The Internet public key infrastructure

by M. Benantar

Long before the advent of electronic systems, different methods of information scrambling were used. Early attempts at data security in electronic computers employed some of the same transformations. Modern secret key cryptography brought much greater security, but eventually proved vulnerable to brute-force attacks. Public key cryptography has now emerged as the core technology for modern computing security systems. By associating a public key with a private key, many of the key distribution problems of earlier systems are avoided. The Internet public key infrastructure provides the secure digital certification required to establish a network of trust for public commerce. This paper explores the details of the infrastructure.

Public key cryptography has emerged as a core technology and has been adopted in many modern computing security systems. The concept of related private and public key pairs is probably its most appealing aspect. The notion that one cryptographic operation—encryption—can be performed using one key from the pair, while the reverse transformation can only be computed using the other key in the pair, is indeed a giant step toward solving the secret key distribution problem. The proliferation of public cryptographic keys, on the other hand, needs to be achieved in a controlled fashion to ensure that public keys are securely bound to legitimate entities. The Internet public key infrastructure defines secure digital certification for public keys. This paper explores the details of this infrastructure. We begin with an overview of secret key cryptography; we then introduce the secret key distribution problem and explain how public key cryptography contributes to its resolution. Subsequently, we discuss the foundations of the Internet public key certification, the reasons it is needed, and its defining components.

Overview of secret key cryptography

By "data confidentiality" we mean an attempt to confine knowledge of the represented information within a particular set of entities, either human or programmable. Secrecy is achieved by scrambling the plaintext form of the data into a representation that perhaps has no syntax, and certainly should have no semantics.

Long before the advent of electronic systems, different methods of data-scrambling transformations, known in contemporary terms as the science of *cryptography*, have been used. A cryptographic transformation of data is a deterministic procedure by which data, in their plaintext form, are disguised to result in a ciphertext representation that does not reveal the original data. Similarly, the ciphertext can be reverse-transformed in a deterministic fashion by the target recipient so that the original data can be recovered.

Early cryptographic algorithms manipulated the plaintext input, character by character, using the methods of substitution and transposition. A substitution operation replaces a character in the input stream by another character from the alphabet set of the target ciphertext. On the other hand, a trans-

0018-8670/01/$5.00 © 2001 IBM

position, also referred to as a permutation, replaces a character from the input stream by another character of that same input and thus results in shuffling character positions and preserving all characters of the plaintext in the final ciphertext. An example of a substitution is the famous "Caesar cipher," which is said to have been used by Julius Caesar to communicate with his army. This cipher replaces each character of the input text by the third character to its right in the alphabet. Formally, this transformation consists of adding 3 to the position of the input character (modulo 26) to yield the substituting character. Figure 1 shows how this simple transformation is applied.

A transposition cipher, generally, consists of breaking the plaintext into separate blocks; a deterministic procedure is then applied to shuffle characters across the different blocks. Figure 2 illustrates a character transposition example in which the secret message "RETURN TO BASE" is first split into two blocks consisting of "RETURN" and "TO BASE," then characters are shuffled across the two blocks in a cyclic fashion to result in the ciphertext "ROTBRS TE UANE."

Even though it employs a very basic algorithm, the substitution example points to the concept of the secret key (the number of positions to shift right, 3). Keeping the key secret while divulging the algorithm will, in this case, permit the plaintext to be recovered only by exhaustively processing the key space, which simply consists of the set of integers {1, 2, 3, 4, . . . , 26}. The strength of the methods used in this era rested on the secrecy of the encryption algorithm itself.

With the advent of electronic computers, early modern cryptography carried on these same concepts, employing transposition and substitution transformations. The primary difference is that these transformations are now applied at the *bit* level, rather than the character level, of the binary representation of data. Strength of the encryption method no longer rests in the secrecy of its algorithm, but rather in the secrecy of the key used by that algorithm. This development gave rise to modern secret key cryptography, best known through the Data Encryption Standard (DES) algorithm.[1]

DES, a symmetric cipher in which the same key is used for encryption and decryption, was developed by IBM cryptographers in the early 1970s and has been adopted as a U.S. government standard since 1976.

Figure 1 A simple substitution cipher

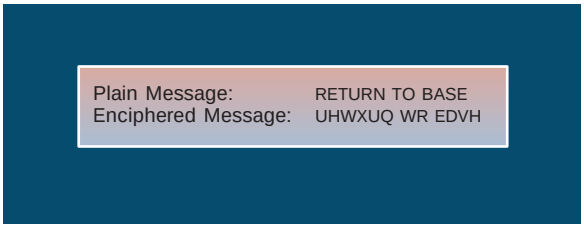
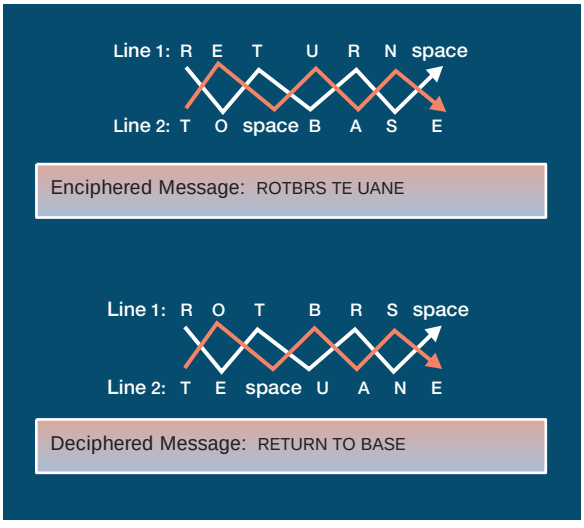
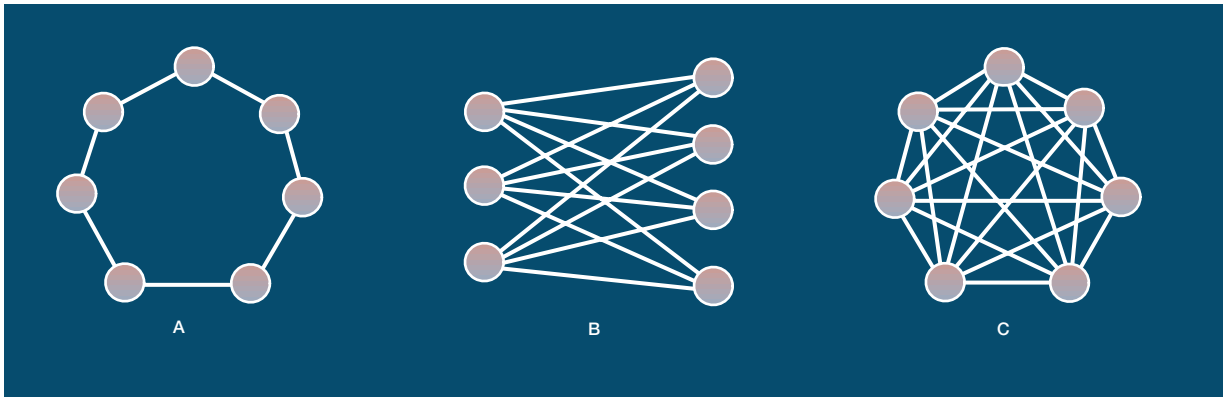


Figure 2 A character transposition enciphering/deciphering example



The algorithm is a block cipher, in which a 64-bit input block is transformed into a corresponding 64-bit output ciphertext. It employs a 56-bit key expressed as a 64-bit quantity in which the least relevant bit in each byte is used for parity checking. DES, in its standard form, iterates over 16 rounds, in each of which data are manipulated using a combination of permutation and substitution transformations along with standard arithmetic and logical operations, such as *exclusive-OR*, based upon the key. For many years the DES algorithm withstood attacks, but in recent years and mostly due to the increased speed of computing systems, DES has come under brute-force attack on several occasions, demonstrating its vulnerability to exhaustive search of the key space.[2]

Figure 3 Number of secret key distributions is directly related to the underlying communication pattern



The secret key distribution and management problem

Assume that a group of n people decides to establish a cryptographic communications channel among its members based on a symmetric cipher. Different scenarios for secret key distribution may arise within the group. In the first, all the group members decide to share a single secret key and use it to encrypt and decrypt any exchanged messages. In this basic scenario the shared secret key requires n distributions, and each user needs to manage a single key. A breach in secrecy of the key results in all communications among the group members being compromised.

In a second scenario, each group member decides to maintain a separate secret key and therefore needs to communicate it to each of the other members of the group. Here $n(n-1)$ key distributions are required, and n keys need to be stored and managed by each user. Compromising one key results in exposing all the communications destined to that key owner.

In the second scenario, the secret key of each user is divulged to the rest of the group members. One user might masquerade under another user's identity—a potential security threat. To resolve this problem, each pair of users may resort to a separate key for their communications. In a scenario where every two members of the community require a communications channel, the group must distribute $n(n-1)/2$, whereas each member must manage $n-1$ keys. Figure 3 illustrates three variations in the communication patterns that can take place within a group of seven users. Each member of the group is represented by a node of a graph, with the edge adjacency in the graph representing two-way communication links among the members. Assuming that each pair of users maintains a distinct secret key, the total number of key distributions in each scenario will be equal to the number of edges of the corresponding graph. Therefore, in (A) 7 key distributions are required; in (B), where users are partitioned along a bipartite graph, 12 key distributions are required; and in the case of complete graph (C), in which each user needs to communicate with the rest of the group, a total of 21 key distributions are needed.

These scenarios point out the fact that the number of secret key distributions among a population of users is increasingly proportionate to the number of communication links among the group. Upon renewal of a secret key, the key distribution process takes place all over again. Naturally, the more often a secret key is distributed, the more likely it is to become compromised. A compromise can occur when the key is in transmission or while it is on a storage medium. Distribution of long-term secret keys goes against the core premise of symmetric key cryptography, for which the strength lies in the secrecy of the key.

Advances in software systems have mitigated the problem posed by secret key distribution and management by adopting a central repository of keys, managed by a single server, the key distribution center (KDC). Each of the communicating entities divulges its secret key to the KDC only, resulting in n key distributions where n is the size of the community involved. The mere presence of a KDC, however,

is not sufficient to disseminate the secret keys across the community of users. A security protocol is also needed, to introduce the communicating parties to one another. The Needham-Schroeder scheme[3] presents a novel method for achieving such a secure introduction of entities. Authenticity and confidentiality of a communication is achieved through a temporary secret key, generated by the KDC. This key applies only to the active session and is shared between the two communicating entities. Advancement in this area came in a variant of the Needham-Schroeder scheme known as the *Kerberos*[4] protocol, which has proven to be one of the best third-party authentication protocols ever devised. Figure 4 illustrates the concept of the trusted third-party KDC; the number of secret key distributions necessary is always equal to the size of the community involved.

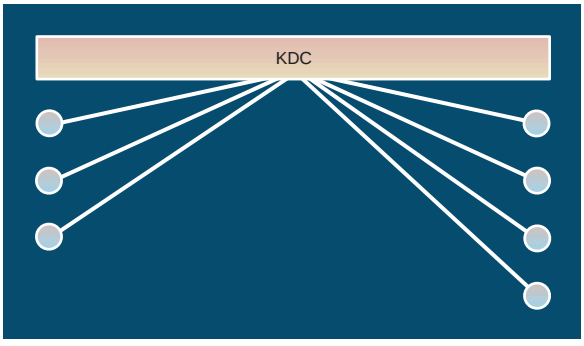
Kerberos version 5 has been integrated in a number of operating systems and has become an Internet standard. With all its protocol elegance and security, it still has a few shortcomings in today's pervasive paradigm of computing over the Internet. For one thing, the use of the third-party KDC server requires its availability to the communicating parties for the introduction step. Additionally, the KDC, in maintaining all the secret keys, becomes a single point of a catastrophic failure once it is compromised.

The problem with secret key distribution is not so much the number of distributions needed to propagate the keys; rather it is the need to find a secure channel for their distribution. Due to the recursive nature of this problem, secret key cryptographic systems alone cannot resolve the key distribution issue. In the next section we introduce *public key cryptography*, which deals with the key distribution problem.

Foundations of public key cryptography

Public key cryptography emerged in the mid-1970s with the work published by Whitfield Diffie and Martin Hellman[5] and separately by Ralph Merkle.[6] The concept is simple and elegant, yet it has had far-reaching effects on the science of cryptography and its applications. Public key cryptography is based on the notion that encryption keys come in related pairs, private and public. The private key remains concealed by the key owner, while the public key is freely disseminated. The premise is that it is computationally infeasible to compute the private key by knowing the public key—data encrypted using the public key can only be decrypted using the associated private key. The elegance and strength of public key

Figure 4 A centralized key distribution scheme



cryptography are derived from its reliance on purely mathematical foundations that are based on the one-way "trapdoor" functions that exist in the abstractions of number theory. Encryption is the easy direction; decryption is hard. With knowledge of the trapdoor, or private key, decryption can be as easy as encryption. Two of these currently known one-way functions form the basis of modern public key cryptography. We discuss these functions in the next sections.

The problem of factoring large numbers. The first of these one-way functions is based on the ease of multiplying two large prime numbers; the reverse process, of factoring a very large number, is far more complex. Factoring an integer n means finding a series of prime factors such that their product yields n. A prime number is one that has only two irreducible factors, itself and 1. Factoring large numbers (more than 1024 bits) is known to be computationally infeasible with today's computers; with modular arithmetic, the multiplication of such numbers is far easier. With this in mind, we now summarize the widely adopted Rivest-Shamir-Adleman public key algorithm, known by its acronym RSA.[7]

Randomly pick two large prime numbers p and q (of 100 to 200 decimal digits). Compute the product $n = p \times q$. Then randomly select another number e that is relatively prime with the product of $(p - 1) \times (q - 1)$; i.e., the greatest common divisor of e and $(p - 1) \times (q - 1)$ is equal to 1. Compute $\varphi(n) = (p - 1) \times (q - 1) = n + 1 - p - q$. Then use the extended Euclidean algorithm[8] to compute the multiplicative inverse d of e modulo $\varphi(n)$. The numbers e and n define the public key and are known as the *exponent* and the *modulus*, respectively, while d becomes the private key. Both encryption

and the inverse function of decryption consist simply of modular reduction operations that map an element of Z/nZ onto another element of Z/nZ, where Z/nZ is the set of integers modulo n. For any block of plaintext T, the following equations hold:

$$C = f(T) = T^e \bmod n \text{ and}$$

$$T = f^{-1}(C) = C^d \bmod n$$

Computing both $f(T)$ and its inverse is optimized using modular reduction techniques, such as exponentiation by the repeated squaring method. For instance, instead of performing seven multiplications and one large modular reduction, we perform three smaller multiplications and three simple modular reductions as follows:

$$a^8 \bmod n = ((a^2 \bmod n)^2 \bmod n)^2 \bmod n$$

Breaking the RSA algorithm is conjectured to be equivalent to factoring the product of two large prime numbers.

Computing discrete logarithms in a large finite field. The second well-known trapdoor function in number theory is the ease of computing a function f that consists of raising a number to a power in a large finite field, while the inverse function f^{-1}, of computing discrete logarithms in such a field, is known to be much harder. A finite field, also known as a *Galois* field, denoted by $GF(p)$, is the field of integers modulo a prime number p, and thus each element in the field is guaranteed to have a multiplicative inverse that is also in $GF(p)$. The time complexity required for the computation of $f(x) = a^x$ in Z/pZ is polynomial in $\log x$. Computing $x = f^{-1}(y) = \log_b (y)$, given y, is a much harder task known as the discrete logarithm problem. Here both x and y are constrained to be elements of the discrete set Z/pZ as opposed to the much easier continuous problem in the set of real numbers. A number of modern public key cryptographic algorithms are based on the discrete logarithm trapdoor function. Due to the simplicity that it exhibits, we use the ElGamal algorithm[9] as an example, even though it results in a ciphertext that is twice the size of its plaintext. Here the private deciphering key, x, is a randomly picked number, in a very large finite field $GF(p)$ such that $0 < x < p - 1$. Similarly, a second number, g, is randomly picked in $GF(p)$. The public key becomes g^x computed in $GF(p)$. Encrypting a block of plaintext T then consists of picking a third random number k that is relatively prime to $p - 1$ and computing the pair: $(C_1, C_2) = (g^k, Tg^{xk})$. To decrypt the ciphertext (C_1, C_2), we compute $T = C_2/C_1^x \bmod p$.

It is worth noting another source of one-way trapdoor functions—in recent years, elliptic curves over finite fields have been proposed for use with existing public key cryptographic systems. Elliptic curves provide a natural source for both public and private components of such systems.

The fate of secret key cryptography

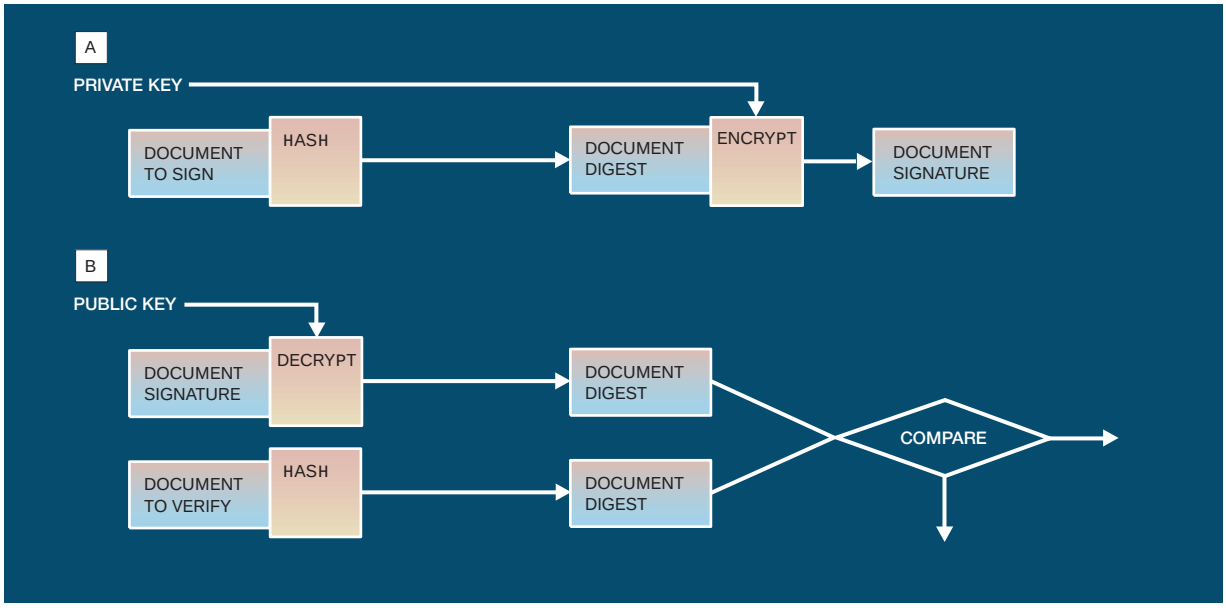
The advent of public key cryptography did not signal the end of secret key cryptography. Rather, one cryptographic method complements the other. Public and secret key cryptography together form most cryptographic protocols in use today. These are called *hybrid* cryptographic systems.[1] A public key system is used for the distribution of a secret key, which can be a long-term key or specific to a particular communications session. Thereafter, the securely distributed secret key is used to encrypt and decrypt a communications channel between two ends of a security protocol. The performance of secret key cryptography over that of public key, and the appeal of key distribution inherent to public key cryptography, are the main reasons for the wide adoption of these hybrid systems. Most notable and elegant of such systems is the Diffie-Hellman key exchange algorithm[10] used to exchange a secret key over a nonsecure channel. Recently, the IETF (Internet Engineering Task Force)[11] has proposed an algorithm that provides proof of possession of private keys by the entities engaged in the Diffie-Hellman key agreement protocol,[12] therefore strengthening the authenticity of the protocol steps.

Public key cryptography and digital signatures

Public key cryptography combined with one-way hash functions gave rise to documents with digital signatures that can withstand repudiation. A one-way hash function, $H(p)$, maps or "digests" its input p onto a fixed-length hash, h, and satisfies the following properties:

- For an arbitrary input p, it is easy to compute $H(p)$.
- It is computationally infeasible to compute the inverse $p = H^{-1}(h)$.
- It is also computationally infeasible to determine

Figure 5 (A) Generating and (B) validating a digital signature using a public key encryption algorithm such as RSA



p' such that $H(p) = H(p')$ (known as *collision-resistance*).

The goal of using a one-way hash function is to compute a unique "fingerprint" that then represents the document over which it is computed. Widely used one-way hash functions include MD5[13] and SHA-1,[14] which digest an input onto 128 and 160 bits, respectively.

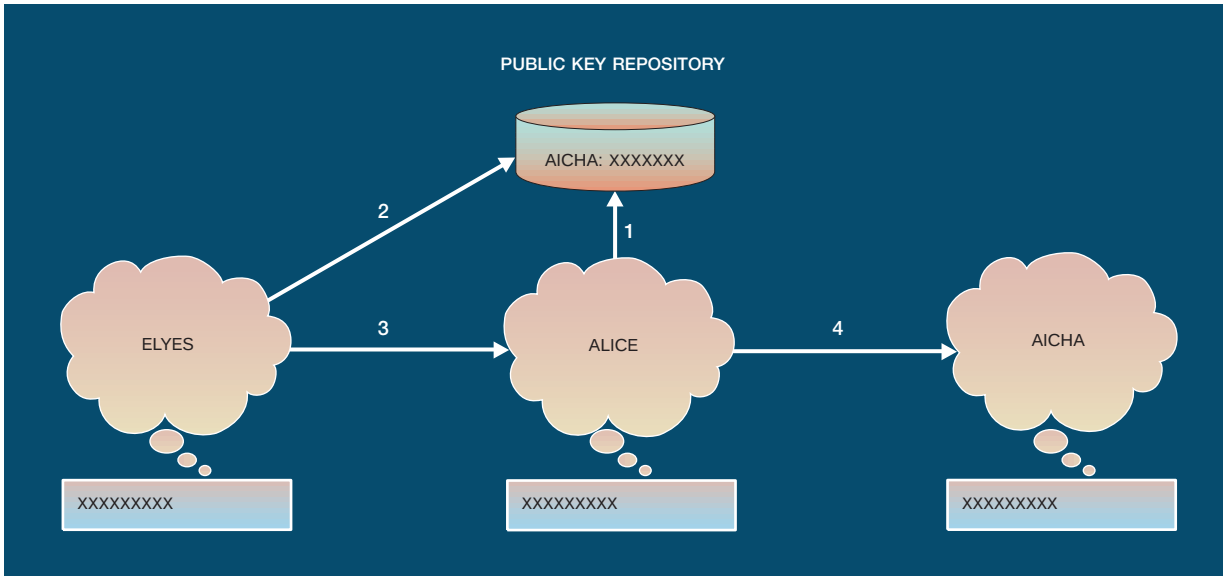
To digitally sign a document using a public key cryptographic algorithm, a hashing function is applied to the document, then the hash is encrypted using the private key of a public key pair. The premise is that the signature can only be verified using the public key corresponding to the private key used during signing. Thus, with the assumption that the private key remains confined to the secrecy of the owner, and furthermore by preventing users from obtaining direct access to their own private keys, a digital signature prevents a user from denying the signing of a document. This property is referred to as *non-repudiation* of the signing action. Preventing direct access to the private key precludes someone from intentionally disclosing his or her own private key and later denying the signing process. Commonly used digital signature algorithms are RSA and DSA.[15] Figure 5 illustrates the process of computing and validating a signature using a public key cryptographic algorithm.

By definition, verifying a digital signature automatically proves the authenticity of the signer. With its inherent support for data origin authentication and nonrepudiation, public key cryptography has taken computer security to a new level. However, there is still a weak point in binding the publicly available key to the legitimate owner of the associated private key. In the next section we provide further details.

Trusting a public key

From the outset, public key cryptography elegantly solved the key distribution and management problem introduced by secret key cryptography. Anyone can use the public key to encrypt data, but only the owner of the private key can decrypt it. The community of users in our earlier example can now adopt a public key cryptographic system for securing its communications, dispelling concerns over key distribution and simply sharing a repository that maintains the public keys of its members. Consider a scenario in which Elyes and Aicha, two members of this community, wish to communicate with each other. Elyes looks up the public key of Aicha from the repository of keys, then uses it to encrypt and send a

Figure 6 Compromising a public key



message to Aicha. A third person, Alice, wants to listen in on this communication channel, mounting the attack illustrated in Figure 6. Before any communication takes place, in step 1, Alice replaces the public key of Aicha in the key repository with her own public key. In step 2, while Elyes thinks he retrieved the public key of Aicha, he in fact now has the public key of Alice. In step 3, Elyes uses the key he retrieved in step 2, encrypts a message, and sends it to Aicha. In step 4, Alice intercepts the message, uses her private key to successfully decrypt it, reads the message, then re-encrypts it using the public key of Aicha, and forwards the message. Finally, Aicha receives the message and decrypts it using her private key, unaware of the eavesdropping.

This illustrates the weakness in using a public key without securely verifying that it indeed belongs to the designated party. It raises the fundamental question of how a secure binding can be achieved between a publicly available key and its holder so that a user of a public key, referred to as a *reliant party*, can securely verify the binding prior to using the key.

One promising answer lies in the certification process that a *public key infrastructure* (PKI) can provide. At the heart of a PKI is the digital signature technology that we introduced earlier. Parties reliant on public keys place their trust in a single entity, known as the *certificate authority* (CA). Before a user's public key is disseminated to a public repository, the underlying high-assurance CA uses its own private key to digitally sign it. A reliant party securely installs the public key of the trusted CA and uses it to verify the signature of each user's public key. Only upon a successful verification of the signature does a reliant party initiate a communications channel. This simple method of certification thwarts an attacker who does not have a public key signed by the same CA as that of the two communicating parties, but fails to do so when the attacker is also in possession of a key signed by the same CA.

In order to provide reliable assurance, a comprehensive public key certification process requires more security elements than simply a signed encryption key. These elements are embedded in the data construct to be certified. For the Internet, this construct is an *X.509 version 3* certificate, and the secure infrastructure that provides it is the public key infrastructure for X.509 (PKIX); the repository in which certificates are kept is based on the standard Lightweight Directory Access Protocol (LDAP) service. We focus on the details of this infrastructure throughout the remainder of the paper.

The Internet public key infrastructure

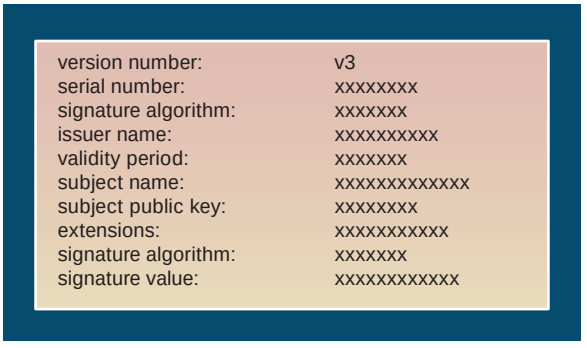
The basic premise of PKIX is the high level of assurance and confidence in a public key that it certifies. It provides a provable binding of a public key with its associated private key. In addition to the public key, such a binding implies a set of other attributes, such as a distinguished name (DN). Furthermore, the infrastructure can securely disassociate itself from a previously certified public key binding when it is deemed no longer valid for use. This disassociation is achieved by generating another construct, the certificate revocation list (CRL). A PKIX therefore manufactures two distinct digitally signed data types—a *certificate* and a *CRL*. We next describe the main data elements that define an Internet digital certificate (version 3) and its associated CRL (version 2). Figure 7 depicts the content of an Internet X.509 certificate.[16]

Three fundamental elements of the certificate contribute greatly to its secure binding with its subject: the certificate serial number, the name of the issuing certificate authority, and the subject name (owner of the certificate). The serial number is a unique integer assigned by the CA to the certificate at the time it is issued. There must be a one-to-one mapping between the serial numbers and the set of certificates issued by a particular CA. In the eavesdropping scenario we described earlier, the certificates of all three parties involved would be assigned different serial numbers, when issued, by the same CA.

The issuing CA name is a hierarchical X.500 name, for example *CN = SecureWay CA*, *OU = SecureWay*, *O = IBM*, *C = US*, that uniquely identifies the certification authority within the naming space. The issuing name is common to all certificates issued by the same CA. Similarly, the subject name is a hierarchical X.500 name, such as *CN = Elyes*, *OU = SecureWay*, *O = IBM*, *C = US*, that uniquely identifies the subject of the certificate within the naming space. Note here that the DN representation is based on RFC1485[17] and represents distinguished names in a user-oriented manner. As a good security practice, it is generally mandated that certificate holders within a particular enterprise be assigned distinct X.500 names.

It is the role of the CA to ensure that a particular subject name is not issued multiple certificates that differ only in the public key value. Multiple certificates can be issued for functionally meaningful reasons; e.g., one key for signing documents and another for enciphering data. In the absence of functionally differentiating factors, such as those defined by Certificate Policy[18] rules, a comprehensive PKIX should require that an old certificate for a subject be revoked before issuing a new one. The key to enforcing the uniqueness rule is the requirement that the CA maintain a repository of all the certificates it issues during its entire time of operation. The CA repository, or Issued Certificates List (ICL), along with a registration procedure enforced by the enterprise PKI, provides control over granting certificates. It should be noted here that a reliable PKIX is one in which certificates are always issued in a controllable way, analogous to the practice of creating users in a legacy authentication registry.

Figure 7 Core content of an X.509 v3 certificate

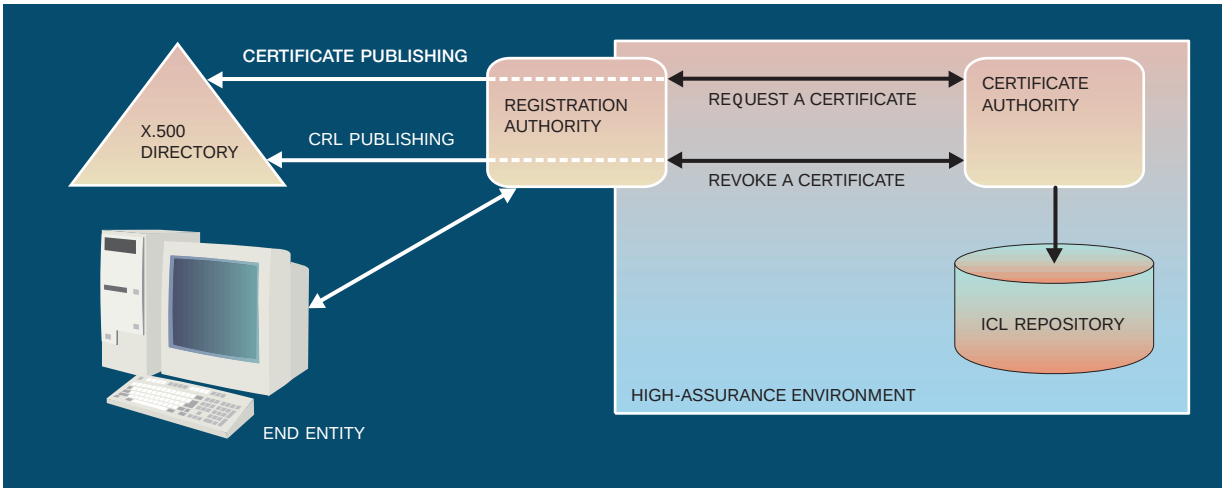


The *validity period* of a certificate is the time interval during which the certificate maintains its validity of use, provided it has not been explicitly revoked. It is defined by beginning and end dates. Time is represented according to an international standard and is computed with respect to Greenwich Mean Time (GMT), ensuring independence from the physical location of applications using PKIX.

The *subject public key* field contains the bit string representing the public key material that is being certified. The *extensions* field represents an interesting aspect of the extendability of an X.509 v3 certificate. It may contain zero or more extensions, each of which adds specific information about the certificate, such as the intended usage of the underlying public key. A number of these extensions have been defined by the IETF body. Private extensions can also be exploited within a particular enterprise.

Finally, the *signature value* field contains the CA digital signature computed over the Distinguished Encoding Rules (DERs) of the X.509 data type as rep-

Figure 8 The main PKIX participating components



resented by Abstract Syntax Notation One (ASN.1),[19] an International Telecommunication Union (ITU) standard syntax for data transfer. It is worth noting here that the CA signature certifies the entire content of a certificate and not just the public key portion of it.

One other fundamental element that builds trust in an enterprise PKIX is the ability to certify a public key for an end-user entity without requiring the corresponding private key be communicated on line or off line to the certification authority or any other entity. In most practical cases, the public and private key pair is generated at the end-user side of the infrastructure with the private key remaining securely stored in the user's local environment. The underlying storage mechanism can be chosen appropriately based upon the exploiting applications. An example of such a mechanism would be a smart card token.

The protocol underlying certification management allows for proof of possession, known as POP, of the private key to be communicated through secure cryptographic means, such as a signature, and verified by the serving CA. An enterprise-level PKIX should support the proof of private key possession in ways appropriate to the type of key usage specified by the requesting entity. For example, when the key to be certified is intended for enciphering data, the CA should use the public key to encrypt a specific data item and challenge the entity to decrypt it using the private key.

The infrastructure topology

As shown in Figure 8, a PKIX has three main components: the end entity (EE) representing the user side, the registration authority (RA) in the middle, and finally the certifying authority (CA). The RA, an optional but integral component of the CA, represents an intermediary point of trust through which an EE request is channeled to the CA. It performs a number of functions, most notably the registration of candidate certificate requesters and their authentication, in a secure fashion. The RA may also perform some processing on the EE request, such as the verification of the POP calculated using the EE private key, or the validation of certain requested certificate extensions in accordance with a policy implemented by the enterprise. The RA also represents the access point for the infrastructure's administrator to perform interactive tasks, such as registration and approval or denial of requests.

Additionally, a directory service such as LDAP is generally made available to the infrastructure for the publication of certificates and CRLs. In an enterprise PKIX, the CA interactions should be limited to the RA, the one entity with which it has established trust. The directory service should connect to the RA that performs certificate and CRL publication in response to CA notifications. Communications between the EE, RA, and CA are driven by the standard Certificate Management Protocol (CMP),[20] which is in turn based on the syntax defined by the Certificate Re-

quest Message Format (CRMF).[21] It is worth noting that, although not widely adopted, certificate management over CMS (CMC)[22] is also a proposed IETF draft standard for a certificate life-cycle management protocol. CMS[23] (Cryptographic Message Syntax) is the IETF standard for a cryptographic enveloping technique for protecting messages.

CMP is a secure protocol that carries a data protection field of its own and thus does not rely on an underlying secure transport for the security of its messages. It consists of a relatively large set of message types and is independent from the transport layer of the underlying communications system. Common implementations have been written mainly over direct Transmission Control Protocol (TCP) with some others over the HyperText Transfer Protocol (HTTP). Additionally, the industry *de facto* standards commonly known as Public Key Cryptographic Standards[24] of PKCS #10 and PKCS #7 are also widely adopted messaging formats for certificate enrollment. CMP has been widely implemented by PKIX vendors, including IBM and Lotus, and has lately been the subject of interoperation testing.

Certificate revocation

A certificate may cease to be valid for two explicit reasons: first, when the current date is not within the validity period stated in the certificate (the certificate has expired or has not yet entered use), second, when the certificate subject is no longer entitled to the certificate. In the latter case, the certificate is revoked by the issuing CA. The breakup of the binding represented in a certificate is announced by a certification authority, through a CRL. The CA periodically issues and signs CRLs, updating them with recently revoked certificates. A typical X.509 version 2 CRL contains the data illustrated in Figure 9.

A CRL shares five data types with a certificate: a version number, a signature algorithm identifier, the issuing CA name, zero or more extensions, and a signature value. The signature is computed over the flattened DER encoding of the CRL content. Each revoked certificate is identified by its serial number. Since certificate serial numbers are unique with respect to the issuing CA, a reliable PKIX revokes certificates through the same CA entity that issued them. The first time stamp encountered in a CRL indicates its date of issuance; the second one is the date for the next CRL update. The most notable component of the extensions field is the *issuing distribution point* name, a standard extension that indicates the location where the CRL is to be published. In the next section we discuss CRL distribution points in further detail.

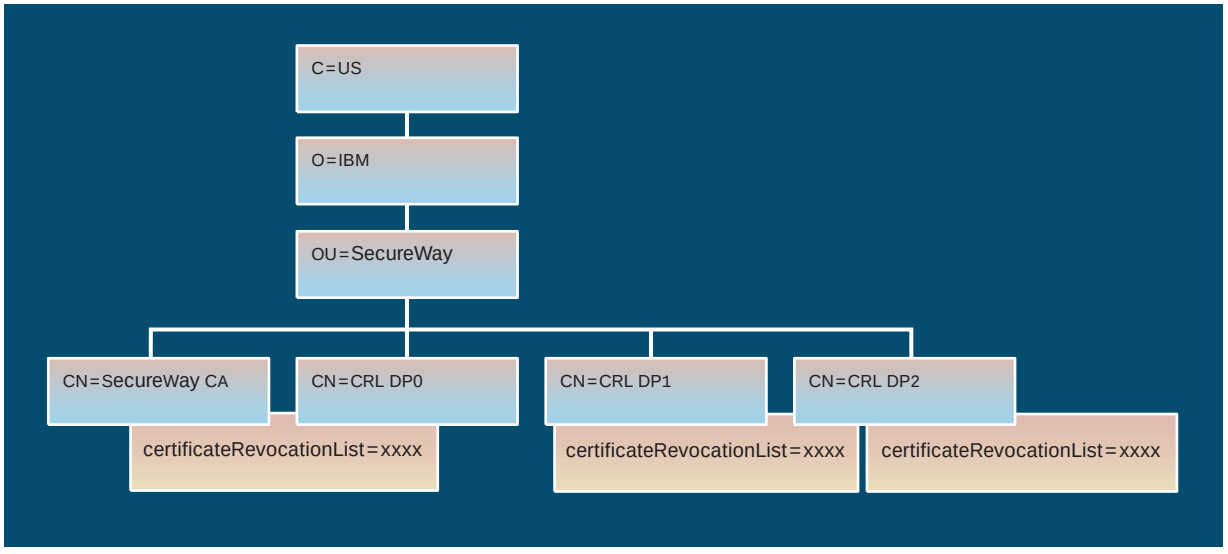
Figure 9 Core content of an X.509 v2 CRL

```
version number:                 v2
signature algorithm:            xxxxxxx
issuer name:                    xxxxxxxxxx
this update:                    xxxx
next update:                    xxxx

revoked certificates {
  certificate serial number:    xxxxxx
  revocation date:              xxxxx
  ...
}

extensions:                     xxxxxxxxxx
signature algorithm:            xxxxxxx
signature value:                xxxxxxxxxx
```

CRL distribution points. Reliable applications that use X.509 certificates are required to actively verify the validity of a certificate at the time of its use, including whether or not the certificate has been explicitly revoked. Conceptually, this verification step is a simple one; it consists of determining whether or not the certificate has become a member of any applicable CRL. Such a simple yet time-sensitive task requires a number of cooperating elements. First, where is the applicable CRL? With the globalization of today's Internet computing, interacting applications are not necessarily tightly coupled within an enterprise-specific environment.

The CRL distribution points extension is an X.509 v3 certificate extension that indicates the location of the revocation information. Distribution points, therefore, represent a bridge between the certificate and its controlling CRL. CRL distribution points also address the issue of scale that might be introduced in an enterprise that could revoke large numbers of certificates. By distributing CRLs across multiple locations, applications are able to off-load smaller portions of CRLs while performing validation. The format of a CRL distribution point name is generalized enough to allow a variety of CRL hosting services, for example, an X.500 directory name, a remote uniform resource identifier (URI) such as an LDAP URL

Figure 10 An example of three CRLs stored at distribution points, all of which are in the form of a directory name



(uniform resource locator), or some other server based on the Internet HTTP protocol.

CRL distribution points also allow redundant CRL locations, so that certificate validation is not affected by a CRL hosting server that becomes unavailable. CRLs can be stored in an LDAP server as attributes of the issuing CA. Figure 10 illustrates three CRLs located at three distribution points in the form of a directory name: CN = CRL DP0, OU = SecureWay, O = IBM, C = US, CN = CRL DP1, OU = SecureWay, O = IBM, C = US, and CN = CRL DP2, OU = SecureWay, O = IBM, C = US.

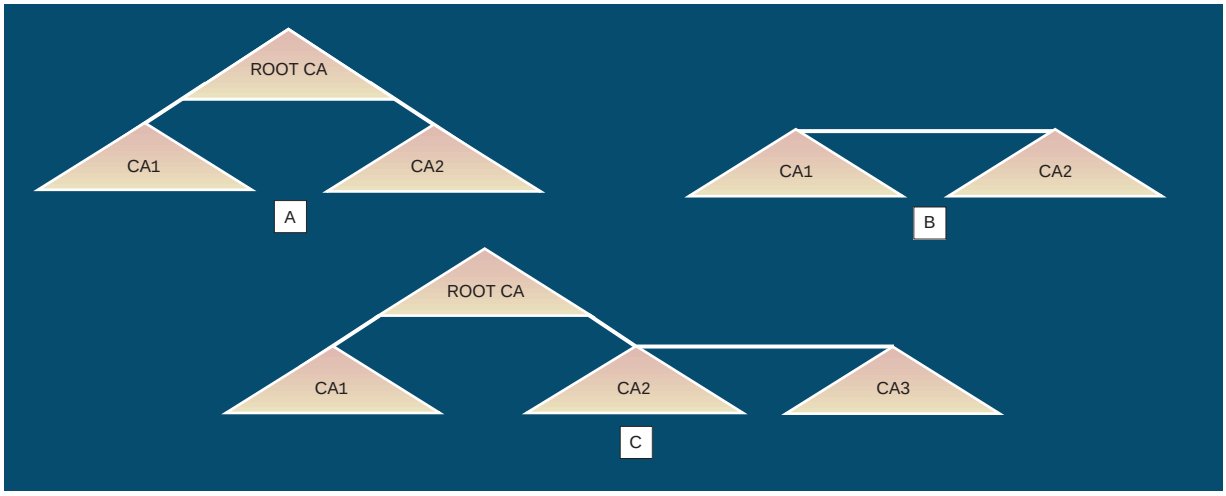
All CRLs located at these points are issued by a CA with name CN = SecureWay CA, OU = SecureWay, O = IBM, C = US. In this example the CRLs are physically stored at their respective directory names as indicated by the distribution points under the attribute *certificateRevocationList*, defined by the standard PKIX LDAP schema.[25]

Cross-domain certification

The proliferation of public key infrastructures ultimately leads to their extension across the boundaries of certification domains. Such domains may consist of multiple organizations, or departments within a single enterprise, or multiple independent enterprises. Bridging multiple domains can be driven by the need to maintain the benefits of PKI-based security in applications that support interactions across enterprises. The basic issue is to join independently deployed public key infrastructures with minimal disruption and the greatest transparency possible, allowing each certification authority to remain the sole authority for its own domain of operations.

PKIX provides two methods for achieving cross-domain certification. The first is through hierarchical certification authorities; the second is through a peer-to-peer relationship in which a CA from one domain is cross certified with a CA of another domain. Figure 11A illustrates a hierarchical relationship between two domains served by CA1 and CA2. The hierarchy implies that CA1 and CA2 are not in possession of self-signed certificates (one in which the issuer and the subject names are the same). Instead, both of these certificates are issued by the root CA bridging the two domains. (Note that the depth of a hierarchy is not limited to a single level as shown in the figure.) Validating a user certificate in the domain served by CA1 or CA2 is a recursive process that applies to the chain of certificates extending from the root CA to the user. Additionally, a certificate issued within the domain of CA1 can be validated by an application running within the domain of CA2 by similarly finding a validation path starting at the high-assurance root CA.

Figure 11 Some scenarios of cross-domain relationships: (A) purely hierarchical, (B) cross certification, and (C) a hybrid scheme



Cross certification is a peer-to-peer joining of two disparate infrastructures. As illustrated in Figure 11B, CA1 issues a cross certificate for CA2 to indicate that user certificates issued by CA2 for its population are now trusted for use within the domain served by CA1, provided of course that they pass the standard test of validity. A cross certificate is simply an X.509 CA certificate that is signed by another authority. A reliant party in the domain of CA1—such as an application server—that receives a service request with a certificate attached to it from a client in the domain of CA2 will perform the following steps:

- Determine that CA2, the issuer of the requester's certificate, is cross certified by CA1. This could be done by looking up the *crossCertificatePair* attribute in the CA2 entry of the requester's directory server.
- Validate the requester's certificate using the public key of CA2
- Validate the cross certificate using the public key of CA1 and hence establish trust in the client certificate

Similarly, CA2 can also cross certify CA1, and thus certificates issued by both certification authorities can be interchangeably consumed by applications across the two domains. Essentially, the two domains then become fully bridged. Note that each of the certification authorities remains the root CA for its own domain and each is still known through its self-signed certificate.

In the hierarchical case, each of the subordinate certificate authorities maintains a certificate issued by a higher CA. This certificate validation path spans an entire branch of the tree starting from the root down to the leaf user entity. This process is applied each time a certificate is used. In the cross-certification scenario, however, validating a certificate from an application running in the same domain does not require use of any CA cross certificates.

The hierarchical scheme establishes the cross-domain trust via a single third-party certification authority. A higher assurance in the infrastructure can, therefore, be maintained by strongly securing one entity, the root certificate authority. Prior knowledge of the legitimate validation path by an application leads to the detection of any compromise of a subordinate authority. In the cross-certification case, the same high level of assurance is required in all of the certificate authorities in order to avoid a compromise. Leaving any of the certification authorities exposed may lead to the compromise in using certificates across domains.

Note that a hybrid method that combines hierarchies and horizontal cross certification can also be used to join disparate infrastructures. Figure 11C illustrates a hybrid cross-domain topology that includes four certification authorities in which CA1 and CA2 are subordinate to a common root, while CA2 and CA3 participate in a horizontal cross-certification re-

lationship. CA hierarchies are generally recommended when joining multiple domains that require a high level of assurance.

Finally, it is worth noting that the trust relationship in the case of horizontal cross certification is not a transitive one; i.e., CA1 cross certifying CA2 and CA2, in turn, cross certifying CA3 does not imply that CA1 has cross certified CA3. Certification here is not implied; it must be explicitly designated. In the hierarchical case, however, the relationship is transitive upwards.

Certificate validation

While a certificate explicitly carries a time period that can be checked for validity with respect to the time of use, it can also become completely invalid for other reasons. A certificate may be revoked by its CA through a CRL issuance. A certificate may also be compromised by replacing it with a masquerading certificate that is issued by the same or a different authority. Certificate validation, therefore, should precede every use of the public key it certifies. Such a procedure consists of a number of validity checks, some of which are summarized here.

Validate the trust chain. Determine a trust path from the root CA to the end entity. The path can span a hybrid scheme in which both horizontal and hierarchical cross-domain certification can be in use. In a purely hierarchical trust topology, the highly assured public key of the root CA is first used to verify the integrity and authenticity of the immediate subordinate CA certificate. This is used to extract the certified public key of the immediate subordinate CA, which now inherits the highly assured trust property. The process is then recursively applied down the tree until the end entity certificate is reached. In the horizontal cross-certificate case, the certificate of the cross-certified CA is verified using the assured key of the certifying peer CA. The verified CA key is then extracted and used to verify the integrity of possibly a subordinate CA certificate or the underlying end entity certificate.

Determine the certificate revocation status. Compute a Boolean decision on the membership of the underlying certificate within an appropriate CRL. The certificate here is uniquely defined through its serial number relative to the issuing certificate authority. The verification process should check for these two elements in the set of revoked certificates represented by the CRL. Due to the reliance of this computation on the most up-to-date CRLs, which are generally stored at a central network location such as an LDAP repository, this task is gradually shifting to a specialized server. This server is known as the Online Certificate Status Protocol (OCSP) responder,[26] which specializes in computing the Boolean decision of a particular certificate membership in the published CRLs.

Certificate validation should precede every use of the public key it certifies.

Determine the certificate usage. Validate the use of the certificate against any applicable policy. An example would be to compare the cryptographic function for which the public key is about to be used against the key usage extension found in the certificate.

Managing the private key

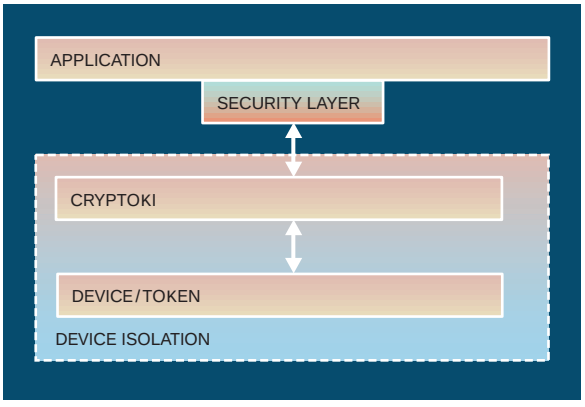
While a public key infrastructure provides a reliable solution to the key distribution problem, a PKI user remains in control of his or her private key material. A breach in the private key eventually leads to the total compromise in security of any communication channel that is governed by the certificate associated with the private key. The process of managing the private key generally centers around the method by which the key material is stored in some encrypted form. Perhaps the simplest of such methods is the wrapping provided by the PKCS #8[24] *de facto* standard where the private key is encrypted using a master key derived from a user password, which is never maintained in any stored form. A more thorough solution that is similar in concept is provided by the PKCS #12[24] standard, which has become widely used as an off-line means of transporting PKI credentials of a user, including certificates and private keys. The wrapping of private keys that is employed here can also be based on secret key encryption where the encryption key is derived from a password.

One method that brings a higher level of assurance and reliability to PKI-based applications is embod-

ied in the *de facto* standards of PKCS #11 and PKCS #15[24] for managing and manipulating private keys. The main concept adopted is the isolation of any manipulation of the private key by a security layer of an application to within the confines of a single component (usually in the form of a shared dynamic library). The component provides a set of cryptographic operations and interacts directly with a storage medium called a credentials token, widely implemented as a hardware device. Software implementations of the token are also available.

Existing Internet browsers are good examples of applications that provide software implementations of the PKCS #11 tokens. As illustrated in Figure 12, the PKCS #11 library, commonly known as *cryptoki* (short for cryptographic token interface), provides for a common logical view of the cryptographic token. When invoked by a security layer of an application, through a standard set of interfaces, the PKCS #11 library interacts with the device driver of the token for cryptographic services based upon the user's public and private keys. PKCS #11 seeks not only to maintain secure storage of a private key through a secret personal identification number (PIN) access, but equally important it provides a portable security layer that isolates users from the details of the hardware tokens in which public key credentials reside.

Figure 12 Isolating the manipulation of a private key



Jonah: The IBM PKIX reference implementation

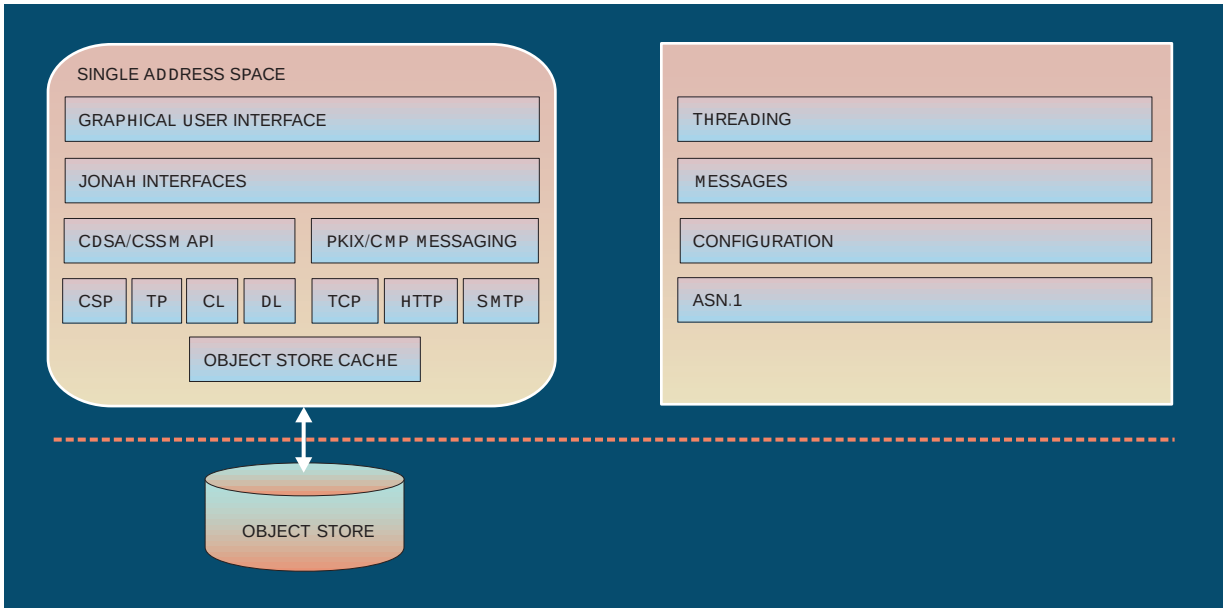
IBM and Lotus Development Corporation demonstrated commitment to the evolving PKIX standards through a project dubbed *Jonah*[27] that implements a core set of PKIX. Jonah was subsequently made available as freeware to the Internet development community. The philosophy behind this project is to encourage the industry adoption of the PKIX-based security technologies and thereby advance the state of e-business over the Internet. As a reference implementation, the Jonah architecture shied away from proprietary software systems and instead relied on standard components. Most notable is its use of Intel's Common Data Security Architecture[28] (CDSA). This specification has been adopted by the Open Group and is backed by a publicly available reference implementation.

CDSA consists of a framework that supports dynamic loading of pluggable low-level services that provide cryptographic functions, known as the Cryptographic Service Providers (CSPs), data storage and retrieval of security constructs such as cryptographic keys and certificates (Data Library, or simply DL), as well as interfaces for the manipulation of certificates (Certificate Library, or CL) and for the verification of Trust Policies (TPs) that an enterprise may elect to enforce. This wide range of security services is accessible through a consistent layer of programming interfaces referred to as the Common Security Services Manager (CSSM). Figure 13 illustrates the unified Jonah run-time architecture that is logically replicated across each of the three services provided by Jonah: the EE, the RA, and the CA.

The model is logical, in that each of the services performs the functions associated with its role, but all share a common architecture. The EE, for example, consists of a lightweight execution run-time environment with a smaller footprint than that of the RA or the CA. The RA and the CA are multithreaded servers that provide the PKIX transactional functions of the certificate life cycle. The RA securely preregisters users, validates any transactions requested by the EE against an existing policy, and approves such requests, which it then forwards to the CA. The CA, having established a trust relationship with its RA through an earlier enrollment process, issues and signs a certificate or a CRL. While it is the CA that drives the LDAP publishing events, the RA interacts with the LDAP server that publishes certificates and CRLs.

Each of the Jonah servers is controlled via its graphical user interface (GUI) layer that starts and configures the server in its intended role. Jonah CMP implementation is over direct TCP. An abstraction layer is created around the socket interface to easily al-

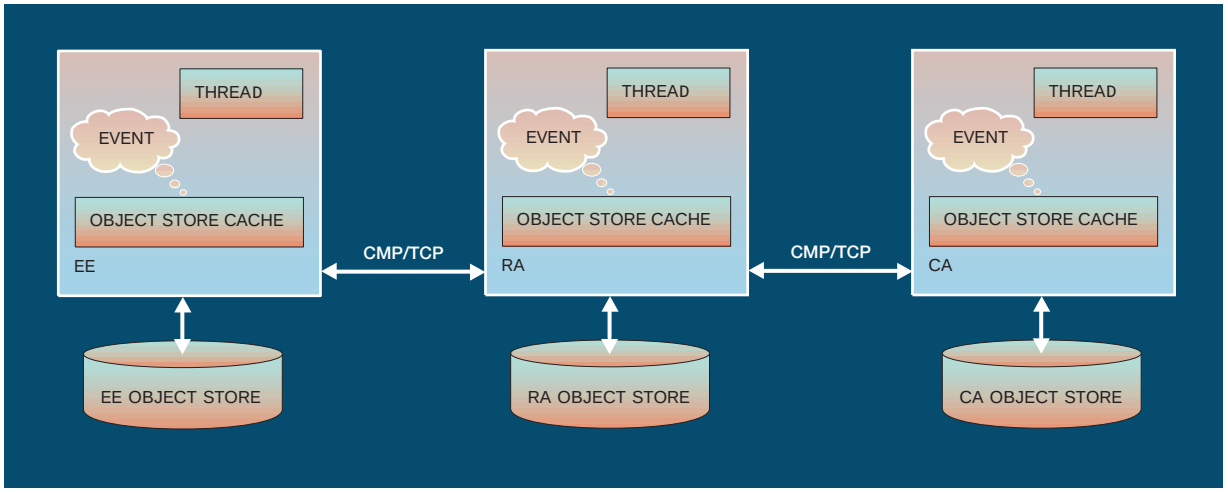
Figure 13 The unified Jonah servers architecture



low for additional transports defined for CMP, such as HTTP and SMTP (Simple Mail Transport Protocol). As we discuss in the next section, the separation of the CA from the RA and the adoption of a transactional model that is based on an object store prevent the loss of PKIX transactions and allow the CA to run in off-line mode. The core functionality of the Jonah RA and CA servers was subsequently integrated into the IBM/Tivoli PKI Trust Authority[29] product offering. IBM Trust Authority provides a robust and reliable registration and approval framework of PKIX client entities as well as a rich set of certificate life-cycle management functions that can be initiated through a browser-based or a native graphical interface. Two Jonah developments are worth noting here: the adoption of a computational model centered on an object store and the development of a well-structured native ASN.1 library.

The Jonah object store. Naturally, certificate enrollment transactions may require some intervention for human approval of a request. Such intervention may incur prolonged service time. Instead of blocking the initiator, CMP was designed as a polling protocol that guarantees a synchronous response to a request. If a service response cannot be completed, it sends back a polling response that tells the initiator to periodically poll until the request is serviced. The polling nature of CMP, along with the possibility of PKIX servers going off line, led to Jonah's architecture around a nonvolatile object store as a medium for representing and maintaining all the PKIX transactions (Figure 14). Processing threads are spawned in response to events detected by monitoring state changes in the object store entities. For instance, an end entity may construct a certificate request by creating an object representing the request, storing it in the object store, then submitting the request to the RA and marking the underlying object as a *surrogate* in the object store indicating that it is *active* in another server, the RA in this case. When a polling response is returned, the request object is updated with polling information that will subsequently drive a series of polling requests until a complete response is received, at which time the object is marked *complete*. Similarly, when the RA server first receives the request it parses it in order to determine its type. It then creates an object in its object store representing the request, marking it *active*. An event is then triggered to dispatch a thread that processes the request. To aid the performance of Jonah servers, the object store on disk is backed by a run-time cache that maintains a subset of the objects representing recently active transactions. Jonah transactional messages are immediately mapped into entries in the object store. As long as they are captured in the object store, Jonah

Figure 14 Capturing PKIX transactions using the Jonah object store model



PKIX transactions are prevented from loss even when a participating Jonah server goes off line.

Jonah processing: Computing over ASN.1 objects. All of the PKIX constructs, including protocol formats, are described in terms of ASN.1 data types. The Jonah developers have designed and implemented an ASN.1 class library for parsing ASN.1 encodings and building run-time ASN.1 C++ objects. The library directly implements the ASN.1 primitive types, such as *character string* and *integer*, and provides a well-structured class hierarchy for building up constructed ASN.1 types such as sequences and sets. High-level PKIX objects (e.g., a certificate or a CRL) are easily defined. Functions needed for the manipulation of a newly defined high-level object become immediately available through the generic methods defined by the class hierarchy. The use of C++ polymorphism allows Jonah ASN.1 objects, where applicable, to be manipulated through common interfaces that in turn behave differently based upon the type of the object. The computational model in Jonah presents a uniform and consistent aspect for transforming and manipulating first-class ASN.1 C++ objects, including object store processing.

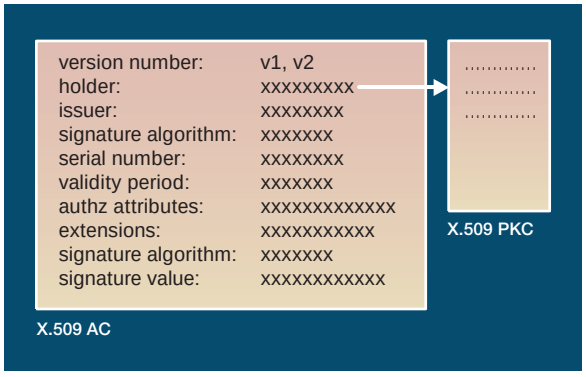
Attribute certificates: The next evolution of PKIX

A Public Key Certificate (PKC) serves as the basis for the authentication of the subject it identifies. Provided a user's certificate is validated with respect to a trusted root authority, a simple method, such as a verifiable digital signature, leads to proof of possessing the underlying identity, otherwise known as authentication. Secure identification is the key precondition for granting access to controllable resources. A resource manager will attempt to enforce an access control policy only after a successful authentication.

The X.509 attribute certificate[30] (AC), of which an X.509 certificate is a fundamental part, seeks to certify or securely bind a set of authorization capabilities to a subject, in the same way that an X.509 PKC binds a public key to its subject. The distinction between these two types of certificates is dictated by the dynamic nature of authorization roles that a particular entity can assume over time, while possessing the same public key certificate. Figure 15 shows how a trust path is established in an AC by tracing back the holder to the associated X.509 PKC.

It is worth noting that the authority issuing an AC uses the public key certified by its PKC in order to sign it. Thus, the validation path of an AC ultimately requires the presence of the holder's PKC and the PKC of the issuer of the AC, as well as a valid trust chain to the root signing authority for each of the public key certificates. The set of authorization privileges that an AC certifies are defined by an ASN.1 *Attribute* type, which is a sequence of (key, value) pairs identifying access right types and their associated values; for example, a *clearance* is identified

Figure 15 Profile of the attribute certificate



through an ASN.1 object identifier of {2 5 1 5 55} whereas its value may carry a security labeling based on a multilevel security policy.

Conclusion

Public key cryptography is rapidly transforming the technical aspects surrounding systems and network security. The concept is based on mathematical foundations and is computationally reliable, simple, and elegant. The ramifications are far-reaching. The Internet public key infrastructure for X.509 certificates is an attempt to remedy the lack of assurance in the public key. The underlying PKIX technologies providing the solution are robust and promising. Internet-based e-business applications are quickly adopting PKIX as the cornerstone of their security solutions. While PKIX attribute certificates for authorization credentials are evolving, enterprises will largely rely on mappings between X.509 public key certificates and local identities, so that legacy access control systems can remain in effect.

Acknowledgment

The author wishes to thank the anonymous reviewers for their helpful suggestions.

Cited references

1. B. Schneir, *Applied Cryptography*, John Wiley & Sons, Inc., New York (1996).
2. See http://www.distributed.net/des/release-desiii.txt, "US Government's Encryption Standard Broken in Less than a Day."
3. R. M. Needham and M. D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Communications of the ACM* **21**, No. 12, 993–999 (December 1978).
4. J. T. Kohl and B. C. Neuman, The Kerberos Network Authentication Service, IETF RFC1510 (September 1993). See http://www.ietf.org/rfc/rfc1510.txt.
5. W. Diffie and M. E. Hellman, "Multiuser Cryptographic Techniques," *Proceedings of AFIPS National Computer Conference*, AFIPS Press, Montvale, NJ (1976), pp. 109–112.
6. R. C. Merkle, "Secure Communications over Insecure Channels," *Communications of the ACM* **21**, No. 4, 294–299 (April 1978).
7. R. L. Rivest, A. Shamir, and L. M. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM* **21**, No. 2, 120–126 (February 1978).
8. N. Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, New York (1994).
9. T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *Advances in Cryptology: Proceedings of CRYPTO 84*, Volume 196 of *Lecture Notes in Computer Science*, G. R. Blakley and D. Chaum, Editors, Springer-Verlag, Berlin (1995), pp. 10–18.
10. W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory* **IT-22**, No. 6, 644–654 (November 1976).
11. See http://www.ietf.org.
12. H. Prafullchandra and J. Schaad, Diffie-Hellman Proof-of-Possession Algorithms, IETF RFC2875 (July 2000). See http://www.ietf.org/rfc/rfc2875.txt.
13. B. Schneir, "One-Way Hash Functions," *Dr. Dobb's Journal* **16**, No. 9, 148–151 (September 1991).
14. Secure Hash Standard, Federal Information Processing Standards Publication 180-1 (1995). See http://www.itl.nist.gov/fipspubs/fip180-1.htm.
15. Digital Signature Standard, National Institute of Standards and Technology Publication 186 (1994). See http://www.nist.gov/public_affairs/releases/digsigst.htm.
16. R. Housley, W. Ford, W. Polk, and D. Solo, Internet X.509 Public Key Infrastructure Certificate and CRL Profile, IETF RFC2459 (January 1999). See http://www.ietf.org/rfc/rfc2459.txt.
17. S. Hardcastle-Kille, A String Representation of Distinguished Names, IETF RFC1485 (July 1993). See http://www.ietf.org/rfc/rfc1485.txt.
18. S. Chokhani and W. Ford, Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework, IETF RFC 2527 (March 1999). See http://www.ietf.org/rfc/rfc2527.txt.
19. Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1) (1998). See http://www.itu.int/itudoc/itu-t/rec/x/x200-499/x208.html/.
20. C. Adams and S. Farrell, Certificate Management Protocols, IETF RFC2510 (March 1999). See http://www.ietf.org/rfc/rfc2510.txt.
21. M. Myers, C. Adams, D. Solo, and D. Kemp, Internet X.509 Certificate Request Message Format, IETF RFC2511 (March 1999). See http://www.ietf.org/rfc/rfc2511.txt.
22. M. Myers, X. Liu, J. Schaad, and J. Weinstein, Certificate Management Messages over CMS, IETF Draft (February 2001). See http://www.ietf.org/internet-drafts/draft-ietf-pkix-2797-bis-00.txt.
23. R. Housley, Cryptographic Message Syntax, IETF RFC2630 (June 1999). See http://www.ietf.org/rfc/rfc2630.txt.
24. RSA, Inc., http://www.rsasecurity.com/rsalabs/pkcs. See PKCS #7, PKCS #8, PKCS #10, PKCS #11, PKCS #12, and PKCS #15.

25. S. Boeyen, T. Howes, and P. Richard, Internet X.509 Public Key Infrastructure Operational Protocols—LDAPv2, IETF RFC2559 (April 1999). See http://www.ietf.org/rfc/rfc2559.txt.
26. M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, X.509 Internet Public Key Infrastructure Online Certificate Status Protocol—OCSP, IETF RFC2560 (June 1999). See http://www.ietf.org/rfc/rfc2560.txt.
27. M. E. Zurko, J. Wray, I. Morrison, M. Shanzer, M. Crane, P. Booth, E. McDermott, W. Macek, A. Graham, J. Wade, and T. Sandlin, "Jonah: Experience Implementing PKIX Reference Freeware," *Proceedings 8th USENIX Security Symposium*, Washington, DC (August 23–26, 1999), pp. 185–200.
28. Common Data Security Architecture, Intel Architecture Labs. See http://developer.intel.com/ial/security/specifications.htm.
29. *IBM SecureWay Trust Authority Version 3 Release 1.2 User's Guide*, IBM Corporation (June 2000).
30. S. Farrell and R. Housley, An Internet Attribute Certificate Profile for Authorization, IETF Draft (January 2001). See http://www.ietf.org/internet-drafts/draft-ietf-pkix-ac509prof-09.txt.

Accepted for publication May 4, 2001.

Messaoud Benantar *IBM/Tivoli Security Business Unit, 9442 Capital of Texas Highway North, Arboretum Plaza One, Suite 500, Austin, Texas 78759 (electronic mail: mbenanta@us.ibm.com or mbenanta@us.tivoli.com).* Dr. Benantar is a senior software engineer with the IBM/Tivoli security business unit. He received his *diplome d'ingenieur en informatique* from the Université des Sciences et de Technologie, Bab Ezzouar, Algiers, Algeria, in 1983, and an M.S. degree in 1986 and a Ph.D. degree in 1992, both in computer science from Rensselaer Polytechnic Institute in Troy, New York. His interests include systems and network security, Internet computing, combinatorial algorithms, and graph theory and its applications.