, I mean that you don't simply hand out a certificate and forget about it: Certificates have a lifespan, meaning they expire. That implies the need to renew a certificate that is still needed. And, of course, certificates can be lost, stolen, or otherwise compromised, which implies the need to revoke certificates and inform anyone who may have been using or relying upon the certificate of its revoked status.

Issuing Certificates

If you can remember applying for your first driver's license, you know that issuing a certificate for the first time isn't necessarily the easiest phase in the certificate's life cycle. Depending upon the *type* of certificate you're applying for, and from where you're obtaining the certificate, there may be significant and thorough investigation required to validate your identity. Different CAs—including both commercial CAs and those that are managed internally by companies—have different procedures; in fact, these differences are primarily what sets CAs apart from one another. In general, though, CAs typically recognize different classes of certificates, with each successive class representing a situation that is potentially more damaging, and therefore worthy of more thorough identity verification.

Those classes were originally introduced by commercial CA VeriSign and are:

- Class 1 certificates are issued to individuals and are used to encrypt or sign email. Commercial CAs may only verify that the requestor has control of the email address to which the certificate is issued. However, enterprises may also use Class 1 certificates for authentication, and may require in-person issuance of the certificate in order to confirm identity.
- Class 2 certificates identify entire organizations.
- Class 3 certificates are typically used for servers and for software signing, which—if the certificate's identity isn't well-validated—could potentially affect a lot of people. These certificates often require independent verification of identity, such as through organizations like Dun & Bradstreet.
- Class 4 certificates were created for online business transactions conducted between companies.
- Class 5 certificates were created for private organizations and governmental security.

Today, these exact classes are less important than the *use* for which a certificate is issued: Every certificate is encoded with these uses, and different uses commonly demand more thorough identity verification. Certificates today may be referred to by their purpose, such as an SSL certificate, a code-signing certificate, a personal certificate, and so forth. The important thing to recognize is that the certificate issuance process *must* differ at least somewhat depending on the use of the certificate.

There are parallels in the issuance of real-world certificates such as driver's licenses. A typical driver's license requires a fixed set of processes and procedures: Perhaps a driver's test, an eye test, and so forth. However, some licenses are used for additional purposes: driving a school bus, driving a commercial truck, or driving a motorcycle. Because these additional purposes carry additional risks and responsibilities, the issuance process varies accordingly.

Renewing Certificates

Renewing a certificate, as with renewing a driver's license, is a significantly simpler process than issuing a certificate. Typically, a certificate can be renewed through the CA's Web interface. Renewing essentially re-creates the same certificate with a new expiration date.

Why do certificates need to expire? An infinite certificate lifespan would remove the need to renew entirely, right? It would, but an expiration date gives the CA the opportunity, if needed, to review the certificate and make sure, for example, that the entity to whom the certificate was issued still exists (in the case of a business, it might not). Expiration also ensures that a compromised (stolen or lost) certificate can only be used for a limited period of time before it expires.

Even root CA certificates periodically expire, meaning they must be renewed and their certificates re-issued to people who trust that CA. Although root CA certificates are extremely unlikely to be compromised in most cases, expiration offers another benefit—the opportunity to upgrade encryption technologies and to remove from use older technologies that might be more easily broken or compromised by newer computers and attackers.

Revoking Certificates

When the geniuses who created digital certificates were first doing so, everything probably seemed easy (except maybe for the encryption math) until they thought of revocation. Think about it: You've got certificates spread all over the universe, and one of them is stolen—how do you let everyone know?

Certificate Revocation Lists

The first approach, one that is still widely used, is a Certificate Revocation List (CRL—often pronounced *krill*). When a CA revokes a certificate, they publish a CRL—essentially, a list of revoked certificates—to a predetermined location. Each root CA certificate can contain information about where that CA's CRL is published. If you remember back a couple of decades, this approach is similar to what credit card companies originally used for revoked credit cards: a big list of bad card numbers. Users were expected to scan through this list each time they accepted a card and to decline cards found on the list.

CRLs essentially work the same way: Computer software such as operating systems (OSs) and Web browsers are supposed to download the CRL and check it prior to trusting a certificate. RFC 3280 defines this behavior and defines two potential states for a revoked certificate:

- Revoked, meaning the certificate can never be trusted again
- Hold, meaning the certificate can currently not be trusted but may be trusted again in the future

The CRL also specifies the reason for the revocation; some of the major reasons defined in RFC 5280 include:

- Unspecified
- Certificate was compromised
- The CA was compromised
- The affiliation of the entity to whom the certificate was issued has changed
- The certificate has been replaced
- The entity to whom the certificate was issued has been suspended
- The certificate is on hold

CRLs are typically updated every 24 hours, or when certificates are added or removed from the list. Unfortunately, CRLs from major CAs can become huge—unmanageably huge. To help mitigate this problem, CAs can also produce *delta CRLs,* which contain only updates to an already published CRL. The delta CRL is smaller and requires less bandwidth to download and less overhead to process but puts a bit more work on client computers that have to do all that work. Figure 1.9 illustrates how CRLs are used.

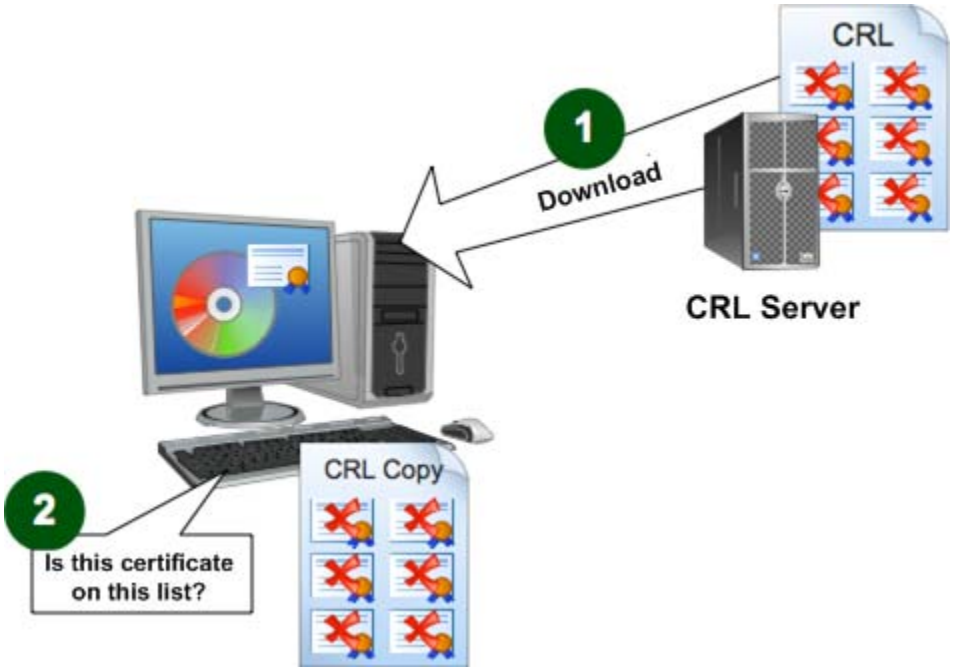

Figure 1.9: Using a CRL.

CRLs also create the opportunity for Denial of Service (DoS) attacks. By maliciously overwhelming the server(s) hosting the CRL, an attacker can prevent legitimate attempts to retrieve the CRL. When this happens, legitimate clients need to decide whether they'll accept a certificate without being able to check the CA's CRL. This decision is often presented to the computer's user, who often makes an uninformed decision that could either potentially compromise their computer's security or could limit their computer's functionality. If the user decides to proceed without the CRL, the user may be trusting a certificate that was revoked; if they don't proceed, they prevent their computer from completing whatever task they were attempting to accomplish.

Once a certificate is added to a CRL, it must generally remain there until the certificate's normal expiration date has passed, at which point the certificate isn't supposed to be used anyway.

In reality, the additional time required to check a CRL means that many clients, such as Web browsers and email clients, don't bother checking by default. Figure 1.10 shows that Internet Explorer 7 on Windows XP, for example, offered two options: One to automatically check for software publisher certificate revocation (for things such as signed ActiveX controls) and another to check for Web server certificate revocation.

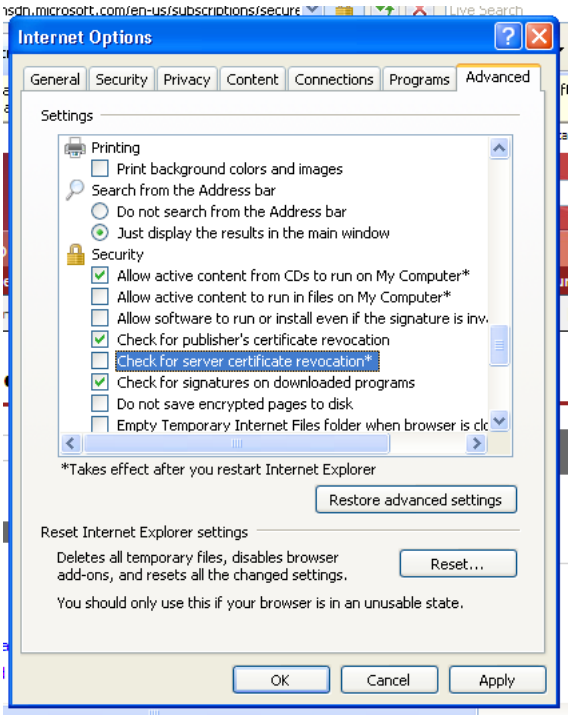

Figure 1.10: Configuring revocation checking in Internet Explorer 7 on Windows XP.

Online Certificate Status Protocol

To help address the shortcomings of CRLs, RFC 2560 created the Online Certificate Status Protocol OCSP). If CRLs are like the old bad-card-number bulletins issued by credit card companies, OCSP is like today's modern, on-demand card verification. OCSP usually occurs over an HTTP connection and allows a client to query the status of a single certificate rather than download an entire CRL and scan through it. Clients typically cache OCSP results so that once a client learns of a revoked certificate, the client will automatically reject it in the future without re-consulting the OCSP server. Responses from OCSP servers are themselves digitally signed, assuring the client that the response came from a legitimate, authoritative server and not from a fraudulent one. Figure 1.11 shows how it works.

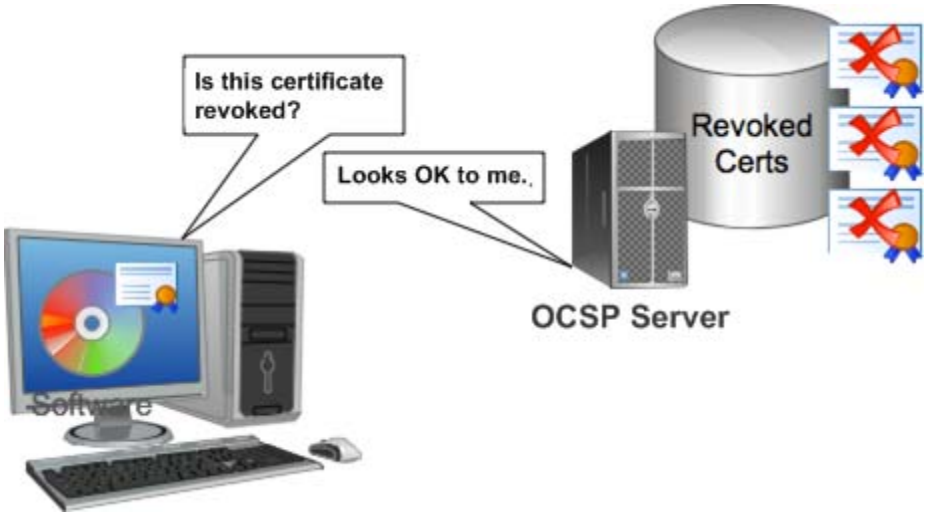

Figure 1.11: Using OCSP.

Note

Technically, OCSP services don't have to be provided by the actual computer that acted as CA and issued the certificate. The entire collection of CAs and other computers that help manage certificates is called a *Public Key Infrastructure* (PKI); although it is possible for a CA to be an OCSP server, it is also possible for a PKI to contain one or more servers dedicated to handling OCSP requests.

OCSP also provides clients the ability to send a *nonce*, or "number used once." This nonce is then included in the encrypted OCSP reply. The idea behind this technique is to prevent someone from capturing and replaying an "all's well" response from an OCSP server; because each client request can contain a unique number and because the client checks each response to make sure it has the expected number, replay attacks are effectively thwarted.

OCSP is relatively new; it was first implemented in Internet Explorer 7 on Windows Vista and Windows Server 2008 (it was not included in Internet Explorer 7 on other versions of Windows); all versions of the Firefox browser support OCSP, although it was not enabled by default until v3; Apple Safari and later versions of Opera also support OCSP. Newer email clients may also implement OCSP support, although it may not be enabled by default. Currently, no shipping OSs implement OCSP for checking software publisher certificate status.

Note

Both OCSP and CRLs require connectivity to the CA that issued the certificates; in the case of commercial CAs that are accessible only via the Internet, both CRLs and OCSP are useless on computers that are not Internet-connected—as is sometimes the case for enterprise servers. In those cases, however, the number and type of software running on the computers is usually very limited, fairly static, and known in advance, making certificate revocation checks less critical.

A newer permutation of OCSP is called *OCSP Stapling*. Because OCSP can itself create high volumes of traffic, stapling was created to help lower some of the traffic and place the burden of providing OCSP responses on the presenter of a certificate. Essentially, the *owner* of the certificate periodically queries an OCSP server to make sure their certificate hasn't been revoked. When they do so, the OCSP server provides them with a time-stamped, signed response. When the certificate is presented to someone, that signed OCSP response is "stapled" to the certificate, allowing the client to see the OCSP response without querying the OCSP server. The client can use the stapled response's timestamp to ensure it is recent, and use the signature to ensure it is genuine.

Note

Privacy advocates like OCSP stapling because it eliminates the need for a client computer to contact a third-party. In the case of Web server certificates, for example, OCSP servers could maintain a log of incoming requests and construct patterns of user Web browsing behavior; by using OCSP stapling, this is not possible.

Stapling is defined in RFC 3546; to date it has not seen wide deployment and is only useful in situations in which certificates are dynamically presented, such as in authentication scenarios. Web server SSL certificates, for example, are a candidate for OCSP stapling because the server presents its certificate to client Web browsers upon connection.

Protecting Your Certificates

Perhaps it should go without saying that private keys must be *rigorously* protected—but let's say it anyway: Keep them *private*. In many uses, certificates are simply used from a file, meaning that file—especially the one containing the private key—may be best stored on a CD-ROM and kept in a locked safe when not actually in use.

Smart cards were invented precisely to secure private keys. The private key is installed into the memory on the smart card. Typically, the smart card is connected to a computer by means of a card reader—often a built-in or USB device. When the private key is needed, the data to be encrypted is transmitted to the smart card and encrypted on the card, and the result is transmitted back to the computer. In other words, the smart card isn't just a fancy, flat storage device; it's actually a small computer of sorts, and it ensures that the private key never leaves the card. The card often requires a PIN or password to be entered before the card can be used, helping to further secure the private key and how it can be used.

Other forms of hardware devices, called *tokens*, can serve a similar function as smart cards. They may connect directly to the computer via USB, and essentially serve as a single-purpose smart card+reader combination device. An advantage of any kind of hardware device is that it can easily be locked in a secure place, such as a safe, when it isn't actively being used. Access to that safe might be controlled by individuals other than those who actually use the key, helping to further prevent accidental or intentional misuse.

If your private key is installed into your computer (this is often the case for Windows developers, for example, who are signing their code), be sure the private key is protected by a strong password. This prevents any use of the key without your knowledge: Any attempt to access the key through Windows' CryptoAPI will result in the appearance of a password-request dialog box.

Particularly sensitive keys—those used by a commercial software vendor to sign production-quality code, for example— may be protected by having more than one person keep separate halves of the private key or by protecting the private key with a password consisting of two parts, each part known to different people. This technique ensures that the key cannot be used without the cooperation of multiple individuals, both protecting it from theft and preventing misuse—either accidental or intentional.

Summary

There you have it: a concise look at what certificates are, how they work, and how they are managed. Whether you plan to use certificates issued from a commercial CA or those issued from an internal PKI, you'll be using these procedures and processes to work with certificates. There are a couple of key things to remember:

- Certificates enable encryption
- Encryption enables identity verification when a trusted third-party attests to the certificate holder's identity
- The usefulness of a certificate ultimately comes down to your trust in the entity who issued the certificate
- When a CA issues a certificate, it is primarily an attestation to the certificate holder's identity
- Certificates are issued for various purposes, and the identity verification process will differ based on those purposes
- Certificates expire, and can be revoked, and there are a couple of ways in which a client can check for revocation
- Certificates must be protected—primarily the private key portion

In the next chapter, we'll start looking at the *many* ways in which certificates can be used. We'll look at not only the technical implementation of these uses but also the business benefits of them. You'll begin to see that certificates have great potential across the entire IT environment to help improve security, stability, and reliability.