
All About Certificates

Contributed by David Torre

From securing web and email transactions to performing crucial authentication services, certificates are used everywhere today. Yet as common as the certificate is, their inner workings are often misunderstood. Learn more about what certificates are, what they provide, and how they actually work.

{mospagebreak title=The Need for Certificates}

The Need for Certificates

SSL or Secure Sockets Layer, in conjunction with digital certificates, are used to assist in the securing of a digital transmission. Before we get too involved with how certificates work, it's imperative to understand what it is you're trying to secure with a certificate in the first place. Perhaps we should start by defining what "security" actually entails.

Most security systems are based on three fundamental components: confidentiality, integrity, and authentication. Confidentiality is usually what people think of when they think of security; it is what keeps a conversation private from unauthorized eavesdroppers. In technology, confidentiality is typically achieved through the use of encryption. Message integrity is a means of ensuring that a given message wasn't tampered with during transit. Integrity is typically achieved with some type of cryptographic function. Finally, authentication guarantees the person or system at the other end of the conversation is actually who they claim to be. This is where digital certificates come into play.

Trust and Authenticity

Simply put, digital certificates provide authentication. In the context of e-commerce, providing the authentication component of a security system certifies to your customers you really are who you say you are. Authentication is essential, but it's important to remember that digital certificates alone do not provide security. (Make this a blockquote)In order to have a secure system, one must not neglect to include confidentiality and message integrity. It's not uncommon to witness IT departments spent large sums of money on SSL certificates, only to install them on web servers which support weak encryption ciphers and antiquated versions of SSL.

{mospagebreak title=How Certificates Work}

How Certificates Work

A digital certificate is similar to a birth certificate. As you know, a birth certificate conveys information such as name, date of birth, gender, and other attributes which ultimately identifies a given individual. Yet somewhere on the certificate the hospital provided a seal of authenticity which, as it turns out, is difficult to imitate. As people inherently trust legitimate hospitals, your birth certificate (with proper seal of authenticity) essentially proves your identity to others.

Digital certificates are similar to birth certificates in that they too provide identifying characteristics, and are ultimately signed by a trusted third party. The real magic behind proving authenticity is achieved with a cryptographic function called a "digital signature." Before we get too deep into certificates and digital signatures, an ultra high-level overview contrasting the two categories of encryption is in order. Don't worry-- we'll keep it simple.

Symmetric Vs Asymmetric Cryptography

Symmetric encryption (also known as "private key" encryption) uses a single key to both encrypt and decrypt data. This single key is to be distributed to both parties wishing to communicate with each other, and must ultimately be kept private. Although relatively fast, private key encryption requires a unique key for every entity you wish to communicate with. (You never want to re-use the same key to communicate with more than one party.) This cumbersome "key management" issue prevents private key encryption from being used exclusively in e-commerce. Imagine needing a unique key for each customer you intent to do business with. For large e-commerce sites with many customers, millions of keys would be required, which is simply isn't feasible.

Asymmetric encryption (also known as "public key" encryption) is encryption which uses one key to encrypt data, and another key to decrypt data. These two keys are referred to as the "public" and "private" keys. While the private key must be kept secret, the other is made public and thus widely distributed. Those who wish to communicate with you would simply use your public key to encrypt the data. As only you

own the corresponding private key, you are the only entity that can decrypt the message. Whether you're communicating with one or millions of customers, only two keys are ever needed. While this drastically simplifies key management, public key encryption is often CPU-intensive and therefore terribly slow.

Digital Signatures

Now that that we've covered symmetric vs asymmetric cryptography, let's get back to digital signatures. A digital signature is a function which starts out by taking a message of variable length, such as the text from web page or an email, and computing a fixed-length “fingerprint” of that message. This is known as a hash code. The important thing to remember about hash codes is that they are one-way functions. So what does that mean? It means that you can take a message of variable length (say 5,000 bytes for example) and derive a unique and static-sized hash code from the message, but not the inverse. In other words, the unique fingerprint or “hash code” cannot be used to produce the original message. The other important aspect of a hash code is that it is intended to be unique per a given message. If two large files which contain the exact same text are individually hashed, the corresponding hash codes should be identical. However, if you were to change just a single character within one of the files, still leaving the thousands of other words unchanged, the hash functions would yield completely different results. For example, let's examine what happens when we take the hash of two files with identical content:

FILE: test.txt

“The quick brown fox jumps over the lazy brown dog”

MD5 Checksum: 6666c5e7269dc93c66df8f797bf961ed

FILE: test2.txt

“The quick brown fox jumps over the lazy brown dog”

MD5 Checksum: 6666c5e7269dc93c66df8f797bf961ed

Notice that two different files with the same textual content yield the same cryptographic fingerprint. However, let's see what happens when we change only a single character in file “test2.txt”

FILE: test.txt

“The quick brown fox jumps over the lazy brown dog”

MD5 Checksum: 6666c5e7269dc93c66df8f797bf961

FILE: test2.txt

“The quick Grown fox jumps over the lazy brown dog”

MD5 Checksum: 627c8ceb4fdc48cac0f950858397353b

As you can see, changing only a single character (brown to Grown) radically alters the fingerprint. Often times, a hash code or “checksum” alone is used to convey authenticity. For example, software distributors who offer digital downloads of their products will publish checksums of the corresponding files on their web site to ensure to customers that the files haven't been tampered with, or perhaps become corrupted during the download. The problem with solely depending on a checksum for authenticity is that if an attacker successfully broke into your web server and altered your download, why wouldn't he simply alter the checksum as well? While checksums are useful, generating a unique signature of a file does not ensure security. While the checksum may be mathematically correct, we still can't prove who generated it.

A digital signature takes the concept of a checksum one step further. Like a checksum, a digital signature starts out by deriving a fixed-length fingerprint from a message. However, to prove that you and only you generated the hash code, we must electronically “sign” the hash code. This is done by encrypting the hash code with a secret encryption key. Technically speaking, this is the corresponding private key from an asymmetric public/private key pair.

At this point, we now have a working digital signature which can be distributed along with our data to ensure authenticity. To verify the authenticity of the original message, the receiver would first generate their own hash code of the data, using the same algorithm we originally used. The receiver would then use our widely distributed public key to decrypt our encrypted (or “signed”) hash code. Remember that in asymmetric cryptography, one key encrypts while the other decrypts. As such, the “public key” is able to decrypt the hash code which was encrypted by the highly secret “private key.”

Finally, if the decrypted hash code matches the hash code generated by the receiver, the message should be deemed authentic. If the hash code differ, then something went wrong along the way and the resulting message should not be considered legitimate.

Distributing Your Digital Signatures

Now that we know how to prove authenticity through digital signatures, we need to properly distribute our digital signature and public key in a standardized format. This is accomplished by encapsulating the digital signature and public key in what is known as an X.509 certificate. An X.509 certificate consists of your identifying attributes such as company name, a common name (www.acme.org for example), and finally your widely distributed public key. An actual X.509 certificate would appears as follows:

Certificate:

Data:

Version: 1 (0x0)

Serial Number: 1 (0x1)

Signature Algorithm: sha1WithRSAEncryption

Issuer: C=US, ST=California, L=El Granada, O=Atomic Fission Certificates,

OU=Certificate Authority Division,

CN=certificates.atomicfission.com/emailAddress=info@atomicfission.com

Validity

Not Before: Jul 13 17:36:34 2007 GMT

Not After : Jul 12 17:36:34 2008 GMT

Subject: C=US, ST=New York, L=lthaca, O=Acme Burritos 123, OU=IT Division,

CN=www.acmeburritos123.com/emailAddress=info@acmeburritos123.com

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:c5:6f:5d:56:fe:6e:83:b1:4d:4c:84:65:a2:f1:

fb:93:0c:28:ba:b2:dc:d2:4d:46:89:83:f9:0d:c3:

57:a9:ad:30:a5:ce:76:52:76:82:76:73:34:b3:eb:

33:77:cd:82:22:be:56:4e:01:6c:c4:91:5f:46:b3:

97:fa:b1:e8:5a:a9:a4:29:a4:03:32:9a:a4:cc:8f:

25:eb:42:b0:48:ba:96:57:ad:f1:c8:0e:5a:11:64:

f8:ba:d7:35:1d:fa:50:8b:fb:f7:98:52:88:63:b6:

3c:61:3f:9a:0a:72:04:4f:09:85:31:c3:14:7e:93:

90:b0:e8:b7:54:2d:47:d9:87

Exponent: 65537 (0x10001)

Signature Algorithm: sha1WithRSAEncryption

be:01:54:79:46:2b:5c:3f:9e:1c:e4:97:d6:1c:68:69:93:a2:

b9:92:69:93:a9:33:21:c4:d4:7d:f5:71:f2:57:66:be:4a:90:

3c:c5:40:ef:ba:02:96:e3:f6:50:f0:7b:28:b2:33:6a:0c:66:

ac:53:3c:6f:25:10:03:79:e2:81:1a:3e:37:d2:d2:b9:84:f6:

c0:76:8b:fc:70:b3:0d:d2:b0:79:5b:6c:a2:89:4c:84:9f:92:

ed:bd:4e:24:7a:2f:8b:12:8d:9f:86:62:bb:22:b0:7a:41:bc:

f8:a3:92:db:0e:cd:1c:43:00:b4:91:a1:64:5f:4f:4d:b8:de:

b1:00

Once your identifying attributes and public key are neatly packaged into a properly formatted X.509 certificate, the entire certificate gets its own unique fingerprint via a checksum routine. This checksum is then "signed" by a trusted certificate authority (CA) such as Verisign or Thawte. The certificate authority uses their private key to sign (or essentially encrypt) your certificate's fingerprint. Your customers will use the CA's widely distributed public key to decrypt the signed fingerprint in order to determine your authenticity. If the fingerprint decrypts properly, you can safely assume that the signature was signed by a trusted certificate authority. For our certificate above, the certificate authority used the SHA1 checksum algorithm to create a fingerprint of the X.509 certificate. The checksum was then encrypted using RSA encryption:

Signature Algorithm: sha1WithRSAEncryption
be:01:54:79:46:2b:5c:3f:9e:1c:e4:97:d6:1c:68:69:93:a2:
b9:92:69:93:a9:33:21:c4:d4:7d:f5:71:f2:57:66:be:4a:90:
3c:c5:40:ef:ba:02:96:e3:f6:50:f0:7b:28:b2:33:6a:0c:66:
ac:53:3c:6f:25:10:03:79:e2:81:1a:3e:37:d2:d2:b9:84:f6:
c0:76:8b:fc:70:b3:0d:d2:b0:79:5b:6c:a2:89:4c:84:9f:92:
ed:bd:4e:24:7a:2f:8b:12:8d:9f:86:62:bb:22:b0:7a:41:bc:
f8:a3:92:db:0e:cd:1c:43:00:b4:91:a1:64:5f:4f:4d:b8:de:
b1:00

Key Strength

Simply put, encryption "bit lengths" represent the number of possibilities for a given key. For example, if your encryption key is "Hi-YourIT182" and your encryption algorithm was a 64 bit cipher, then it would take up to 264 attempts, or about 18 quintillion (18,446,744,073,709,551,616) tries to systematically guess every potential password combination before correctly guessing the phrase "Hi-YourIT182".

You may often hear computer professionals assert that 128 bit encryption is "strong" encryption, while anything less than 128-bit encryption should be considered "weak" encryption. Generally speaking, this is not an accurate statement due to the fact that not all encryption ciphers are created equally. Recall that there are two categories of encryption: "symmetric" and "asymmetric." A 128-bit (or greater) encryption key length is quite strong for a symmetric encryption algorithm such as 3DES, RC4, or AES. However, a 128-bit key

size for an asymmetric algorithm such as RSA or DSS would be dangerously short. Unlike their symmetric counterparts, asymmetric algorithms require key sizes of at least 512 bits. In fact, most security professionals will not implement asymmetric algorithms with a key size less than 768 bits.

Symmetric encryption is fast, yet it requires a unique encryption key for each individual you plan on communicating with. For web sites which have numerous customers, attempting to securely create and issue a unique key for each customer would be incredibly time consuming. On the other hand, asymmetric cryptographic solves the key exchange issue by issuing everyone only a single public key which is to be used for encryption. The encrypted messages are then decrypting using a single secret key which only your server has access to. The downside to asymmetric cryptography is that it is very CPU intensive and thus far slower than symmetric encryption.

In an attempt to have the best of both worlds, the SSL protocol takes a hybrid approach to encryption. Specifically, asymmetric encryption is used only initially in order to exchange certificates and agree on a (temporary) shared key. Once the SSL session is established, asymmetric encryption ceases and symmetric encryption commences using the recently created shared key.

In most web server environments, you and your certificate authority must agree on the asymmetric (or "public") key size. This is because your certificate authority is the entity that actually converts your certificate signing request (CSR) into an actual, usable certificate. Although the CA will determine a public key length, customers often have a say in the matter, and tend to standardize on 1024-bit RSA. The symmetric (or "private") ciphers are completely within your control as they're configured within your web server's properties. By default, web servers will support a vast array of public/private encryption combinations. These combinations, known as "cipher suites," are presented to web clients in hopes that the server and client can agree on one combination in particular. In SSL v3 and TLS v1, the client is allowed to suggest cipher suites, but the server has the final say as to which cipher suite is actually used. Therefore, security-conscious sites should remove any weak or "export grade" cipher suites from the server altogether. This is the only way to ensure that only strong encryption will be used in SSL sessions.

A primer on certificates wouldn't be complete without at least a parenthetical note regarding United States encryption export law. Prior to January of 2000, the United States Bureau of Export Administration restricted the export of "strong" cryptography to countries outside the United States. Although the restrictions were incrementally reduced over time, the historical consensus was that nothing over 40-bit symmetric/512-bit asymmetric could be exported. As of today, strong encryption may be exported to many foreign nations, as long as they are not an embargoed country. Although many years have passed since the US "liberalization" of encryption export law, the concept of "export-compatibility" is still a feature-set found in many encryption applications. Essentially what this means is that a cryptographic application that supports "export grade" ciphers will fundamentally support weak ciphers. With protocols such as SSLv2 the client, rather than the server, may negotiate which cipher suite to be used.

Consequently, hackers may exploit situations such as this to “downgrade” encrypted sessions in order to make cracking the session far easier. Given the fact that computing power has increased significantly over the last decade, weak or “export-grade” ciphers may now be cracked fairly quickly and must therefore never be used under any circumstance.

Certificate Authorities and Public Key Infrastructure

A certificate authority is analogous to the traditional Better Business Bureau. A CA serves as a trusted third-party that can testify to the validity of business when you alone cannot. Certificate authorities are hierarchical, meaning they can either sign client X.509 certificates directly, or they can sign a lower-tiered certificate authority's certificate, which effectively grants the lower-tiered CA the ability to sign client certificates directly. This hierarchical system of trusted certificate authorities forms what is known as the public key infrastructure, or PKI.

At this point, you may be wondering how your computer will inherently trust a certificate signed by an authority such as Verisign, while completely rejecting lesser-known CAs such as “Acme Certificates” or similar. The method by which your computer either trusts or rejects a CA is application-specific. Microsoft Windows for example, provides a centralized root certificate repository which is intended to be queried by all Microsoft-based applications. Although infrequent, new root CAs are pushed to Windows systems via standard Windows Updates. Cross-platform applications on the other hand, such as Mozilla Firefox and Sun Microsystems' Java Runtime Environment, cannot depend on such platform-specific certificate repositories and must therefore depend on their own built-in cache of trusted root certificate authorities. Application-specific management of trusted root CAs is fairly transparent to the average end-user. Just note, however, that this situation can prove arduous for organizations who wish to build their own internal trusted root certificate authorities. This is because the newly created root CA certificate must now be added to several different repositories on each computer!

{mospagebreak title=Various Certificate Types}

Various Certificate Types

Any certificate you purchase from a trusted root certificate authority should conform to the X.509 standard described in the previous section. Nevertheless, certificate resellers often use ambiguous terminology to categorize the different types of certificates they provide. Marketing terms such as “turbo” or “extended-validation” mean little to those who aren't familiar with certificates, and in the end just convolute the purchasing decision. Therefore, let us take a moment to briefly cover the various types of certificates by their associated (non-standard) nicknames.

Standard Certificates

A certificate authority's primary role is essentially to vouch for your company's authenticity to non-trusting customers. While your customers may have never heard of your organization, they inherently trust a set of pre-installed root certificate authorities. If you can successfully influence one of those root CAs to trust your business, then all clients associated with that particular CA will in turn trust you as well. However, gaining the trust of a root CA is not always trivial.

A standard certificate may be viewed as a "mid-level" assurance certificate due to the average level of documentation an organization must provide during the verification process. The process of issuing a standard certificate typically starts out with CA verifying your organizational structure from two or more public databases-- usually WHOIS and Dun and Bradstreet. From those two sources, the CA can ascertain both the technical contact for your domain name, as well as administrative contacts such as a chief financial officer or human resources director.

Once the CA locates your organization's official headquarters through a trusted third party, they will resume subsequent communications through that location. For example, the root CA may send a verification form to your organization's fax number which was listed by Dun and Bradstreet. The CA may additionally request further proof of organization by requesting a copy of your organization's Articles of Incorporation, or proof of 501(c)(3) status for non-profit organizations. Once all appropriate documentation has been verified, the CA will finally sign your organization's X.509 certificate, thereby declaring your corporate validity.

Domain-Validated Certificates

While many certificate authorities insist on the systematic steps involved in the issuance of a "standard" certificate, many potential certificate customers feel as though this degree of verification is unnecessarily stringent, and perhaps the overall process is just too complex to accommodate. While some CAs balked at the idea of lowering their strict verification standards, other CAs viewed this as an opportunity to create a new "simplified" type of certificate which allows for verification to be achieved in far fewer steps.

Domain-validated certificates are marketed under several vendor-specific monikers. While GoDaddy refers to them as "Turbo" certificates, Thawte refers to the very same type of product as the "SSL123" certificate. Regardless of the name, the term "domain-validation" says it all-- if you can successfully validate your domain, the CA will issue you a signed certificate. The process begins with the CA performing a "WHOIS" lookup on your domain name. Once the individual with administrative and/or technical control over the domain is located, a simple verification email is sent to that individual. The verification process is completed once that individual clicks on a special activation hyper link embedded within the email message. Obviously, domain-validated certificates are issued far faster, and often at a much lower price than traditional "standard" certificates. The down side of course, is that if an attacker was to somehow gain access to the email box of your domain's designated contact, it would be trivial for the hacker to generate a certificate on your company's behalf.

You may be wondering if domain-validated certificates are too good to be true. The reality is that any certificate signed by a trusted root certificate authority will essentially provide the necessary authentication of your business. While a technically discerning web visitor may be able to pick apart your certificate and determine the exact level of assurance, the vast majority of your visitors will not. A domain-validated certificate will certainly provide the trusted SSL “padlock” icon within your clients' web browser windows, which should give them the assurance they need to do business with your site.

Extended-Verification Certificates

Thus far, we've covered the verification process of the low-assurance “domain-validated” certificate and the mid-level assurance “standard” certificate. Extended-verification certificates on the other hand boast the highest level of assurance by performing several steps above and beyond standard verification. The specifics of extended-verification vary somewhat among the various CAs. Nevertheless, the extended-verification process strives to verify the overall legitimacy of a business through the collection of additional documentation, and perhaps through the collection of external references. Extended-verification certificates are often expensive, and are typically offered exclusively to incorporated entities.

As described previously, any certificate signed by a trusted root certificate authority will ultimately yield the protected SSL “padlock” icon within web browser windows. As such, prospective certificate buyers may view the additional cost and administrative burden of extended-verification certificates unnecessary. However, some of the newer web browsers do in fact give end-users visual queues as to which sites use extended-verification certificates. If such functionality becomes commonplace, extended-verification certificates may be worth the extra effort.

Wildcard Certificates

While standard certificates are intended to protect a single hostname, wildcard certificates may be viewed as “blanket certificates” which essentially protect a pattern of hosts within a given domain. When purchasing a certificate, the certificate authority will ask for a “common name” which must match your primary domain name. For example, the “common name” field for Amazon books would simply be “www.amazon.com.” This effectively protects the server named “www” which lives within the “amazon.com” domain. Although this approach may suffice for most web sites, this method requires the creation of a new certificate for each unique hostname within your domain. In other words, you would need a distinct certificate for “www.yourdomain.com,” another for “email.yourdomain.com,” yet another for “shopping.yourdomain.com,” and so on. A convenient alternative to single-host certificates is the wildcard certificate. The wildcard certificate allows you to substitute any one field of the hostname with the special star (*)

character, which fundamentally matches any string. Therefore, a common-name of `*.atomicfission.com` would match `www.atomicfission.com`, `mail.atomicfission.com`, and `forums.atomicfission.com`.

Note however, that you may substitute one, and only one, component of a hostname. For example, let's say your organization has two domains: one on the east coast (`east.yourcompany.com`) and the other on the west coast (`west.yourcompany.com`). If you wanted to deploy wildcard certificates, you would actually need to purchase a wildcard certificate for each subdomain-- one for the east coast office (`*.east.yourcompany.com`) and a second for the west coast office (`*.west.yourcompany.com`). If you were to purchase only one wildcard certificate (`*.yourcompany.com`), then perhaps deploy a new mail server on the east coast (`email1.east.yourcompany.com`), clients would receive certificate mismatch errors. This is due to the fact that your star character (*) is living within the the third label from the right. Since the server you want to protect (`email1`) is located within the fourth label from the right, the wildcard star character no longer matches that particular host living within the given domain name.

Aside from the special common-name field which should contain a special star character, the process of acquiring a wildcard certificate is identical to that of acquiring a standard certificate. Just note that the added convenience of a wildcard certificate does come with a higher price tag. As you are protecting more than one host within a given domain, expect to pay a bit more for this type of certificate.

Chained Certificates

A certificate-aware client application determines whether or not to trust your server's certificate by first consulting with its own local key store. If the client possess a corresponding public key from the certificate authority that signed your certificate, then the client should inherently trust the validity of your certificate. As an example, let's assume you purchased a certificate from Verisign. If a Firefox user then requests your certificate, Firefox will check its local key store to determine if it has a copy of Verisign's widely distributed public key. Since Verisign is a longstanding certificate provider with considerable tenure in this line of business, Firefox does indeed ship with the Versign public key by default. Therefore, there should be no issues with Firefox trusting your certificate in this scenario. This is obviously great news for Verisign customers, but what happens if you purchase a certificate from a newer, or perhaps lesser-known certificate authority?

Recall that certificates are published through a system known as the public key infrastructure. Fortunately, PKIs are hierarchical, which allows for `root` certificate authorities such as Verisign to give other lower-level `intermediate` certificate authorities the ability to sign certificates themselves. Binding an intermediate certificate authority to a root certificate authority is known as `certificate chaining`. The overall links of the chain create what is know as the `certification path`. The most important aspect of a certification path is that it must ultimately end with a root level certificate authority that is inherently trusted by clients. If PKI newcomers such as GoDaddy can acquire the trust of an existing and well-established root certificate authority, then GoDaddy and other resellers can in turn begin to sell certificates directly to customers. This creates competition amongst the various certificate vendors, which then of course equates to better prices for the consumer.

Chained certificates are often far less expensive than certificates offered directly by root certificate authorities. For the cost-conscious, this presents an attractive alternative to an otherwise expensive technology. Nevertheless, chained

certificates must be chosen carefully. The intermediate certificate authority issuing your chained certificate must be trusted by a root certificate authority themselves. If the intermediate certificate authority isn't trusted (either directly or indirectly) by a root CA trusted by the client, then the certification path will fail.

Self-Signed Certificates

Self-signed certificates are used in conjunction with privately-owned, internally managed public key infrastructures. In addition to SSL-enabled intranet applications, X.509 certificates may also be used within applications such as secure SMTP or wireless (WiFi) authentication. Self-signed certificates work by first creating an internal root certificate authority. The public keys for this internal root CA must then be pushed to all clients within the organization. (This requires administrative access to the machines.) Once the internal root CA is trusted by all clients, application servers may then commence using certificates signed by the internal root CA. Ongoing key management is typically handled by an enterprise-class certificate management system such as Red Hat's Certificate System or Microsoft's Windows Server 2003 Certificate Services. The most important item to note with respect to self-signed certificates is that the internal root CA certificate must be forced onto client computers. As you cannot force the installation of internal CAs on remote / unmanaged client computers, using self-sign certificates on public web sites will certainly result in errors on the client side of the transmission.

Certificate Formats

While a single certificate encoding standard would be convenient, the reality is that there are several formats to choose from. If you plan on installing only a single certificate on a single server, you can probably get by with following your vendor's instructions. If you plan on managing multiple certificates across several different platforms, you may want to take a brief look at the various certificate formats as you'll certainly encounter them sooner or later.

PEM Format – Oddly enough, the PEM format originated from a standard which never really caught on--“privacy enhanced mail.” PEM is text-based encoding format, which allows one to copy and paste certificates, as well as transfer certificates over text-based protocols such as e-mail, HTTP, FTP, and others. PEM is the default certificate format for many UNIX-based applications such as Apache and OpenSSL. PEM certificates may contain a single public key, a single private key, a single certificate, or any mixed combination thereof. Common file extensions include: *.key for private keys, *.crt for certificates, or simply *.pem. Two other types of certificate files commonly encoded in PEM format are certificate signing request (CSR) files, and certificate revocation list (CRL) files.

DER Format – The ASN.1 “distinguished encoding rules” format is a headerless, binary format. Like the PEM format, a DER-based file may contain any one key/certificate, or a combination of all three. DER-encoded files typically have .der or .cer extension. The DER format is commonly used in Java-based applications.

PKCS/PFX – Personal inFormation eXchange is a Microsoft-based protocol which is compatible with RSA's Public Key Cryptography Standard. Being a Microsoft-based protocol, it is most common in (you guessed it!) Microsoft products

such as Windows and Internet Explorer. PFX files typically contain one or more encrypted keys, as well as basic X.509 certificate information. Common file extensions include *.pfx and *.p12.

{mospagebreak title=Acquiring and using your Certificate}

Acquiring and using your Certificate

Once you've chosen a certificate authority, you must submit a standardized request to the CA in order to obtain your certificate. This standardized request is known as a "certificate signing request" or CSR. The CSR phase starts out with generating a cryptographic public/private key pair. As their names imply, the private key is kept secret, while the public key is widely distributed.

Your public key is bundled with other identifying information such as your company name, company address, and web site URL. This bundle is then "signed" with your (secret) private key, and finally sent off to your certificate authority of choice. A typical CSR is shown below:

```
-----BEGIN CERTIFICATE REQUEST-----
MIIB7zCCAVgCAQAwga4xCzAJBgNVBAYTAIVTMREwDwYDVQQIEwhOZXcgWW9yazEP
MA0GA1UEBxMGSXRoYWNhMR0wGAYDVQQKExFhY211IEJ1cnJpdG9zIDEyMzEUMBIG
A1UECXMlSVQgRG12aXNpb24xIDAeBgNVBAMTF3d3dy5hY211YnVycml0b3MxMjMu
Y29tMScwJQYJKoZIhvcNAQkBFhhpbmZvQGZjWVidXJyaXRvczEyMy5jb20wgZ8w
DQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMVvXVb+boOxTUyEzALx+5MMKLqy3NjN
RomD+Q3DV6mtMKXOdIj2gnZzNLPrM3fNgik+Vk4BbMSRX0azl/qx6FqppCmkAzKa
pMyPJetCsEi6llet8cgOWhFk+LrXNR36Uiv795hSiGO2PGE/mgpyBE8JhTHDFH6T
kLDot1QtR9mHAgMBAAGgADANBgkqhkiG9w0BAQUFAAOBgQBYjFVdJLyZO+5gE1U
HWv//FbUUG2Go/Wf7+/J0PkvwkAa2VZv5pNWXNJkp3rQgpdk9u4w337du4Q3B5mF
UPLb1+lmHYXJ0kLspTMX3oGU4gRdPfhGjeljCY93x4QpfgylZOB++yI8uTYZvaSK
xFVAkcajxi1TlxFAF2qzopINw==
-----END CERTIFICATE REQUEST-----
```

Upon receiving your CSR, the certificate authority will then begin the verification phase of the process. During this phase, the CA will attempt to verify the legitimacy of your business through various methods. In some cases, verification may be as simple as replying to an automated email, or as complex as faxing numerous documents providing proof of organization. Ultimately, you must choose the right balance between security and convenience for your organization.

Once the certificate authority successfully verifies the legitimacy of your business, they will then issue you an actual X.509 certificate to be used for SSL/TLS communications on your server. The key point here is that the certificate has been signed with the CA's private key. As the computer systems used by your customers will have the corresponding CA's public key installed, they will deem your certificate as authentic and will therefore trust that the given CA legitimately signed your certificate. Once installed on your server, your digital certificate will be presented to visitors to prove the fact that the legitimacy of your business has been verified by a trust third party.

In addition to verifying the authenticity of your business, the certificate also provides a vehicle for the delivery of your public cryptographic key to the customer. Once the customer has your public key, he or she will use this key to commence secure (encrypted) SSL sessions. As mentioned earlier, asymmetric cryptography helps to solve the key distribution dilemma by securely distributing keys to potential customers. The unfortunate side-effect of public key cryptography is that it's quite slow. In order to provide balance between secure key distribution and performance, SSL starts out by building a secure channel using public key cryptography, but then switches to private key crypto once a secure channel has been established. For example, a typical SSL session may start out using 768-bit RSA, then switch to 128-bit RC4.

Customers commencing new secure sessions with your server will use your server's public key to encrypt all data. Your server will then use the highly secret “private key” to decrypt all data. Even though customers may encrypt data with your public key, only the private key holder is capable of decrypting the information. Therefore, securing your private key is of utmost importance. Keeping your private key on a secured file system in which only proper personnel have access to would be the first step in keeping your key safe. Additional steps may include storing the key on an encrypted file system, or securing the key file itself with a passphrase.

If your private key is ever lost, the corresponding public key becomes useless as you may no longer decrypt data from public key wielding customers. If your private key is compromised by an intruder, communications may no longer be deemed confidential as the intruder may use tools such as `ssldump` to decode previously recorded encrypted communications. In either scenario, the certificate must be immediately replaced using a new public/private key pair. You may optionally place the old certificate in a special list which conveys to customers that it must no longer be used. This list is known as a certificate revocation list, or “CRL” for short. Well-written SSL and TLS applications should examine relevant CRLs using protocols such as OCSP prior to using a server's public key.

{mospagebreak title=Certificate Limitations}

Certificate Limitations

Although certificates lend assistance in providing secure communications over untrusted networks, they're far from perfect. As mentioned in the previous section, the integrity of a certificate depends on keeping the private key completely secure. The compromise of a server's private key equates to communications which may no longer be considered secure.

If loosing a single server's private key is distressing, then the loss of a certificate authority's private key is certainly catastrophic. Recall that certificates, as well as the public key infrastructure (PKI) that supports them, all work on a general system of trust. If a customer trusts a given certificate authority such as Verisign, and Verisign trusts or “vouches” for the authenticity of your server, then the client will inherently trust your server. Just as in the

single server scenario, the mechanics behind this system of "vouched trust" hinge on keeping the CA's private key highly secure. If a hacker was to compromise the CA's private key, she may use the private key to engage in various menacing activities, including signing certificates for illegitimate sites, or even decrypting previously recorded encrypted communications. Once a CA's private key is compromised, all clients of that CA must have a new public key re-issued. As you can imagine, having to immediately re-issue public keys to several hundred thousand or more clients is no simple task.

One of the biggest gotchas related to certificates lies not within the X.509 certificate standard itself, but rather one of the most prominent certificate applications-- the Secure Socket Layer protocol. As any web server administrator will explain, one of the woes associated with hosting SSL-enabled web sites is the requirement for each site to have its own unique IP address. Unlike standard (non SSL) virtual hosts which may reside on one server and ultimately share a single IP address, SSL sites do in fact require dedicated IP addresses. In short, SSL establishes a secure communications channel using what is known as a "handshake" prior to exchanging any HTTP-based data. Unfortunately, web browsers connect to the server's IP address, then ask for a specific web site using an HTTP-specific header known as a "host header name." The key point here is that the web site is selected through HTTP-specific request. Somewhat like the chicken-or-the-egg dilemma, SSL forces the client to build a secure channel prior to transmitting any HTTP data. Yet, name-based virtual hosts force the client to use HTTP headers to select the specific web site to communicate with. Currently, the only viable solution to this quandary is to allocate a dedicated IP to each SSL-enabled web site. Such requirements may lead to added administrative overhead, and possibly exhaustion of IP address space allocation.

As if managing SSL-enabled web sites and securing private keys wasn't hard enough, one of the most challenging tasks for any network administrator has got to be deploying a proxy server in a certificate-enabled environment. The very essence of public key cryptography is to provide secure communications over untrusted networks. Preventing "man-in-the-middle" and other various eavesdropping attacks was one of the primary design goals behind public key cryptography. Conversely, the goal of a proxy server is to sit in between two communicating parties and inspect (and possibly alter) the traffic for various reasons. So how does one build a network which implements certificates to prevent eavesdropping, yet mandates the use of an application proxy? The simple answer is, you can't. If a client is using SSL to communicate with a server, it is impossible for a third-party to decrypt the traffic. As only the server has a copy of the highly secretive private key, the proxy would never be able to decrypt the communications. Attempting to alter the encrypted traffic would result in invalid checksums on the server end, and communications would come to a grinding halt.

While deploying a proxy between an external SSL-enabled parties is impossible, several viable alternative designs do exist. For reverse proxies, one may place a copy of the private key on the proxy itself, thus enabling the proxy to decrypt and inspect inbound traffic. As for traditional proxies, you may opt to deploy a type of "double-tiered" communications model. Rather than having only one communications channel between client and server, you have one communications channel between client and proxy, then another channel between proxy and server. Although such a system may introduce performance ramifications (not to mention immense cost and complexity), it does provide end-to-end security with the ability to inspect application data.

Finally, it's important to note that certificates do not last forever. Your CA will assign a period of validity to your certificate which typically ranges between one and five years. Once your certificate has reached its predefined end-of-life date, it will be considered "expired" and no longer valid. Unless renewed, customers connecting to your server will receive certificate errors.