Public Key Infrastructure

Terence Spies
Voltage Security

The ability to create, manipulate, and share digital documents has created a host of new applications (email, word processing, e-commerce websites), but also created a new set of problems, namely how to protect the privacy and integrity of digital documents when stored and transmitted. The invention of public key cryptography in the 1970s [1] pointed the way to a solution to those problems, most importantly the ability to encrypt data without a shared key, and the ability to “sign” data, insuring its origin and integrity. While these operations are quite conceptually simple, they both rely on the ability to bind a public key (which is typically a large mathematical object) reliably with an identity sensible to the application using the operation (for example, a globally unique name, a legal identifier, or an email address.) Public Key Infrastructure (PKI) is the umbrella term used to refer to the protocols and machinery used to perform this binding.

The most important security protocols used on the Internet rely on PKI to bind names to keys — a crucial function that allows authentication of users and websites. A set of attacks in 2011 called into question the security of the PKI architecture [2,3], especially when governmental entities might be tempted to subvert Internet security assumptions. A number of interesting proposed evolutions of the PKI architecture have been proposed as potential countermeasures to these attacks. Even in the face of these attacks, PKI remains the most important and reliable method of authenticating networked entities.

1. CRYPTOGRAPHIC BACKGROUND

To understand how PKI systems function, it is necessary to grasp the basics of public key cryptography. PKI systems enable the use of public key cryptography, and they also use public key cryptography as the basis for their operation. While there are thousands of varieties of cryptographic algorithms, we can understand PKI operations by looking at only two: signature and encryption.

Digital Signatures

The most important cryptographic operation in PKI systems is the digital signature. If two parties are exchanging some digital document, it may be important to protect that data so that the recipient knows that the document has not been altered since it was sent, and that any document received was indeed created by the sender. Digital signatures provide these guarantees by creating a data item, typically attached to the document in question that is uniquely tied to the data and the sender. The recipient then has some verification operation that confirms that the signature data matches the sender and the document.

Figure e39.1 illustrates the basic security problem that motivates signatures. An attacker controlling communications between the sender and receiver can insert a bogus document, fooling the receiver.

The aim of the digital signature is to block this attack by attaching a signature that can only be created by the sender, as shown in Figure e39.2.

Cryptographic algorithms can be used to construct secure digital signatures. These techniques (for example, the RSA or DSA algorithms) all have the same three basic operations, as shown in Table e39.1.

Public Key Encryption

Variants of the three operations used to construct digital signatures can also be used to encrypt data. Encryption uses a public key to scramble data in such a way that only the holder of the corresponding private key can unscramble it (see Figure e39.3).

Public key encryption is accomplished with variants of the same three operations used to sign data, as shown in Table e39.2.

The security of signature and encryption operations depends on two factors: first, the ability to keep the private key private and, second, the ability to reliably tie a public key to a sender. If a private key is known to an attacker, they can then perform the signing operation on
arbitrary bogus documents, and can also decrypt any document encrypted with the matching public key. The same attacks can be performed if an attacker can convince a sender or receiver to use a bogus public key.

PKI systems are built to securely distribute public keys, thereby preventing attackers from inserting bogus public keys. They do not directly address the security of private keys, which are typically defended by measures at a particular endpoint, such as keeping the private key on a smartcard, encrypting private key data using operating system facilities, or other similar mechanisms. The remainder of this section will detail the design, implementation, and operation of public key distribution systems.

FIGURE e39.1 Block diagram of altering an unsigned document.

FIGURE e39.2 Block diagram showing prevention of an alteration attack via digital signature.

<table>
<thead>
<tr>
<th>TABLE e39.1 The Three Fundamental Digital Signature Operations.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key Generation</strong></td>
</tr>
<tr>
<td>Using some random source, the sender creates a public and private key, called K_{public} and K_{private}. Using K_{public}, it is cryptographically difficult to derive K_{private}. The sender then distributes K_{public}, and keeps K_{private} hidden.</td>
</tr>
<tr>
<td><strong>Signing</strong></td>
</tr>
<tr>
<td>Using a document and K_{private}, the sender generates the signature data.</td>
</tr>
<tr>
<td><strong>Verification</strong></td>
</tr>
<tr>
<td>Using the document, the signature, and K_{public}, the receiver (or any other entity with these elements) can test that the signature matches the document, and could only be produced with the K_{private} matching K_{public}.</td>
</tr>
</tbody>
</table>
PKI systems solve the problem of associating meaningful names with essentially meaningless cryptographic keys. For example, when encrypting an email, the user will typically specify a set of recipients that should be able to decrypt that mail. The user will want to specify these as some kind of name (email address or a name from a directory), not as a set of public keys. In the same way, when signed data is received and verified, the user will want to know what user signed the data, not what public key correctly verified the signature. The design goal of PKI systems is to securely and efficiently connect user identities to the public keys used to encrypt and verify data.

The original Diffie-Hellman paper [1] that outlined public key cryptography proposed that this binding would be done through storing public keys in a trusted directory. Whenever a user wanted to encrypt data to another user, they would consult the “public file” and request the public key corresponding to some user. The same operation would yield the public key needed to verify the signature on signed data. The disadvantage of this approach is that the directory must be online and available for every new encryption and verification operation. While this older approach was never widely implemented, variants of this approach are now reappearing in newer PKI designs. For more information, see the section on Alternative PKI Architectures.

PKI systems solve this online problem and accomplish identity binding by distributing “digital certificates”, chunks of data that contain an identity and a key, all authenticated by digital signature, and providing a mechanism to validate these certificates. Certificates, first invented by Kohnfelder in 1978, are essentially a digitally signed message from some authority stating “Entity X is associated with public key Y.” Communicating parties can then rely on this statement (to the extent they trust the authority signing the certificate) to use the public key Y to validate a signature from X or to send an encrypted message to X. Since time may pass between when the signed certificate is produced and when someone uses that certificate, it may be useful to have a validation mechanism to check that the authority still stands by certificate. We will describe PKI systems in terms of producing and validating certificates.

There are multiple standards that describe how certificates are formatted. The X.509 standard, promulgated by the ITU [4], is the most widely used, and is the certificate format used in the TLS/SSL protocols for secure Internet connections, and the S/MIME standards for secured email. The X.509 certificate format also implies a particular model of how certification works. Other standards have attempted to define alternate models of operation and associated certificate models. Among the other standards that describe certificates are: Pretty Good Privacy (PGP) [pgp], and the Simple Public Key Infrastructure...
3. THE X.509 MODEL

The X.509 model is the most prevalent standard for certificate-based PKIs, though the standard has evolved such that PKI-using applications on the Internet are mostly based on the set of IETF standards that have evolved and extended the ideas in X.509. X.509 style certificates are the basis for SSL, TLS, many VPNs, the US Federal Government PKI, and many other widely deployed systems.

The History of X.509

A quick historical preface here is useful to explain some of the properties of X.509. X.509 is part of the X.500 directory standard owned by the International Telecommunications Union Telecommunications Standardization Sector (ITU-T). X.500 specifies a hierarchical directory useful for the X.400 set of messaging standards. As such, it includes naming system (called “distinguished naming”) that describes entities by their position in some hierarchy. A sample X.500/X.400 name might look like this:

CN = Joe Davis, OU = Human Resources, O = WidgetCo, C = US

This name describes a person with a Common Name (CN) of “Joe Davis” that works in an Organizational Unit (OU) called “Human Resources”, in an Organization called “WidgetCo”, in the United States. These name components were intended to be run by their own directory components (so, for example, there would be “Country” directories that would point to “Organizational” directories, etc.) and this hierarchical description was ultimately reflected in the design of the X.509 system. Many of the changes made by IETF and other bodies that have evolved the X509 standard were made to reconcile this hierarchical naming system with the more distributed nature of the Internet.

The X.509 Certificate Model

The X.509 model specifies a system of Certifying Authorities (CAs) that issue certificates for end entities (users, web sites, or other entities that hold private keys.) A CA issued certificate will contain (among other data) the name of the end entity, the name of the CA, the end entity’s public key, a validity period, and a certificate serial number. All of this information is signed with the CA’s private key. (Additional details on the information in a certificate and how it is encoded is in Section Z.) To validate a certificate, a relying party uses the CA’s public key to verify the signature on the certificate, checks that the time falls within the validity period, and may also perform some other online checks.

This process leaves out on important detail: where did the CA’s public key come from? The answer is that another certificate is typically used to certify the public key of the CA. This “chaining” action of validating a certificate by using the public key from another certificate can be performed any number of times, allowing for arbitrarily deep hierarchies of CAs. Of course, this must terminate at some point, typically at a self-signed certificate that is trusted by the relying party. Trusted self-signed certificates are typically referred to as “root” certificates. Once the relying party has verified the chain of signatures from the end-entity certificate to a trusted root certificate, it can conclude that the end-entity certificate is properly signed, and then move onto whatever other validation steps (proper key usage fields, validity dates in some time window, etc.) are required to fully trust the certificate. Figure e39.4 shows the structure of a typical certificate chain.

One other element is required for this system to function securely: CAs much be able “undo” a certification action. While a certificate binds an identity to a key, there are many events that may cause that binding to become invalid. For example, a CA operated by a bank may issue a certificate to a newly hired person that gives that user the ability to sign messages as an employee of the bank. If that person leaves the bank before the certificate expires, the bank needs some way of undoing that certification. The physical compromise of a private key is another circumstance that may require invalidating a certificate. This is accomplished by a validation protocol, where (in the abstract) a user examining a certificate can ask the CA if a certificate is still valid. In practice, revocation protocols are used that simulate this action without actually contacting the CA.

Root certificates are critical to the process of validating public keys through certificates. They must be inherently trusted by the application, since no other certificate signs these certificates. This is most commonly done by installing the certificates as part of the application that will use the certificates under a set of root certificates. For example, Internet Explorer uses X.509 certificates to validate keys used to make Secure Socket Layer (SSL) connections. Internet Explorer has a large set of root certificates installed which can be examined by opening the Internet Options menu item and selecting “Certificates” in the “Content” tab of the Options dialog. A list like the one in Figure e39.5 will appear:

This dialog can also be used to inspect these root certificates. The Microsoft Root certificate details look like the ones shown in Figure e39.6.

The meaning of these fields will be explored in subsequent sections.
4. X.509 IMPLEMENTATION
ARCHITECTURES

While, in theory, the Certification Authority is the entity that creates and validates certificates, in practice, it may be desirable or necessary to delegate the actions of user authentication and certificate validation to other servers. The security of the CA’s signing key is crucial to the security of a PKI system. By limiting the functions of the server that holds that key, it should be subject to less risk of disclosure or illegitimate use. The X.509 architecture defines a delegated server role, the Registration Authority (RA), which allows delegation of authentication. Subsequent extensions to the core X.509 architecture...
have created a second delegated role, the Validation Authority (VA), which owns answering queries about the validity of a certificate after creation.

A Registration Authority is typically used to distribute the authentication function needed to issue a certificate without needing to distribute the CA key. The RA’s function is to perform the authentication needed to issue a certificate, then send a signed statement containing the fact that it performed the authentication, the identity to be certified, and the key to be certified. The CA validates the RA’s message, and issues a certificate in response.

For example, a large multi-national corporation wants to deploy a PKI system using a centralized CA. It wants to issue certificates on the basis of in-person authentication, so needs some way to distribute authentication to multiple locations in different countries. Copying and distributing the CA signing key creates a number of risks, not only due to the fact that the CA key will be present on multiple servers, but also due to the complexities of creating and managing these copies. Sub-CAs could be created for each location, but this requires careful attention to controlling the identities allowed to be certified by each sub-CA (otherwise, an attacker compromising one sub CA could issue a certificate for any identity they liked.) One possible way to solve this problem is to create RAs at each location, and have the CA check that the RA is authorized to authenticate a particular employee when a certificate is requested. If an attacker subverts a given RA signing key, they can request certificates for employees in the purview of that RA, but it is straightforward, once discovered, to deauthorize the RA, solve the security problem, and create a new RA key.

Validation Authorities are given the ability to revoke certificates (the specific methods used to effect revocation are detailed in the X.509 Revocation Protocols section), and offload that function from the CA. Through judicious use of RAs and VAs, it is possible to construct certification architectures where the critical CA server is only accessible to a very small number of other servers, and network security controls can be used to reduce or eliminate threats from outside network entities.

5. X.509 CERTIFICATE VALIDATION

X.509 certificate validation is a complex process, and can be done to several levels of confidence. This section will outline a typical set of steps involved in validating a certificate, but is not an exhaustive catalog of the possible methods that can be used. Different applications will often require different validation techniques, depending on the application’s security policy. It is rare for an application to implement certificate validation, as there are several APIs and libraries available to perform this task. Microsoft CryptoAPI, OpenSSL, and the Java JCE all provide certificate validation interfaces. The Server-based Certificate Validity Protocol (SCVP) can also be used to validate a certificate. However, all of these interfaces offer a variety of options, and understanding the validation process is essential to properly using these interfaces.

While a complete specification of the certificate validation process would require hundreds of pages, we supply a sketch of what happens during certificate validation. It is not a complete description, and is purposefully simplified. The certificate validation process typically proceeds in three steps, and typically takes three inputs. The first is the certificate to be validated, the second is any intermediate certificates acquired by the applications, and the third is a store containing the root and intermediate certificates trusted by the application. The following steps are a simplified outline of how certificates are typically validated. In practice, the introduction of bridge CAs and other non-hierarchical certification models have led to more complex validation procedures. IETF RFC 3280 [5] presents a complete specification for certificate validation, and RFC 4158 [6] presents a specification for constructing a certification path in environments where non-hierarchical certification structures are used.
Validation Step 1: Construct the Chain and Validate Signatures

The contents of the target certificate cannot be trusted until the signature on the certificate is validated, so the first step is to check the signature. To check the signature, the certificate for the authority that signed the target certificate must be located. This is done by searching the intermediate certificates and certificate store for a certificate with a Subject field that matches the Issuer field of the target certificate. If multiple certificates match, the validator can search the matching certificates for a Subject Key Identifier extension that matches the Issuer Key Identifier extension in the candidate certificates. If multiple certificates still match, the most recently issued candidate certificate can be used. (Note that, because of potentially revoked intermediate certificates, multiple chains may need to be constructed and examine through steps 2 and 3 to find the actual valid chain.) Once the proper authority certificate is found, the validator checks the signature on the target certificate using the public key in the authority certificate. If the signature check fails, the validation process can be stopped, and the target certificate deemed invalid.

If the signature matches, and the authority certificate is a trusted certificate, the constructed chain is then subjected to steps 2–4. If not, the authority certificate is treated as a target certificate, and Step 1 is called recursively until it returns a chain to a trusted certificate or fails.

Constructing the complete certificate path requires that the validator is in possession of all the certificates in that path. This requires that the validator keep a database of intermediate certificates or that the protocol using the certificate supplies the needed intermediates. The Server Certificate Validation Protocol (SCVP) provides a mechanism to request a certificate chain from a server, which can eliminate these requirements. The SCVP protocol is described in more detail in a subsequent section.

Step 2: Check Validity Dates, Policy and Key Usage

Once a chain has been constructed, various fields in the certificate are checked to insure that the certificate was issued correctly and that it is currently valid. The following checks should be run on the candidate chain:

- The certificate chain times are correct. Each certificate in the chain contains a validity period with a not before and not after time. For applications outside validating the signature on a document, the current time must fall after the not before time and before the not after time. Some applications may require “time nesting”, meaning that the validity period for a certificate must fall entirely within the validity period of the issuer’s certificate. It is up to the policy of the application if it treats out-of-date certificates as invalid, or treats it as a warning case that can be overridden by the user. Applications may also treat certificates that are not yet valid differently than certificates that have expired.
- Applications that are validating the certificate on a stored document may have to treat validity time as the time that the document was signed as opposed to the time that the signature was checked. There are three cases of interest. The first, and easiest, is where the document signature is checked, and the certificate chain validating the public key contains certificates that are currently within their validity time interval. In this case, the validity times are all good, and verification can proceed. The second case is where the certificate chain validating the public key is currently invalid because one or more certificates are out-of-date, and the document is believed to be signed at a time when the chain was out-of-date. In this case, the validity times are all invalid, and the user should be at least warned.
- The ambiguous case arises when the certificate chain is currently out-of-date, but the chain is believed to have been valid with respect to time when the document was signed. Depending on its policy, the application can treat this case in several different ways. It can assume that the certificate Validity times are strict, and fail to validate the document. Alternatively, it can assume that the certificates were good at the time of signing, and validate the document. The application can also take steps to insure that this case does not occur by using a time-stamping mechanism in conjunction with signing the document, or provide some mechanism for resigning documents before certificate chains expire.

Once the certificate chain has been constructed, the verifier must also verify that various X.509 extension fields are valid. Some common extensions that are relevant to the validity of a certificate path are:

- **BasicConstraints**: This extension is required for CAs, and limits the depth of the certificate chain below a specific CA certificate.
- **NameConstraints**: This extension limits the namespace of identities certified underneath the given CA certificate. This extension can be used to limit a specific CA to issuing certificates for a given domain or X.400 namespace.
- **KeyUsage** and **ExtendedKeyUsage**: These extensions limit the purposes a certified key can be used for. A CA certificates must have KeyUsage set to allow certificate signing. Various values of ExtendedKeyUsage may be required for some certification tasks.

Step 3: Consult Revocation Authorities

Once the verifier has concluded that it has a suitably signed certificate chain with valid dates and proper
keyUsage extensions, it may want to consult the revocation authorities named in each certificate to check that the certificates are currently valid. Certificates may contain extensions that point to Certificate Revocation List (CRL) storage locations or to Online Certificate Status Protocol (OSCP) responders. These methods allow the verifier to check that a CA has not revoked the certificate in question. The next section details these methods in more detail. Note that each certificate in the chain may need to be checked for revocation status. The next section on certificate revocation details the mechanisms used to revoke certificates.

6. X.509 CERTIFICATE REВOCATION

Since certificates are typically valid for a significant period of time, it is possible that during the validity period of the certificate, a key may be lost or stolen, an identity may change, or some other event may occur that causes a certificate’s identity binding to become invalid or suspect. To deal with these events, it must be possible for a CA to revoke a certificate, typically by some kind of notification that can be consulted by applications examining the validity of a certificate. Two mechanisms are used to perform this task: Certificate Revocation Lists (CRLs) and the Online Certificate Status Protocol (OCSP).

The original X.509 architecture implemented revocation via a Certificate Revocation List (CRL). A CRL is a periodically issued document containing a list of certificate serial numbers that are revoked by that CA. X.509 has defined two basic CRL formats, V1 and V2. When CA certificates are revoked by a higher-level CA, the serial number of the CA certificate is placed on an Authority Revocation List (ARL), which is formatted identically to a CRL. CRLs and ARLs, as defined in X.509 and IETF RFC 3280 are ASN.1 encoded objects that contain the following information shown in Table e39.3.

This header is followed by a sequence of revoked certificate records. Each record contains the following information shown in Table e39.4.

The list of revoked certificates is optionally followed by a set of CRL extensions that supply additional information about the CRL and how it should be processed. To process a CRL, the verifying party checks that the CRL has been signed with the key of the named issuer, and that the current date is between the thisUpdate time and the nextUpdate time. This time check is crucial, because if it is not performed, an attacker could use a revoked certificate by supplying an old CRL where the certificate had not yet appeared. Note that expired certificates are typically removed from the CRL, which prevents the CRL from growing unboundedly over time.

Note: CRLs can only revoke certificates on time boundaries determined by the nextUpdate time. If a CA publishes a CRL every Monday, for example, a certificate that is compromised on a Wednesday will continue to validate until its serial number is published in the CRL on the following Monday. Clients validating certificates may have downloaded the CA’s CRL on Monday, and are free to cache the CRL until the nextUpdate time occurs. This caching is important, as it means that the CRL is only downloaded once per client per publication period, rather than for every certificate validation. However, it has the unavoidable consequence of having a potential time-lag between a certificate becoming invalid and its appearance on a CRL. The online certificate validation protocols detailed in the next section attempt to solve this problem.

The costs of maintaining and transmitting CRLs to verifying parties has been repeatedly identified as an important component of the cost of running a PKI system [7,8], and several alternative revocation schemes have been proposed to lower this cost. The cost of CRL distribution was also a factor in the emergence of online certificate status checking protocols like OCSP and SCVP.

Delta CRLs

In large systems that issue many certificates, CRLs can potentially become quite lengthy. One approach to reducing the network overhead associated with sending the

<table>
<thead>
<tr>
<th>TABLE e39.3 Data Fields in an X.509 CRL.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version</strong></td>
</tr>
<tr>
<td><strong>Signature Algorithm</strong></td>
</tr>
<tr>
<td><strong>Issuer</strong></td>
</tr>
<tr>
<td><strong>thisUpdate</strong></td>
</tr>
<tr>
<td><strong>nextUpdate</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE e39.4 Format of a Revocation Record in an X.509 CRL.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Serial Number</strong></td>
</tr>
<tr>
<td><strong>Revocation Date</strong></td>
</tr>
<tr>
<td><strong>CRL Extensions</strong></td>
</tr>
</tbody>
</table>
complete CRL to every verifier is to issue a Delta CRL along with a Base CRL. The base CRL contains the complete set of revoked certificates up to some point in time, and the accompanying Delta CRL contains only the additional certificates added over some time period. Clients that are capable of processing the Delta CRL can then download the Base CRL less frequently, and download the smaller Delta CRL to get recently revoked certificates. Delta CRLs are formatted identically to CRLs, but have a critical extension added in the CRL that denotes that they are a Delta, not Base CRL. IETF RFC 3280 [5] details how Delta CRLs are formatted, and the set of certificate extensions that indicate that a CA issues Delta CRLs.

### Online Certificate Status Protocol

The Online Certificate Status Protocol (OSCP) was designed with the goal of reducing the costs of CRL transmission, and eliminating the time-lag between certificate invalidity and certificate revocation inherent in CRL-based designs. The idea behind OSCP is straightforward. A CA certificate contains a reference to an OSCP server. A client validating a certificate transmits the certificate’s serial number, a hash of the issuer name, and a hash of the subject name, to that OSCP server. The OSCP server checks the certificate status and returns an indication as to the current status of the certificate. This removes the need to download the entire list of revoked certificates, and also allows for essentially instantaneous revocation of invalid certificates. It has the design tradeoff of requiring that clients validating certificates have network connectivity to the required OCSP server.

OSCP responses contain the basic information as to the status of the certificate, in the set of “good”, “revoked”, or “unknown.” They also contain a thisUpdate time, similarly to a CRL and are signed. Responses can also contain a nextUpdate time, which indicates how long the client can consider the OSCP response definitive. The reason the certificate was revoked can also be returned in the response. OSCP is defined in IETF RFC 2560 [9].

### 7. SERVER-BASED CERTIFICATE VALIDITY PROTOCOL

The X.509 certificate path construction and validation process requires a non-trivial amount of code, the ability to fetch and cache CRLs, and, in the case of mesh and bridge CAs, the ability to interpret CA policies. The Server-based Certificate Validity Protocol [10] was designed to reduce the cost of using X.509 certificates by allowing applications to delegate the task of certificate validation to an external server. SCVP offers two levels of functionality: Delegated Path Discovery (DPD), which attempts to locate and construct a complete certificate chain for a given certificate, and Delegated Path Validation (DPV), which performs a complete path validation, including revocation checking, on a certificate chain. The main reason for this division of functionality is that a client can use an untrusted SCVP server for DPD operations, since it will validate the resulting path itself. Only trusted SCVP servers can be used for DPV, since the client must trust the server’s assessment of a certificate’s validity.

SCVP also allows checking certificates according to some defined certification policy. The can be used to centralize policy management for an organization that wishes all clients to follow some set of rules with respect to what set of CAs are trusted, what certification policies are trusted, etc. To use SCVP, the client sends a query to an SCVP server, which contains the following parameters:

- **QueriedCerts.** This is the set of certificates that the client wants the server to construct (and optionally validate) paths for.
- **Checks.** The Checks parameter specifies what the client wants the server to do. The checks parameter can be used to specify that the server should build a path, should build a path and validate it without checking revocation, or should build and fully validate the path.
- **WantBack.** The WantBack parameter specifies what the server should return from the request. This can range from the public key from the validated certificate path (in which case the client is fully delegating certificate validation to the server), to all certificate chains that the server can locate.
- **ValidationPolicy.** The ValidationPolicy parameter instructs the server how to validate the resultant certificate chain. This parameter can be as simple as “use the default RFC 3280 validation algorithm” or can specify a wide range of conditions that must be satisfied. Some of the conditions that can be specified with this parameter are:
  - **KeyUsage and Extended Key Usage.** The client can specify a set of KeyUsage or ExtendedKeyUsage fields that must be present in the end-entity certificate. This allows the client to only accept, for example, certificates that are allowed to perform digital signatures.
  - **UserPolicySet.** The client can specify a set of certification policy OIDs that must be present in the CAs used to construct the chain. CAs can assert that they follow some formally defined policy when issuing certificates, and this parameter allows the client to only accept certificates issued under some set of these policies. For example, if a client wanted to only accept certificates acceptable under the Medium Assurance Federal Bridge CA
policies, it could assert that policy identifier in this parameter. For more information on policy identifiers, see the section on X.509 Extensions.

- **InhibitPolicyMapping.** When issuing bridge or cross-certificates, a CA can assert that a certificate policy identifier in one domain is equivalent to some other policy identifier within it’s domain. By using this parameter, the client can state that it does not want to allow these policy equivalences to be used when validating certificates against values in the UserPolicySet parameter.

- **TrustAnchors.** The client can use this parameter to specify some set of certificates that must be at the top of any acceptable certificate chain. By using this parameter a client could, for example, say that only VeriSign Class 3 certificates were acceptable in this context.

- **ResponseFlags.** This specifies various options as to how the server should respond (if it needs to sign or otherwise protect the response) and if a cached response is acceptable to the client.

- **ValidationTime.** The client may want a validation performed as if it was a specific time, so that it can find if a certificate was valid at some point in the past. Note that SCVP does not allow for “speculative” validation, in terms of asking if a certificate will be valid in the future. This parameter allows the client to specify the validation time to be used by the server.

- **IntermediateCerts.** The client can use this parameter to give additional certificates that can potentially be used to construct the certificate chain. The server is not obligated to use these certificates. This parameter is used where the client may have received a set of intermediate certificates from a communicating party, and is not certain that the SCVP server has possession of these certificates.

- **RevInfos.** Like the IntermediateCerts parameter, the RevInfos parameter supplies extra information that may be needed to construct or validate the path. Instead of certificates, the RevInfos parameter supplies revocation information like OSCP responses, CRLs, or Delta CRLs.

8. **X.509 BRIDGE CERTIFICATION SYSTEMS**

In practice, large scale PKI systems proved to be more complex than could be easily handled under the X.509 hierarchical model. For example, Polk and Hastings [11] identified a number of policy complexities that presented difficulties when attempting to build a PKI system for the United States Federal Government. In this case, certainly one of the largest PKI projects ever undertaken, they found that the traditional model of a hierarchical certification system was simply unworkable. They state:

“The initial designs for a federal PKI were hierarchical in nature because of government’s inherent hierarchical organizational structure. However, these initial PKI plans ran into several obstacles. There was no clear organization within the government that could be identified and agreed upon to run a governmental “root” CA. While the search for an appropriate organization dragged on, federal agencies began to deploy autonomous PKIs to enable their electronic processes. The search for a “root” CA for a hierarchical federal PKI was abandoned, due to the difficulties of imposing a hierarchy after the fact.”

Their proposed solution to this problem was to use a “mesh CA” system to establish a Federal Bridge Certification Authority. This Bridge architecture has since been adopted in large PKI systems in Europe and the financial services community in the United States. The details of the European Bridge CA can be found at http://www.bridge-ca.org. This part of the chapter will detail the technical design of bridge CAs, and the various X.509 certificate features that enable bridges.

**Mesh PKIs and Bridge CAs**

Bridge CA architectures are implemented using a non-hierarchical certification structure called a Mesh PKI. The classic X.509 architecture joins together multiple PKI systems by subordinating them under a higher-level CA. All certificates chain up to this CA, and that CA essentially creates trust between the CAs below it. Mesh PKIs join together multiple PKI systems using a process called “cross certification” that does not create this type of hierarchy. To cross certify, the top level CA in a given hierarchy creates a certificate for an external CA called the Bridge CA. This bridge CA then becomes, in a manner of speaking, a sub-CA under the organization’s CA. However, the Bridge CA also creates a certificate for the organizational CA, so it can also be viewed as a top level CA certifying that organizational CA.

The end result of this cross-certification process is that if two organizations, A and B have joined the same bridge CA, the can both create certificate chains from their respective trusted CAs through the other organization’s CA to end-entity certificates that it has created. These chains will be longer than traditional hierarchical chains, but have the same basic verifiable properties. Figure e39.7 shows how two organizations might be connected through a bridge CA, and what the resultant certificate chains look like.

In the case illustrated in Figure e39.7, a user that trusts certificates issued by PKI A (that is, PKI A Root is a “trust anchor”) can construct a chain to certificates issued
by the PKI B SubCA, since it can verify Certificate 2 via
its trust of the PKI A Root. Certificate 2 then chains to
then is a trusted issuer certificate for certificates issued by
the PKI B SubCA.

Mesh architectures create two significant technical
problems: path construction and policy evaluation. In a
hierarchical PKI system, there is only one path from the
root certificate to an end-entity certificate. Creating a cer-
tificate chain is as simple as taking the current certificate,
locating the issuer in the subject field of another certifi-
cate, and repeating until the root is reached (completing
the chain) or no certificate can be found (failing to con-
struct the chain.) In a Mesh system, there are now be
cyclical loops where this process can fail to terminate
with a failure or success. This is not a difficult problem
to solve, but it is more complex to deal with than the hier-
archical case.

Policy evaluation becomes much more complex in the
Mesh case. In the hierarchical CA case, the top level CA
can establish policies that are followed by Sub CAs, and
these policies can be encoded into certificates in an unambiguously way. When multiple PKIs are joined by a
bridge CA, these PKIs may have similar policies, but
expressed with different names. PKI A and PKI B may
both certify “medium assurance” CAs which perform a
certain level of authentication before issuing certificates,
but may have different identifiers for these policies.
When joined by a bridge CA, clients may reasonably
want to validate certificates issued by both CAs, and
understand the policies that those certificates are issued
under. The PolicyMapping technique allows similar poli-
cies under different names from disjoint PKIs to be trans-
lated at the bridge CA.

While none of these problems are insurmountable,
they increase the complexity of certificate validation
code, and helped drive the invention of server-based vali-
dation protocols like SCVP. These protocols delegate
path discovery and validation to an external server, rather
than require that applications integrate this functionality.
While this may lower application complexity, the main
benefit of this strategy is that questions of
acceptable policies and translation can be configured at
one central verification server rather than distributed to
every application doing certificate validation.

9. X.509 CERTIFICATE FORMAT

The X.509 standard (and the related IETF RFCs) specify
a set of data fields that must be present in a properly for-
matted certificate, a set of optional extension data fields
that can be used to supply additional certificate informa-
tion, how these fields must be signed, and how the signa-
ture data is encoded. All of these data fields (mandatory
fields, optional fields, and the signature) are specified in
Abstract Syntax Notation (aka ASN.1), a formal language
that allows for exact definitions of the content of data
fields and how those fields are arranged in a data struc-
ture. An associated specification, Determined Encoding
Rules (DER) is used with specific certificate data and the
ASN.1 certificate format to create the actual binary certificate data. The ASN.1 standard is authoritatively defined in ITU Recommendation X.693. (For an introduction to ASN.1 and DER, see kaliski)

**X.509 V1 and V2 Format**

The first X.509 certificate standard was published in 1988 as part of the broader X.500 directory standard. X.509 was intended to provide public key based access control to an X.500 directory, and defined a certificate format for that use. This format, now referred to as X.509 v1 defined a static format containing an X.400 Issuer name (the name of the CA), an X.400 Subject name, a validity period, the key to be certified, and the signature of the CA. While this basic format allowed for all the basic PKI operations, the format required that all names be in the X.400 form, and it did not allow for any other information to be added to the certificate. The X.509 v2 format added two more Unique ID fields, but did not fix the primary deficiencies of the v1 format. As it became clear that name formats would have to be more flexible, and certificates would have to accommodate a wider variety of information, work began on a new certificate format.

**X.509 V3 Format**

The X.509 certificate specification was revised in 1996 to add an optional extension field which allows encoding a set of optional additional data fields into the certificate (see Table e39.5). While this change may seem minor, in fact it allowed certificates to carry a wide array of information useful for PKI implementation, and also for the certificate to contain multiple, non-X.400 identities. These extension fields allow for key usage policies, CA policy information, revocation pointers, and other relevant information to live in the certificate. The V3 format is the most widely used X.509 variant and is the basis for the certificate profile in RFC 3280 [5] issued by the Internet Engineering Task Force.

**X.509 Certificate Extensions**

This section is a partial catalog of common X.509 V3 extensions. There is no existing canonical directory of V3 extensions, so there are undoubtedly extensions in use outside this list. The most common extensions are defined in RFC 3280 [5], which contains the IETF certificate profile, used by S/MIME and many SSL/TLS implementations. These extensions address a number of deficiencies in the base X.509 certificate specification, and, in many cases, are essential for constructing a practical PKI system. In particular, the Certificate Policy, Policy Mapping, and Policy Constraints extensions form the basis for the popular bridge CA architectures.

**Authority Key Identifier**

The Authority Key Identifier extension identifies which specific private key owned by the certificate issuer was used to sign the certificate. The use of this extension allows a single issuer to use multiple private keys, and unambiguously identifies which key was used. This allows issuer keys to be refreshed without changing the issuer name, and enables handling events such as an issuer key being compromised or lost.

**Subject Key Identifier**

The Subject Key Identifier extension, like the Authority Key Identifier, indicates which subject key is contained in the certificate. This extension provides a way to quickly identify which certificates belong to a specific key owned by a subject. If the certificate is a CA certificate, the Subject Key Identifier can be used to construct chains by connecting a Subject Key Identifier with a matching Authority Key Identifier.

**Key Usage**

A CA may wish to issue a certificate that limits the use of a public key. This may lead to an increase in overall system security by segregating encryption keys from

---

**TABLE e39.5 Data Fields in an X.509 Version3 Certificate.**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>The version of the standard used to format the certificate</td>
</tr>
<tr>
<td>Serial Number</td>
<td>A number, unique relative to the issuer, for this certificate</td>
</tr>
<tr>
<td>Signature</td>
<td>The specific algorithm used to sign the certificate</td>
</tr>
<tr>
<td>Issuer</td>
<td>Name of the authority issuing the certificate</td>
</tr>
<tr>
<td>Issuer Unique ID Obsolete field</td>
<td></td>
</tr>
<tr>
<td>Subject Unique ID Obsolete field</td>
<td></td>
</tr>
<tr>
<td>Validity</td>
<td>The time interval this certificate is valid for</td>
</tr>
<tr>
<td>Subject</td>
<td>The identity being certified</td>
</tr>
<tr>
<td>Subject Public Key</td>
<td>The key being bound to the Subject</td>
</tr>
<tr>
<td>Extensions</td>
<td>A list of additional certificate attributes</td>
</tr>
<tr>
<td>Signature</td>
<td>A digital signature by the Issuer over the certificate data</td>
</tr>
</tbody>
</table>
signature keys, and even segregating signature keys by utilization. For example, an entity may have a key used for signing documents and a key used for decryption of documents. The signing key may be protected by a smart card mechanism that requires a PIN per signing, while the encryption key is always available when the user is logged in. The use of this extension allows the CA to express that the encryption key cannot be used to generate signatures. This capability is used in electronic transaction scenarios where it is important that signers cannot disavow a transaction.

3. **keyEncipherment** – The key can be used to wrap a symmetric key that is then used to bulk encrypt data. This is used in communications protocols, and applications like S/MIME, where an algorithm like AES is used to encrypt data, and the public key in the certificate is used to then encrypt that AES key. In practice, almost all encryption applications are structured in this manner, as public keys are generally unsuitable for the encryption of bulk data.

4. **dataEncipherment** – The key can be used to directly encrypt data. Because of algorithmic limitations of public encryption algorithms, the keyEncipherment technique is nearly always used instead of directly encrypting data.

5. **keyAgreement** – The key can be used to create a communication key between two parties. This capability can be used in conjunction with the encipherOnly and decipherOnly capabilities.

6. **keyCertSign** – The key can be used to sign another certificate. This is a crucial key usage capability, as it essentially allows creation of sub-certificates under this certificate, subject to basicConstraints.

7. **cRLSign** – The key can be used to sign a Certificate Revocation List (CRL). CA certificates may have this bit set, or they may delegate CRL creation to a different key, in which case this bit will be cleared.

8. **encipherOnly** – When the key is used for keyAgreement, the resultant key can only be used for encryption.

9. **decipherOnly** – When the key is used for keyAgreement, the resultant key can only be used for decryption.

### An Agenda for Action To Define Key Usage Capabilities

The extension defines the following capabilities that need to be completed (Check All Tasks Completed):

1. **digitalSignature** – The key can be used to generate digital signatures.

2. **nonRepudiation** – Signatures generated from this key can be tied back to the signer in such a way that the signer cannot deny generating the signature. This capability is used in electronic transaction scenarios where it is important that signers cannot disavow a transaction.

3. **keyEncipherment** – The key can be used to wrap a symmetric key that is then used to bulk encrypt data. This is used in communications protocols, and applications like S/MIME, where an algorithm like AES is used to encrypt data, and the public key in the certificate is used to then encrypt that AES key. In practice, almost all encryption applications are structured in this manner, as public keys are generally unsuitable for the encryption of bulk data.

4. **dataEncipherment** – The key can be used to directly encrypt data. Because of algorithmic limitations of public encryption algorithms, the keyEncipherment technique is nearly always used instead of directly encrypting data.

5. **keyAgreement** – The key can be used to create a communication key between two parties. This capability can be used in conjunction with the encipherOnly and decipherOnly capabilities.

6. **keyCertSign** – The key can be used to sign another certificate. This is a crucial key usage capability, as it essentially allows creation of sub-certificates under this certificate, subject to basicConstraints.

7. **cRLSign** – The key can be used to sign a Certificate Revocation List (CRL). CA certificates may have this bit set, or they may delegate CRL creation to a different key, in which case this bit will be cleared.

8. **encipherOnly** – When the key is used for keyAgreement, the resultant key can only be used for encryption.

9. **decipherOnly** – When the key is used for keyAgreement, the resultant key can only be used for decryption.

### Subject Alternative Name

This extension allows the certificate to define non-X.400 formatted identities for the subject. It supports a variety of name spaces, including email addresses, DNS names for servers, Electronic Document Interchange (EDI) party names, Uniform Resource Identifiers (URIs), and IP addresses, among others.

### Policy Extensions

Three important X.509 certificate extensions (Certificate Policy, Policy Mapping, and Policy Constraints) form a complete system for communicating CA policies for how certificates are issued, revoked, and CA security is maintained. They are interesting in that they communicate information that is more relevant to business and policy decision making than the other extensions which are used in the technical processes of certificate chain construction and validation. As an example, a variety of CAs run multiple Sub-CAs that issue certificates according to a variety of issuance policies, ranging from “Low Assurance” to “High Assurance.” The CA will typically formally define in a policy document all of its’ operating policies, state them in practice statement, define an ASN.1 Object Identifier (OID) that names this policy, and distribute it to parties that will validate those certificates. The policy extensions allow CAs to attach a policy OID to its certificate, translate policy OIDs between PKIs, and limit the policies that can be used by sub CAs.

### Certificate Policy

The Certificate Policy extension, if present in an issuer certificate, expresses the policies that are followed by the CA, both in terms of how identities are validated before certificate issuance, but also how certificates are revoked, and the operational practices that are used to insure integrity of the CA. These policies can be expressed in two ways: as an OID, which is a unique number that refers to
one given policy, and as a human-readable Certificate Practice Statement (CPS). One Certificate Policy extension can contain both the computer-sensible OID and a printable CPS. One special OID has been set aside for “AnyPolicy”, which states that the CA may issue certificates under a free-form policy.

IETF RFC 2527 [12] gives a complete description of what should be present in a CA policy document and CPS. More details on the 2527 guidelines are given in the PKI Policy Description section.

Policy Mapping
The Policy Mapping extension contains two policy OIDs, one for the Issuer domain, the other for the Subject domain. When this extension is present, a validating party can consider the two policies identical, which is to say, the Subject OID, when present in the chain below the given certificate, can be considered to be the same as the policy named in the Issuer OID. This extension is used to join together two PKI systems with functionally similar policies that have different policy reference OIDs.

Policy Constraints
The Policy Constraints extension enables a CA to disable policy mapping for CAs farther down in the chain, and also to require explicit policies in all the CAs below a given CA.

10. PKI POLICY DESCRIPTION
In many application contexts, it is important to understand how and when certifying authorities will issue and revoke certificates. Especially when bridge architectures are used, an administrator may need to evaluate a certifying authority’s policy to determine how and when to trust certificates issued under that authority. For example, the United States Federal Bridge CA maintains a detailed specification of its operating procedures and requirements for bridged CAs at the US CIO office web site: (http://www.cio.gov/fpkipa/documents/FBCA_CP_RFC3647.pdf). Many other commercial CAs, such as VeriSign, maintain similar documents.

To make policy evaluation easier and more uniform, IETF RFC 2527 [12] specifies a standard format for certifying authorities to communicate their policy for issuing and revoking certificates. This specification divides a policy specification document into the following sections:

• Introduction: This section describes the type of certificates that the CA issues, the applications that those certificates can be used in, and the OIDs used to identify CA policies. The Introduction also contains the contact information for the institution operating the CA.
• General Provisions: This section details the legal obligations of the CA, any warranties given as to the reliability of the bindings in the certificate, and details as to the legal operation of the CA, including fees and relationship to any relevant laws.
• Identification and Authentication: This section details how certificate requests are authenticated at the CA or RA, and how events like name disputes or revocation requests are handled.
• Operational Requirements: This section details how the CA will react in case of key compromise, how it renews keys, how it publishes CRLs or other revocation information, how it is audited, and what records are kept during CA operation.
• Physical, Procedural, and Personnel Security Controls: This section details how the physical location of the CA is controlled, and how employees are vetted.
• Technical Security Controls: This section explains how the CA key is generated and protected throughout its lifecycle. CA key generation is typically done through an audited, recorded key generation ceremony to assure certificate users that the CA key was not copied or otherwise compromised during generation.
• Certificate and CRL Profile: The specific policy OIDs published in certificates generated by the CA are given in this section. The information in this section is sufficient to accomplish the technical evaluation of a certificate chain published by this CA.
• Specification Administration: The last section explains the procedures used to maintain and update the certificate policy statement itself.

These policy statements can be substantial documents. The Federal Bridge CA policy statement is at least 93 pages long, and other certificate authorities have similarly exhaustive documents. The aim of these statements is to provide enough legal backing for certificates produced by these CAs so that they can be used to sign legally binding contracts and automate other legally relevant applications.

11. PKI STANDARDS ORGANIZATIONS
The PKIX Working Group was established in the fall of 1995 with the goal of developing Internet standards to support X.509-based Public Key Infrastructures (PKIs). These specifications form the basis for numerous other IETF specifications that use certificates to secure various protocols, such as S/MIME (for secure email), TLS (for securing TCP connections), and IPSEC (for securing internet packets).
IETF PKIX

The PKIX working group has produced a complete set of specifications for an X.509 based PKI system. These specifications span 36 RFCs, and at least eight more RFCs are being considered by the group at the moment. In addition to the basic core of X.509 certificate profiles and verification strategies, the PKIX drafts cover the format of certificate request messages, certificates for arbitrary attributes (rather than for public keys), and a host of other certificate techniques.

Other IETF groups have produced a group of specifications that detail the usage of certificates in various protocols and applications. In particular, the S/MIME group, which details a method for encrypting email messages, and the SSL/TLS group, which details TCP/IP connection security, use X.509 certificates.

SDSI/SPKI

The Simple Distributed Security Infrastructure (SDSI) group was chartered in 1996 to design a mechanism for distributing public keys that would correct some of the perceived complexities inherent in X.509. In particular, the SDSI group aimed at building a PKI architecture that would not rely on a hierarchical naming system, but would instead work with local names that would not have to be enforced to be globally unique. The eventual SDSI design, produced by Ron Rivest and Butler Lampson, has a number of unique features:

- **Public key-centric design.** The SDSI design uses the public key itself (or a hash of the key) as the primary indentifying name. SDSI signature objects can contain naming statements about the holder of a given key, but the names are not intended to be the “durable” name of an entity.

- **Free-form namespaces.** SDSI imposes no restrictions on what form names must take, and imposes no hierarchy that defines a canonical namespace. Instead, any signer may assert identity information about the holder of a key, but no entity is required to use (or believe) the identity bindings of any other particular signer. This allows each application to create a policy about who can create identities, how those identities are verified, and even what constitutes an identity.

- **Support for groups and roles.** The design of many security constructions (access control lists, for example) often include the ability to refer to groups or roles instead of the identity of individuals. This allows access control and encryption operations to protect data for groups, which may be more natural in some situations.

The Simple Public Key Infrastructure (SPKI) group was started at nearly the same time, with goals similar to the SDSI effort. In X, the two groups were merged, and the SDSI/SPKI 2.0 specification was produced, incorporating ideas from both architectures.

IETF OpenPGP

The Pretty Good Privacy (PGP) public key system, created by Phillip Zimmermann, is a widely deployed PKI system that allows for the signing and encryption of files and email. Unlike the X.509 PKI architecture, the PGP PKI system uses the notion of a “Web of Trust” to bind identities to keys. The Web of Trust (WoT) [13] replaces the X.509 idea of identity binding via an authoritative server with identity binding via multiple semi-trusted paths.

In a WoT system, the end user maintains a database of matching keys and identities, each of which are given two trust ratings. The first trust rating denotes how trusted the binding between the key and the identity is, and the second denotes how trusted a particular identity is to “introduce” new bindings. Users can create and sign a certificate, and import certificates created by other users. Importing a new certificate is treated as an introduction. When a given identity and key in a database are signed by enough trusted identities, that binding is treated as trusted.

Because PGP identities are not bound by an authoritative server, there is also no authoritative server that can revoke a key. Instead, the PGP model states that the holder of a key can revoke that key by posting a signed revocation message to a public server. Any user seeing a properly signed revocation message then removes that key from their database. Because revocation messages must be signed, only the holder of the key can produce them, so it is impossible to produce a false revocation with out compromising the key. If an attacker does compromise the key, then production of a revocation message from that compromised key actually improves the security of the overall system, because it warns other users to not trust that key.

12. PGP CERTIFICATE FORMATS

To support the unique features of the Web of Trust system, PGP invented a very flexible packetized message format that can encode encrypted messages, signed messages, key database entries, key revocation messages, and certificates. This packetized design, described in IETF RFC 2440, allows a PGP certificate to contain a variable number of names and signatures, as opposed to the single-certification model used in X.509.
A PGP certificate (known as a transferrable public key) contains three main sections of packetized data. The first section contains the main public key itself, potentially followed by some set of relevant revocation packets. The next section contains a set of User ID packets, each of which contains an identity and a signature of the User ID packet and the main public key. Each of these User ID packets essentially forms an identity binding. Because each PGP certificate can contain any number of these User ID/Signature elements, a single certificate can assert that a public key is bound to multiple identities (for example, multiple email addresses that correspond to a single user), certified by multiple signers. This multiple signer approach enables the Web of Trust model. The last section of the certificate is optional and may contain multiple subkeys, which are single function keys (for example, an encryption only key) also owned by the holder of the main public key. Each of these subkeys must be signed by the main public key.

PGP Signature packets contain all the information needed to perform a certification, including time intervals for which the signature is valid. Figure e39.8 shows how the multi-name, multi-signature PGP format differs from the single-name with single-signature X.509 format.

13. PGP PKI IMPLEMENTATIONS

The PGP PKI system is implemented in commercial products sold by the PGP corporation, and several open source projects, including Gnu Privacy Guard (GPG) and OpenPGP. Thawte offers a Web of Trust service that connects people with “Web of Trust notaries” that can build trusted introductions. PGP Corporation operates a PGP Global Directory that contains PGP keys along with an email confirmation service to make key certification easier. The OpenPGP group (www.openpgp.org) maintains the IETF specification (RFC 2440) for the PGP message and certificate format.

14. W3C

The World Wide Web Consortium (W3C) standards group has published a series of standards on encrypting and signing XML documents. These standards, XML Signature and XML Encryption have a companion PKI specification called XKMS (XML Key Management Specification.)

The XKMS specification describes a meta-PKI that can be used to register, locate, and validate keys that may be certified by an outside X.509 CA, a PGP referrer, a SPKI key signer, or the XKMS infrastructure itself. The specification contains two protocol specifications, X-KISS (XML Key Information Service Specification) and X-KRSS (XML Key Registration Service Specification). X-KISS is used to find and validate a public key referenced in an XML document, and X-KRSS is used to register a public key so that it can be located by X-KISS requests.

15. IS PKI SECURE?

PKI has formed the basis of Internet security protocols like S/MIME for securing email, and the SSL/TLS protocols for securing communications between clients and web servers. The essential job of PKI in these protocols is binding a name, like an email address or domain name to a key that is controlled by that entity. As seen in this chapter, that job boils down to a CA issuing a certificate for an entity. The security of these systems then rests on the trustworthiness of the CAs trusted within an application. If a CA issues a set of bad certificates, the security of the entire system can be called into question.

The issue of a subverted CA was largely theoretical until attacks on the Comodo and DigiNotar CAs [2,3] in 2011. Both of these CAs discovered that an attacker had bypassed their internal controls and obtained certificates...
for prominent Internet domains (google.com, yahoo.com.) These certificates were revoked, but the incident caused the major browser vendors to revisit their policies about what CA roots are trusted, and the removal of many CAs. In the case of DigiNotar, these attacks ultimately led to the bankruptcy of the company.

By attacking a CA and obtaining a false certificate for a given domain, the attacker can set up a fake version of the domain’s web site, and using that certificate, create secure connections to clients that trust that CA’s root certificate. This secure connection can be used as a “man-in-the-middle” server that reveals all traffic between the client (or clients) and the legitimate web site.

Can these attacks be prevented? There are research protocols such as “Perspectives” [14] that attempt to detect false certificates that might be signed by a legitimate CA. These protocols use third party repositories to track what certificates and keys are used by individual web sites. A change that is only noticed by some subset of users may indicate an attacker using a certificate to gain access to secured traffic.

Some other products that rely on PKI certification have introduced new features to make these attacks harder to execute. Google’s Chrome browser also has recently incorporated security features [15] intended to foil attacks on PKI infrastructure. The Mozilla Foundation, owners of the Firefox browser, have instituted an audit and review system that requires all trusted CAs to attest they have specific kinds of security mechanisms in place to prevent the issuance of illegitimate certificates.

As a general principle, systems built to rely on PKI for security should understand the risks involved in CA compromise, and also understand how critical it is to control exposure to these kinds of attacks. One simple mechanism for doing this is to restrict the number of CAs that are trusted by the application. These large number of roots trusted by the average web browser is quite large, which makes auditing of the complete list of CAs difficult.

16. ALTERNATIVE PKI ARCHITECTURES

PKI systems have proven to be remarkably effective tools for some protocols, most notably SSL, which has emerged as the dominant standard for encrypting internet traffic. Deploying PKI systems for other types of applications or as a general key management system has not been as successful. The differentiating factor seems to be that PKI keys for machine end-entities (like web sites) do not encounter usability hurdles that emerge when issuing PKI keys for human end-entities. Peter Guttman [notdead] has a number of overviews of PKI that present the fundamental difficulties of classic X.509 PKI architectures. Alma Whitten and Doug Tygar [16] published “Why Johnny Can’t Encrypt”, a study of various users attempting to encrypt email messages using certificates. This study showed substantial user failure rates, due to the complexities of understanding certificate naming and validation practices. A subsequent study [17] showed similar results when using X.509 certificates with S/MIME encryption in Microsoft Outlook Express. The majority of the research on PKI alternatives has focused on making encryption easier to use and deploy.

17. MODIFIED X.509 ARCHITECTURES

Some researchers have proposed modifications or redesigns of the X.509 architecture to make obtaining a certificate easier, and lower the cost of operating applications that depend on certificates. The goal of these systems is often to allow internet based services to use certificate based signature and encryption service without requiring the user to consciously interact with certification services or even understand that certificates are being utilized.

Perlman and Kaufman’s User-Centric PKI

Perlman and Kaufman proposed the “User-centric PKI” [Perlman], which allows the user to act as their own CA, with authentication provided through individual registration with service providers. It has several features that attempt to protect user privacy through allowing the user to pick what attributes are visible to a specific service provider.

Guttman’s Plug and Play PKI

Guttman’s proposed “Plug and Play PKI” [gutmann-pnp] provides for similar self-registration with a service provider and adds location protocols to establish how to contact certifying services. The goal is to build a PKI which provides a reasonable level of security and which is essentially transparent to the end user.

Callas’ Self-Assembling PKI

In 2003, Jon Callas [18] proposed a PKI system that would use existing, standard PKI elements bound together by a “robot” server that would examine messages sent between users, and attempt to find certificates that could be used to secure the message. In the absence of an available certificate, the robot would create a key on behalf of the user, and send a message requesting authentication. This system has the benefit for speeding deployment of PKI systems for email authentication, but loses many of the strict authentication attributes that drove the development of the X.509 and IETF PKI standards.
18. ALTERNATIVE KEY MANAGEMENT MODELS

While PKI systems can be used for encryption as well as digital signature, these two applications have different operational characteristics. In particular, systems that use PKIs for encryption require that an encrypting party has the ability to locate certificates for its desired set of recipients. In digital signature applications, a signer only requires access to their own private key and certificate. The certificates required to verify the signature can be sent with the signed document, so there is no requirement for verifiers to locate arbitrary certificates. These difficulties have been identified as factors contributing to the difficulty of practical deployment of PKI based encryption systems like S/MIME.

In 1984, Adi Shamir [19] proposed an Identity-Based Encryption (IBE) system for email encryption. In the identity-based model, any string can be mathematically transformed into a public key, typically using some public information from a server. A message can then be encrypted with this key. To decrypt, the message recipient contacts the server and requests a corresponding private key. The server is able to mathematically derive a private key, which is returned to the recipient. Shamir disclosed how to perform a signature operation in this model, but did not give a solution for encryption.

This approach has significant advantages over the traditional PKI model of encryption. The most obvious is the ability to send an encrypted message without locating a certificate for a given recipient. There are other points of differentiation:

- **Key recovery.** In the traditional PKI model, if a recipient loses the private key corresponding to a certificate, all messages encrypted to that certificate’s public key cannot be decrypted. In the IBE model, the server can recompute lost private keys. If messages must be recoverable for legal or other business reasons, PKI systems typically add mandatory secondary public keys that senders must encrypt messages to.

- **Group support.** Since any string can be transformed to a public key, a group name can be supplied instead of an individual identity. In the traditional PKI model, groups are either done by expanding a group to a set of individuals at encrypt time, or by issuing group certificates. Group certificates pose serious difficulties with revocation, since individuals can only be removed from a group as often as revocation is updated.

In 2001, Boneh and Franklin gave the first fully described secure and efficient method for IBE [20]. This was followed by a number of variant techniques, including Hierarchical Identity-Based Encryption (HIBE) and Certificateless Encryption. HIBE allows multiple key servers to be used, each of which control part of the namespace used for encryption. Certificateless [21] encryption adds the ability to encrypt to an end user using an identity, but in such a way that the key server cannot read messages. IBE systems have been commercialized, and are the subject of standards under the IETF (RFC 5091) and IEEE (1363.3).

19. SUMMARY

A Public Key Infrastructure (PKI) is the key management environment for public key information of a public key cryptographic system. As discussed in this chapter, there are three basic PKI architectures based on the number of Certificate Authorities (CAs) in the PKI, where users of the PKI place their trust (known as a user’s trust point), and the trust relationships between CAs within a multi-CA PKI.

The most basic PKI architecture is one that contains a single CA that provides the PKI services (certificates, certificate status information, etc.) for all the users of the PKI. Multiple CA PKIs can be constructed using one of two architectures based on the trust relationship between the CAs. A PKI constructed with superior-subordinate CA relationships is called a hierarchical PKI architecture. Alternatively, a PKI constructed of peer-to-peer CA relationships is called a mesh PKI architecture.

Directory Architectures

As discussed in this chapter, early PKI development was conducted under the assumption a directory infrastructure (specifically a global X.500 directory) would be used to distribute certificates and certificate revocation lists (CRL). Unfortunately, the global X.500 directory did not emerge resulting in PKIs being deployed using various directory architectures based on how directory requests are serviced. If the initial directory cannot service a request, the directory can forward the request to other known directories using directory chaining. Another way a directory can resolve an unserviceable request is to return a referral to the initiator of the request indicating a different directory that might be able to service the request. If the directories cannot provide directory chaining or referrals, pointers to directory servers can be embedded in a PKI certificate using the Authority Information Access (AIA) and Subject Information Access (SIA) extensions. In general, all PKI users interface to the directory infrastructure using the Lightweight Directory Access Protocol (LDAP) irregardless of how the directory infrastructure is navigated.
Bridge Cas and Revocation Modeling

Bridge Certification Authorities (BCAs) provide the means to leverage the capabilities of existing corporate PKIs as well as Federal PKIs. Public key infrastructures (PKIs) are being fielded in increasing size and numbers, but operational experience to date has been limited to a relatively small number of environments. As a result, there are still many unanswered questions about the ways in which PKIs will be organized and operated in large scale systems. Some of these questions involve the ways in which individual certification authorities (CAs) will be interconnected. Others involve the ways in which revocation information will be distributed.

Most of the proposed revocation distribution mechanisms have involved variations of the original CRL scheme. Examples include the use of segmented CRLs and delta-CRLs. However, some schemes do not involve the use of any type of CRL (on-line certificate status protocols and hash chains).

A model of certificate revocation presents a mathematical model for describing the timings of validations by relying parties. The model is used to determine how request rates for traditional CRLs change over time. This model is then extended to show how request rates are affected when CRLs are segmented. This chapter also presented a technique for distributing revocation information, over-issued CRLs. Over-issued CRLs are identical to traditional CRLs but are issued more frequently. The result of over-issuing CRLs is to spread out requests from relying parties and thus to reduce the peak load on the repository.

A more efficient use of delta-CRLs uses the model described in a model of certificate revocation to analyze various methods of issuing delta-CRLs. It begins with an analysis of the “traditional” method of issuing delta-CRLs and shows that, in some circumstances, issuing delta-CRLs in this manner fails to provide the efficiency gains for which delta-CRLs were designed. A new method of issuing delta-CRLs, sliding window delta-CRLs, was presented. Sliding window delta-CRLs are similar to traditional delta-CRLs, but provide a constant amount of historical information. While this does not affect the request rate for delta-CRLs, it can significantly reduce the peak request rate for base CRLs. The chapter provided an analysis of sliding window delta-CRLs along with advice on how to select the optimal window size to use when issuing delta-CRLs.

Finally, let’s move on to the real interactive part of this Chapter: review questions/exercises, hands-on projects, case projects and optional team case project. The answers and/or solutions by chapter can be found in the Online Instructor’s Solutions Manual.

CHAPTER REVIEW QUESTIONS/EXERCISES

True/False
1. True or False? The most important security protocols used on the Internet do not rely on PKI to bind names to keys—a crucial function that allows authentication of users and websites.
2. True or False? To understand how PKI systems function, it is not necessary to grasp the basics of public key cryptography.
3. True or False? The most important cryptographic operation in PKI systems is the digital signature.
4. True or False? Variants of the three operations used to construct digital signatures can also be used to encrypt data.
5. True or False? PKI systems solve the problem of associating meaningful names with essentially meaningless cryptographic keys.

Multiple Choice
1. What model is the most prevalent standard for certificate based PKIs, though the standard has evolved such that PKI-using applications on the Internet are mostly based on the set of IETF standards that have evolved and extended the ideas in X.509?
   A. X.510
   B. X.509
   C. X.511
   D. X.512
   E. X.513
2. What model specifies a system of Certifying Authorities (CAs) that issue certificates for end entities (users, web sites, or other entities that hold private keys)?
   A. X.510
   B. X.509
   C. X.511
   D. X.512
   E. X.513
3. While, in theory, the _________________ is the entity that creates and validates certificates, in practice, it may be desirable or necessary to delegate the actions of user authentication and certificate validation to other servers.
   A. Evolution
   B. Residue class
   C. Peer-to-peer (P2P)
   D. Certification Authority
   E. Security
4. What certificate validation is a complex process, and can be done to several levels of confidence?
   A. X.509
   B. X.510
5. The contents of the target certificate cannot be trusted until the signature on the certificate is validated, so the first step is to check the:
   A. Physical world
   B. Data retention
   C. Standardization
   D. Permutation
   E. Signature

EXERCISE

Problem

What are the benefits of PKI?

Hands-On Projects

Project

What are the problems with using PKI?

Case Projects

Problem

How does PKI provide management and control?

Optional Team Case Project

Problem

What are the core components of a PKI?

REFERENCES