Digital Certificates

The Minimum You Need To Know

5/28/2007

Jon Green

Digital Certificates- The Minimum You Need To Know

Aruba's position in the IT infrastructure as an authentication point, VPN termination point, and as a part of an enterprise network's security framework increasingly requires that we deal with digital certificates. But what are these things and how do they work? What is X.509? What is a "chain of trust"? And what on Earth is a "self-signed certificate"? This article intends to cover the minimum you need to know to competently deal with digital certificates. Some concepts will be overly simplified; those looking for more in-depth coverage without necessarily resorting to a graduate-level course in cryptography are encouraged to pick up a copy of Cryptography Decrypted by H.X. Mel.

Secret Key versus Public Key Cryptography

At a basic level, cryptography is about hiding secrets. Most modern encryption schemes use a secret key to encrypt data, thus obscuring the contents of the data we wish to protect. The intended receiver of this data decrypts the data using the same secret key. This is a very simple system, and many of the cryptosystems we are familiar with operate this way, such as AES, DES, 3DES, and RC4. However, this type of system brings up the thorny issue of key management: How do we ensure that both the sender and the receiver have the same secret key, and how to we ensure that others don't find out the secret key? This is where public key cryptography comes in.

In public key cryptography, there are two keys instead of just one. The two keys, known as a *public key* and a *private key*, are mathematically related to each other such that data encrypted with a private key can only be decrypted by the public key, and data encrypted with the public key can only be decrypted by the private key. As the names imply, the private key is something you keep private – only you have this key. The public key, on the other hand, is made available to anyone in the world who wants it. The system works as an encryption scheme because the keys are very large numbers, and factoring (finding all the prime numbers that, multiplied together, equal the number) a very large number is a computationally difficult problem.

To use public key cryptography with another person, for example to exchange encrypted emails, you must know the other person's public key. When sending an email, you will encrypt the email with the other person's public key. Upon receipt, the other person will decrypt your message using his private key.

Signing

Public key cryptography is commonly used to encrypt data as well as to sign it. Encryption is about keeping data secret, while signing is about the integrity of the data – making sure it was not tampered with. Very often we may want to leave data in the clear (plaintext or cleartext) but want some assurance that data has not been modified in

transport, by a "man in the middle" for example. Signing can also be used as a "digital signature" for documents in lieu of signing a document in ink. Adobe Acrobat, for example, allows one to imbed a digital signature block in a document. Clicking on this block, you will be prompted to digitally sign the document using public key cryptography. By signing a document with a private key, this proves that you and only you signed the document, since only you have the private key. This brings us to the other major use of signing: non-repudiation. This principle establishes that you and only you signed a document, and that the document was not modified after the time you signed it.

Signing is very simple: Take the document or data that you wish to sign and compute a hash of the document using your favorite hash algorithm such as MD5 or SHA1. Since the odds of two documents that generate the same MD5 hash are extremely (extremely!) low, we now have a solid guarantee of document integrity, right? Wrong: Anyone modifying the document during transport could simply recompute the MD5 hash and replace it. This is where digital signing comes in. Take the MD5 hash and encrypt it using your private key, then attach it to the end of the document. A recipient of the document will decrypt the block at the end using your public key (if you want to make it easy, attach the public key to the document) to get back the MD5 hash. The recipient then computes the MD5 hash of the document, and compares it to the received MD5 hash. If the two match, the document was not altered in transport.

Key Distribution

The astute among you will immediately notice a flaw in the document signing example above:

> *"if you want to make it easy, attach the public key to the document"*

Let's put our "man in the middle" into play here – let's call him "Gregor". Randy wants to send Josh a signed email. Randy types up his email, computes the MD5 hash of his email, encrypts the hash with his private key, and attaches it to the email. Because he has never communicated with Josh before, Josh doesn't have Randy's public key. Because Randy is the helpful sort, he attaches his public key to the email. Gregor intercepts the email, changes the text, and then recomputes the digital signature using a private key he generated himself. He attaches the corresponding public key to the email, replacing the one Randy originally attached. He then forwards the email along to Josh. Upon receipt, Josh installs the attached public key into his electronic "keyring" and uses it to successfully check the integrity of the received email. Since it checks out, he knows the message from Randy is genuine and wasn't modified in transport.

The problem here is a twist on our classic secret key encryption problem*: key distribution.* We have removed the problem of keeping the key secret – the public key is *supposed* to be public. However, we have no way to verify that a public key actually belongs to the person it is supposed to belong to – the public key is simply a large number, with no identifying information attached. An adversary can easily substitute one large number for another, and fool you into using the wrong public key. One solution includes meeting the other person face to face, checking two forms of government-issued

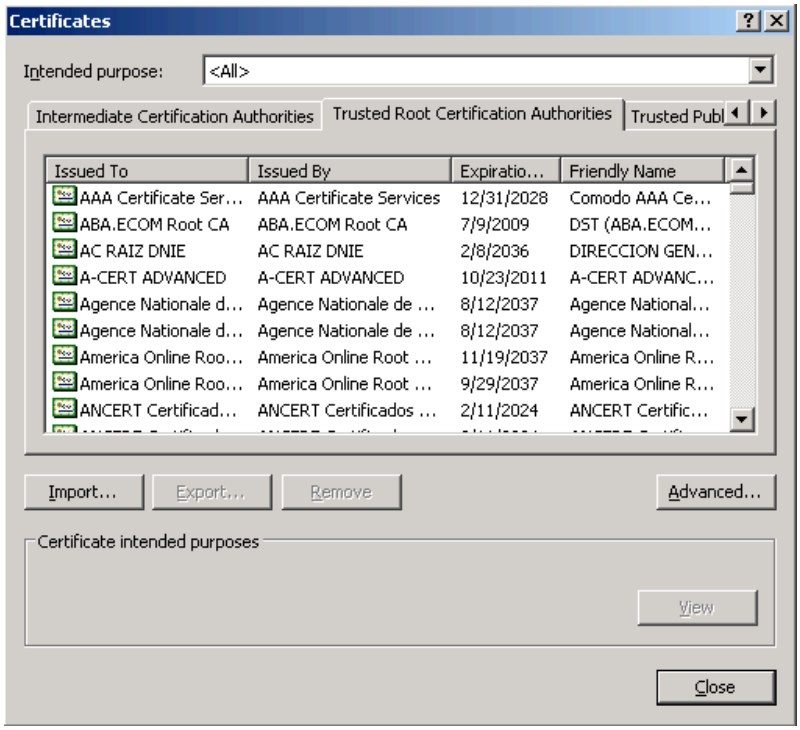
photo ID, and then exchanging public keys on CD-ROMs or floppies. Other solutions involve generating "key fingerprints" and then reading these over the phone to the other person (assuming you have positively established the person is who he claims to be.) These solutions are cumbersome at best, and would not allow public key cryptography to be widely deployed if they were required. Fortunately, digital certificates provide a solution for us.

Digital Certificates

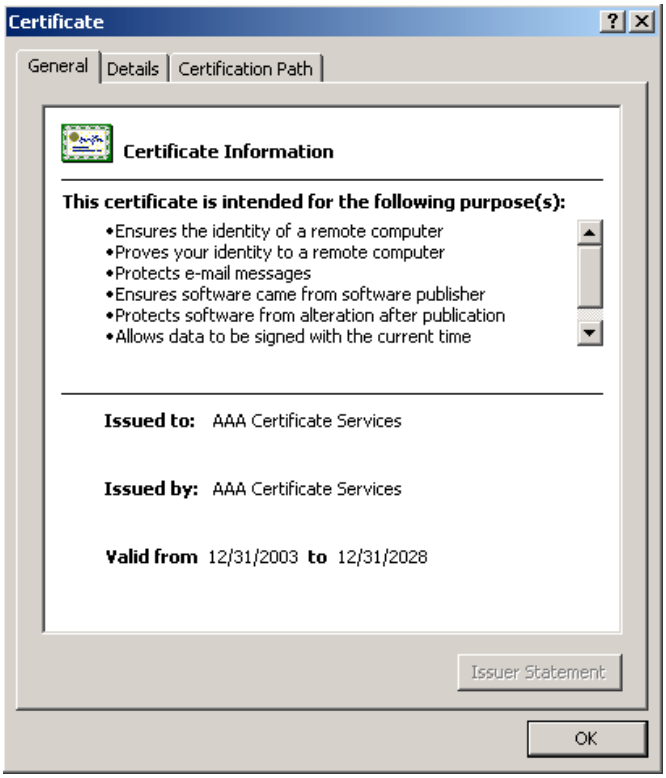
A digital certificate makes someone else do the job of verifying that a public key belongs to the correct person or entity. This "someone else" is known as a *certificate authority (CA)*. A certificate authority is an organization that the users of a digital certificate have chosen to trust to perform the validation of identity and the issuance of digital certificates. Well-known public certificate authorities on the Internet include VeriSign, Thawte, Entrust, and GeoTrust. These companies will, for a fee, verify your identity or the identity of your organization and then issue you a digital certificate, which contains your public key. Other users of the system, having chosen to trust these CAs, will know that your digital certificate is genuine. The entire system is known as a Public Key Infrastructure, or PKI.

The most widely used PKI is the X.509 system, specified by the ITU-T. X.509 specifies a file format that is used for digital certificates. Inside a digital certificate file are a number of fields including the "distinguished name", an alternate name (often an email address), a serial number, a validity date, an expiration date, information about the encryption algorithms used to generate the keys, and other information. The digital certificate file also contains your public key, typically generated using the RSA or DSA algorithm. The certificate authority generates this entire file, then digitally signs the file using a private key.

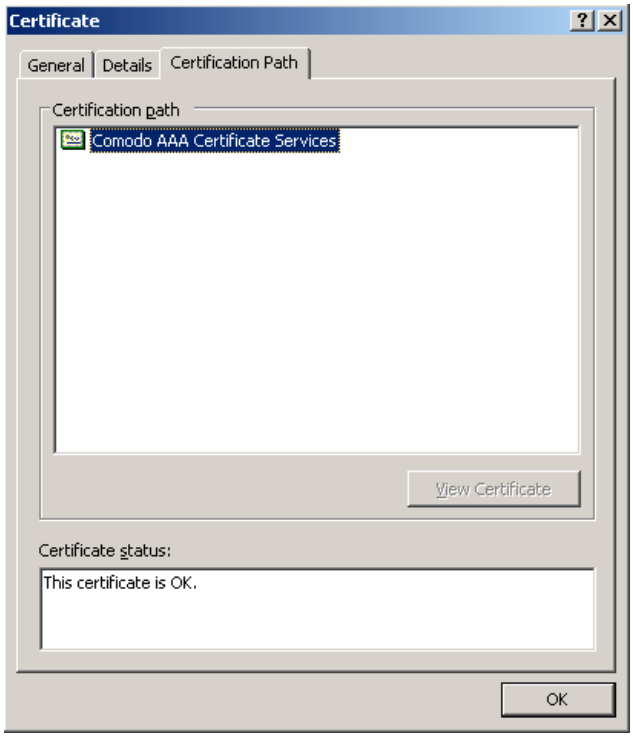
The CA's private key has a corresponding public key. Any user in the PKI must have the CA's public key ahead of time in order to validate a digital certificate generated by that CA. How do we get that public key? Easy: another digital certificate, which contains the CA's public key and is also signed by the CA. This type of certificate is known as a *self-signed* certificate, because it is signed with the same public key that is contained within the certificate. Users in a PKI must choose to trust a CA's self-signed certificate before it can be used to validate other certificates generated by the CA. In some cases, network administrators will take care of establishing trust for a particular CA. In the case of large public Internet CAs such as VeriSign or Thawte, operating system manufacturers and software developers have taken care of this for us. Microsoft Windows users can view the installed CA certificates, for example, by opening Internet Explorer and navigating to Tools->Internet Options->Content->Certificates. Microsoft's term for these CAs is "Trusted Root Certification Authorities".



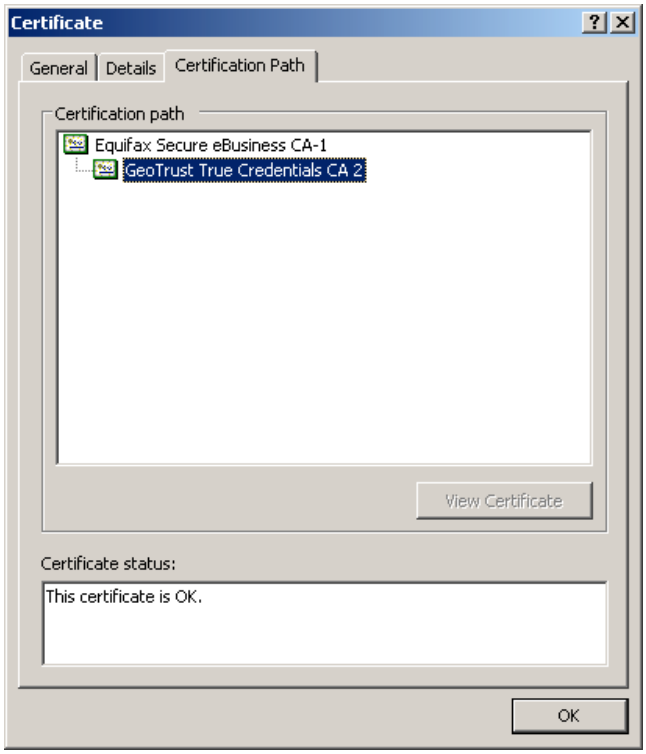
If you double-click one of these CAs, you can view the certificate details. Because these are self-signed root certificate authorities, you will see that the certificate "Issued to" and "Issued by" fields are the same.



If you further navigate to “Certification Path”, you can view the certificate issuance chain. A root CA certificate will contain one and only one entry, because it is self-signed.



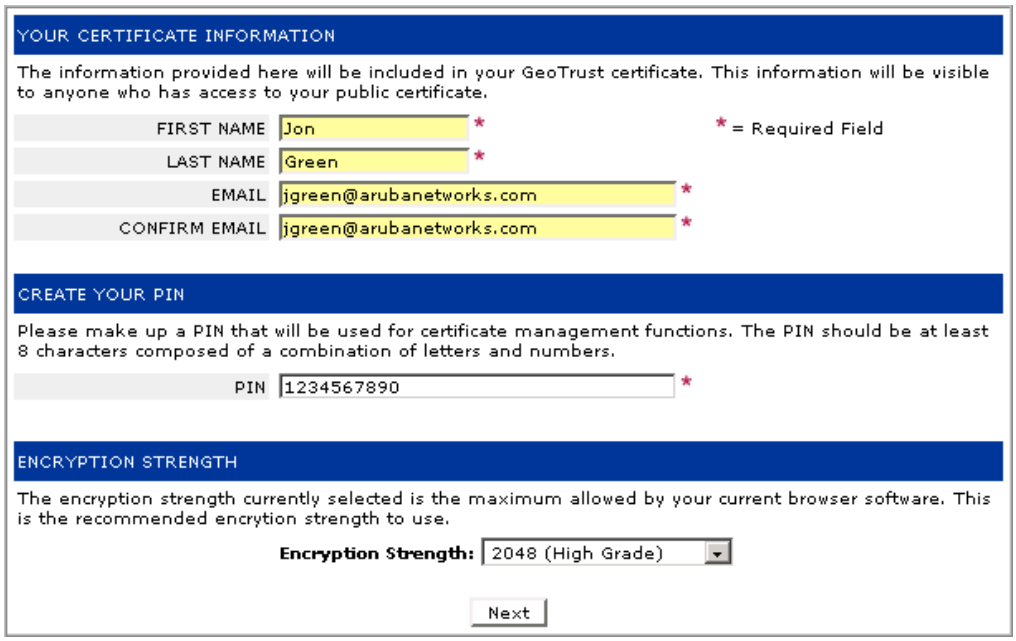
Compare this to an Intermediate Certification Authority. An Intermediate CA is a CA server that can issue trusted digital certificates, but only on the authority of a higher level CA server. If the root CA certificate is revoked or expires, the intermediate CA is also no longer trusted. Below, we see the certification path for an intermediate CA. Note that we must already have a certificate for “Equifax Secure eBusiness CA-1” in our certificate store in order to trust this intermediate CA.



End-User Certificates

Now that we understand the structure of a PKI and how certificate authorities work, let's look at how an end user gets a digital certificate. This could include a company obtaining a digital certificate to be used for an SSL website, or an individual looking for a certificate to sign emails or other documents. I will use for this example my own personal digital certificate, used to digitally sign emails. GeoTrust offers a certificate called "My Credential" that costs $19.95 per year. For this price, they will do some simple verification of identity that involves checking an email address and a phone number. For a higher fee, CAs will verify additional information such as a business license, a physical address of a company, employment status, and other details that increase confidence in identity.

GeoTrust begins the process by asking for some personal information:



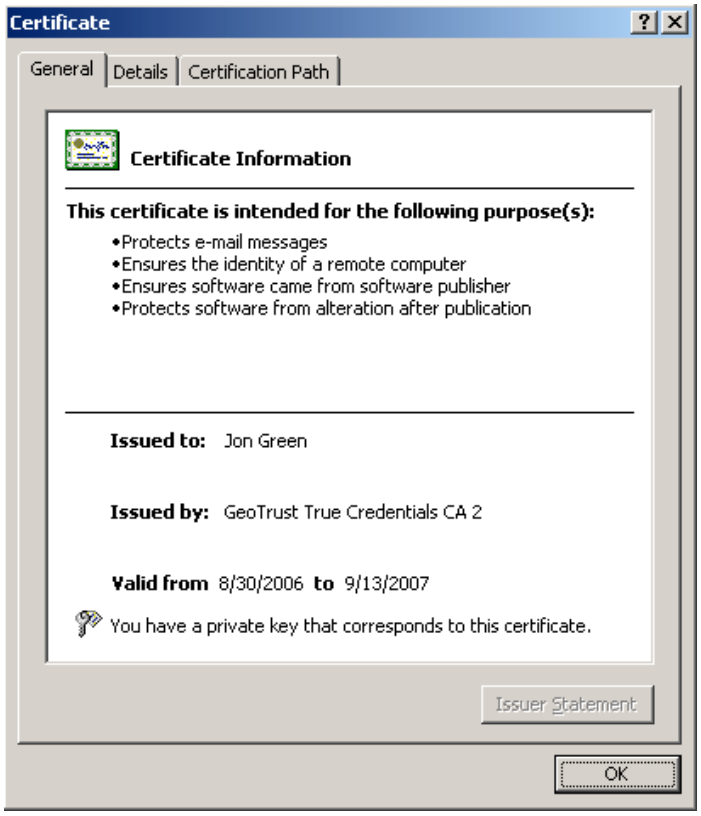
A small window opens up and prompts the user to generate a password. This password is used to generate a public/private keypair, which the GeoTrust server stores. It is important to remember the password – it will be used again later.

After providing payment information, GeoTrust sends an email to the supplied email address and requires the user to validate the email address by clicking on an HTTP link. Next, the GeoTrust system calls the provided phone number and requires that the user validate the phone number as well.

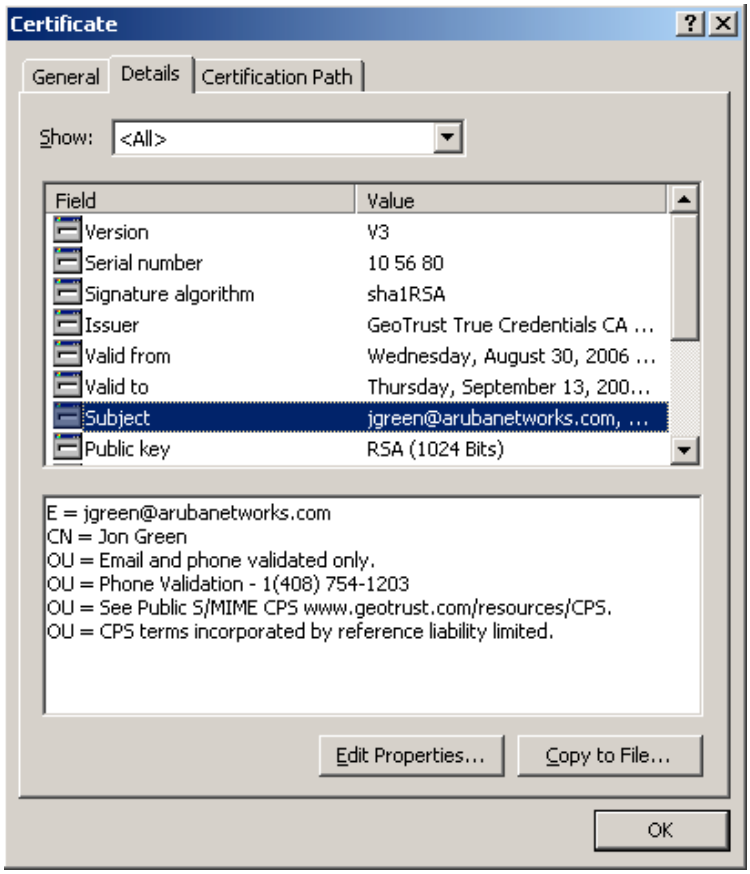
Once identity verification has been completed, the GeoTrust server takes the user's public key along with information such as name, email address, phone number, and validity period, and generates a *Certificate Signing Request (CSR)*. The CSR is fed to the actual CA server. The CA server takes all supplied information, puts it into an X.509 file

format, and signs the file using its private key. Note that the CA server was never given the user's private key – the private key is not part of the digital certificate and thus is not required by the CA server.

The user downloads the certificate and private key together in a file format known as PFX or PKCS#12 (see below for more details on file formats) and installs this certificate into the operating system or email client. Looking at the certificate below, we can see that it was issued by the CA server known as "GeoTrust True Credentials CA 2" and that it is valid for some specific purposes such as protecting email. It is not valid, for example, to be used as a CA certificate itself, and thus cannot be used to issue additional certificates on behalf of GeoTrust. We can also note the presence of "You have a private key that corresponds to this certificate" – this line indicates that this certificate can be used to digitally sign documents or data.



Looking at the certificate details, we find information inserted into the Subject field about how identity was verified. This lets the user of the certificate determine how much trust should be given to it – a certificate where a user physically presented himself and a passport at the CA's office will be given a higher degree of trust than a simple email/phone validation.



Going back to our email example from earlier, Randy (after obtaining his own certificate) can now digitally sign his email to Josh using his private key. But now, instead of attaching his public key to the email, he attaches his digital certificate. Josh receives the digital certificate, and his system looks for a matching CA certificate for the issuing CA. Finding it, his system uses the CA's public key to verify Randy's certificate and determine that it is valid. Next his system extracts Randy's public key from the digital certificate and uses it to check the integrity of the email message. Finding that the integrity check passes, we can be sure that Gregor, our man in the middle, did not modify the message in transit.

Certificate File Formats

Using our GeoTrust example again, a server at GeoTrust packages our new digital certificate and our private key into a single file. This file format is known as PKCS#12 and typically takes the file extension of .p12 or .pfx. In order to protect the confidentiality of this file, it is password-protected using the password we entered at the start of the process. The password protection is important – without it, anyone getting ahold of this file would have a valid digital certificate AND the corresponding private key, and could impersonate us at will.

A file that contains an unprotected certificate and corresponding private key is often known as a PEM (Privacy Enhanced Mail) format file. You can recognize a PEM file because it is base64-encoded plain ASCII text and takes the following format:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDOrkXS8iSveCyNd12JWNVueY+CFPC8YvDCVHb...
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIDjDCCAvWgAwIBAgIBADANBgkqhkiG9w0BAQQFADCBkTELMAk...
-----END CERTIFICATE-----
```

Other common encoding schemes for digital certificates include CER (Canonical Encoding Rules) and DER (Distinguished Encoding Rules). These files are not human-readable text.

Certificate Revocation

Let's say that Randy carelessly stores his digital certificate and private key in a PEM file and stores it on a network drive where others can get access to it. Randy's adversary – let's call him "Taylor" – comes along and finds this file, and begins using it to impersonate Randy. Once the private key to a certificate has been exposed, the digital certificate can no longer be trusted by anyone. This is where certificate revocation comes in. Certificate revocation allows us to go back to the issuing CA and ask that it revoke, or make invalid, a previously issued certificate. Each CA will generate a Certificate Revocation List (CRL) that contains the serial numbers of certificates that have been revoked. An end user or client software should, before trusting a digital certificate, check the CRL from that CA to make sure the certificate is still valid. In practice, this step is rarely done because CRLs can grow very large and there is significant overhead involved in checking a CRL each time.

An alternate to CRLs is known as Online Certificate Status Protocol (OCSP). This is described in RFC 2560 and takes the form of a request/response query to an OCSP server. OCSP queries eliminate the overhead of transferring large CRLs, but can add delay to a transaction (for example, an 802.1x authentication during roaming).

Conclusion

Properly implemented, public key infrastructure provides a solid method of determining identity, encrypting communication, verifying integrity and establishing non-repudiation. If there is any downside to this technology, it is in the number of moving parts and the complexity of understanding how it works. As the example with GeoTrust shows, however, automation can greatly simplify PKI mechanics and allow a user of even modest technical ability to participate in the system. This is a trend that should continue into higher functions such as server certificates and even certificate authorities.