Chapter 3

Elements of Trust Paradigms in Computing

Introduction

Assurance in an identity is established by way of authenticating it. The entity claiming to hold a particular identity asserts its claim by providing verifiable information to the authenticating entity. Trust in identity authentication is founded on computing the following assertion: *The entity performing authentication is presented with information that only the entity being authenticated is able to provide.* This information is referred to as *proof of possession* (POP) of identity. The authenticating entity establishes trust in this process through a secure verification of the presented proof.

While in Chapter 1 we discussed various authentication factors, the POP of an identity has traditionally been based on shared secrets or derivatives thereof, something the holder and the verifier of the identity know. The advent of public key cryptography has led to establishing identities without having to disseminate shared secrets, provided assurance in the binding between a public key and the identity being authenticated can be reliably established. Advances in network-distributed computing have pushed the scope of an established identity beyond the boundaries of hosting systems and local networks to larger networks as wide as the Internet. An established identity yields a verifiable security context, the strength of which depends on the processes involved in providing an identity POP. We refer to the components that establish and maintain the flow of secure contexts as *identity trust mechanisms*.

We survey the major paradigms and mechanisms of identity trust in computing. The objective is to highlight and classify the core techniques known to date. Although some specific ones are broadly discussed, we do not intend to enumerate all known techniques. Even when the elegance, strength, and soundness of one method or another can be apparent, we do not recommend a specific one. The intent is to expose the elements of trust that characterize each method.

Although other aspects such as policy management and enforcement as well as access-control subsystems are all relevant to trust [ABAD93, BLAZ96, BLAZ99, GRAN00, LAMS01, GRAN02], it is evident that trust in identity is the gate to all other factors of trust-management systems. As such, our definition of trust here is specific to the confidence and assurance in an identity. Trust in real-life practices is relative and can be rated along a continuum scale varying from weak to strong [SHAN02]. Trust forms an inverse relationship with the level of risk that can be associated with processes, programming agents, and individuals [KONR99]. Trust as it relates to identity is a reflexive relationship but not always transitive, symmetric, or associative. However, *transitive trust,* also referred to as *delegation,* can be a key requirement along a particular chain of computing tasks in the same way it can be relied on by individuals accomplishing manual processes.

Brokered trust or *trust through a third party* has emerged as one of the key trust paradigms. We classify third-party authentication schemes in two categories. We refer to the first one as the explicit model, while we call the other one implicit. We give examples of each, with detailed descriptions of the trust elements of Kerberos being the most elegant of third-party authentication protocols. The details of trust in the public key model including the Internet public key infrastructure are presented. We conclude by reviewing three mechanisms for expressing and conveying trust over the web. These are the emerging Web services security, the security assertion markup language, and Web cookies.

A Third-Party Approach to Identity Trust

The local paradigm of identity management, as we discussed in the previous chapter, implies that user-identity information be maintained in the user registry of every system used. Furthermore, a user's shared secret under which the element of trust is built (e.g., a password) is expected to be different for each system accessible by that user in order to minimize the scope of a potential compromise. The complexity of managing multiple passwords and secrets, therefore, increasingly becomes an inconvenience to end users as well as to programming agents that rely on them.

Local identity management recognizes each identity as a local construct that is defined within the scope of the system in which it is known. Identity- and trust-management relations in this case can be modeled as a bipartite graph in which n users and m computing systems are tied through the shared secret relationship. As Figure 3.1 illustrates, this requires managing $n \times m$ relations.

The complexity and lack of scalability inherent to the local identity- and trust-management model has led to the emergence of the third-party authentication scheme. Here a single host in a networked environment is designated as the sole entity trusted by all of the participants in the network, such as users, computing systems, and applications. The user registry maintained by this third-party service contains identity information for all network participants. Trust is founded on the secret shared between each entity and the

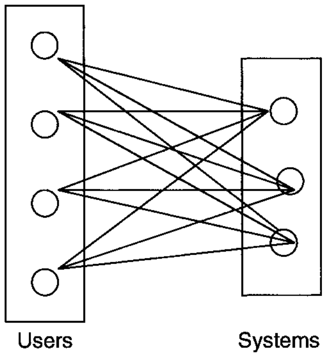


FIGURE 3.1 Managing secret sharing relationships in the local identity model

third-party authentication service. No entity in the network has any direct trust relationship with any of the other entities. Two authentication paradigms that are based on third-party have emerged:

- Implicit authentication by secure introductions of entities to one another via a known and trusted third party-entity and
- Explicit authentication of an entity by invoking a third-party authentication service.

In the first scheme, authentication is cryptographically deduced from the secret shared by an entity and the third party, while in the second case, authentication is explicitly requested from a third party by the authenticating entity. Figure 3.2 illustrates the secret sharing relationships that are in place when an implicit third-party authentication scheme is in use. Providing authentication across n users and m computing services requires managing $n + m$ secrets, a considerable decrease from $n \times m$ required for direct identity relationships between users and destination systems and services.

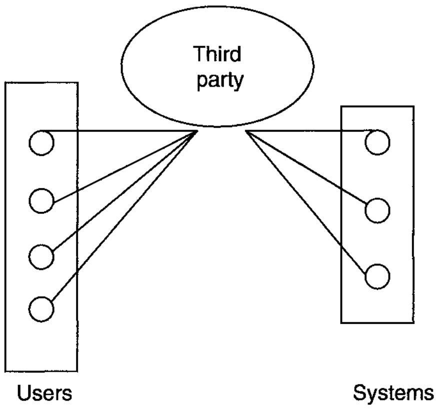


FIGURE 3.2 Reducing the complexity of managing cross-entity authentication relationships using a third party

Essentially, a third-party authentication scheme recognizes two broad entities:

- A third-party authentication service and
- The rest of all other entities.

All of the entities participating in a third-party authentication realm form peer relationships to one another with respect to authentication. As shown in Figure 3.3, the differences between entities of a third-party authentication realm are inexistent. The third party has a consistent view across all entities regardless of whether an entity acts as a client or a server. Each of such entities is now abstracted under the term of a *principal.*

Below we discuss the Kerberos authentication protocol as being the most reliable and well-known third-party authentication system to date. Kerberos follows the implicit authentication paradigm, as we outlined above. We also discuss the mechanisms suited for the third-party authentication that fall along the explicit paradigm.

Kerberos: The Implicit Third-Party Authentication Paradigm

Kerberos is the name that became famously associated with the third-party authentication protocol developed at the Massachusetts Institute of Technology (MIT) in the 1980s. The ideas preceding Kerberos go back to the work published by Roger Needham and Michael Schroeder, in which the third-party authentication concept was introduced [NEED87]. Here a third-party key distribution center (KDC) is trusted by every entity participating in a distributed computing environment to maintain its secret key (i.e., every entity shares its secret key with the KDC). As a result, the trusted KDC

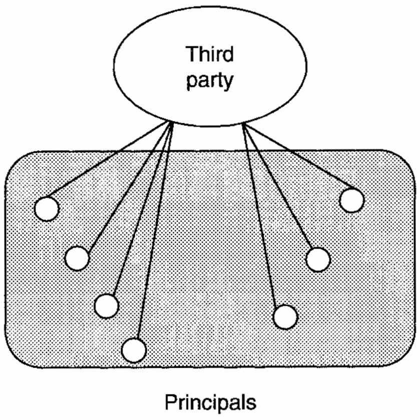


FIGURE 3.3 Peer-to-peer authentication relationships enabled by a third-party scheme

becomes responsible for the secure introduction of the participating network entities to one another. Trust is founded on the simple fact that two entities A and B that wish to communicate with one another are introduced to each other by the trusted KDC. Trust is not assumed. It is rather computed based on the following:

Entity A whose secret key is known to the key distribution center authenticates itself to the KDC by presenting its proof of possession. The KDC, also knowing the secret key of entity B (peer of A), communicates its authentication of entity A to entity B (indirectly via entity A). Trust in this communication is based on a channel encrypted with a key derived from the secret key shared between the KDC and entity B.

A High-Level View of the Kerberos Protocol

Three entities are engaged in the Kerberos protocol sequence:

- ❑ An initiating client,
- ❑ The third-party Kerberos server acting as the KDC, and
- ❑ The target entity, such as an application server.

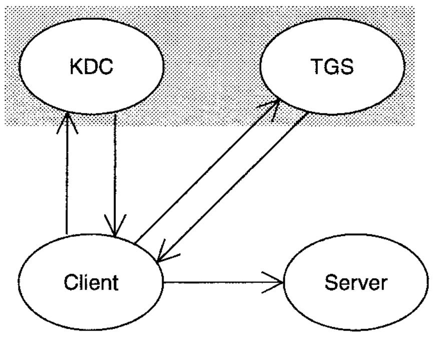
A successful execution of the protocol steps results in the authentication of the client to the application server, via the third party, and establishes a message protection channel that is governed by a secret session key between the two entities. Kerberos v5 has evolved into an Internet standard that is widely implemented [KOHL93].

The underlying data construct used in Kerberos is called a *ticket*. A client c establishes its identity with a target server s by presenting a ticket denoted by $T_{c,s}$ issued by the Kerberos server and an authenticator denoted by A_c. The authenticator protects from replay attacks and indicates the freshness level of its accompanying ticket by carrying a timestamp.

In the first message of this protocol sequence, the client contacts the KDC, identifies itself and, presents a nonce such as a timestamp or some nonrepeating value identifying the request. On receipt of the message, the KDC generates a random encryption key $K_{c,tgs}$, called a *session key*, and constructs a special ticket, the ticket-granting ticket (TGT), intended for use with the ticket-granting service (TGS), a component of the Kerberos server. The TGT identifies the client, contains a session key, and indicates the lifetime of the ticket (start and expiration times). The ticket is then encrypted using the secret key K_{tgs} of the TGS that it shares with the KDC and is sent in the response to the client. In addition to the ticket for the TGS, the response includes the session key and a nonce, both of which are encrypted in the client's secret key K_c (a derivative from the client's password). The client receives the response, decrypts the portion that is encrypted using its secret key, and thus unravels the session key $K_{c,tgs}$, used to establish an encrypted channel with the TGS.

The acquisition of the ticket first for the TGS instead of a target application server is introduced to reduce the risk of exposure of the client's secret key K_c. Once a TGT for the TGS is acquired, the client has no need to keep a copy of its secret key in the runtime environment. With respect to clients, the TGS represents no distinction from any server, such as one representing a business application. The TGS represents a logical distinction from the KDC but is physically colocated on the same host and has access to the same registry of keys, as does the KDC. Furthermore, both the KDC and the TGS can be implemented as separate components that run in the same address space.

A client that has successfully acquired a TGT for the TGS becomes ready to request tickets for participating target-application servers. On each such request, the client presents its TGT to the TGS and identifies the target application. The TGS verifies the ticket, along with the authenticator and the associated request information. It then replies with a ticket for the target application. The reply is protected using the session key with the TGS (as determined from the TGT). The client uses its session key with the TGS to extract its new session key with the target service. It forms a fresh authenticator, encrypts it with the session key, and sends it along with the ticket to the target application. If the client requests mutual authentication from the server, the server responds with a fresh message encrypted using the session key. This establishes the fact that the server used its own secret key to decrypt the ticket and determine the session key. Figure 3.4 illustrates the steps of the Kerberos V5 protocol.



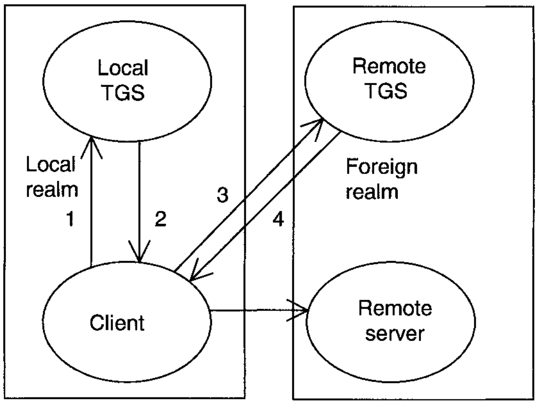
(1) Client $\rightarrow$ KDC: *c, tgs, nonce*
(2) KDC $\rightarrow$ Client: $\{K_{c,\,tgs}, nonce\}\ K_c, \{T_{c,\,tgs}\}\ K_{tgs}$
(3) Client $\rightarrow$ TGS: $\{A_c\}\ K_{c,\,tgs}, \{T_{c,\,tgs}\}\ K_{tgs}, s, nonce$
(4) TGS $\rightarrow$ Client: $\{K_{c,\,s}, nonce\}\ K_{c,\,tgs}, \{T_{c,\,s}\}\ K_s$
(5) Client $\rightarrow$ Server: $\{A_c\}\ K_{c,\,s}, \{T_{c,\,s}\}\ K_s$

FIGURE 3.4 Kerberos V5 protocol steps

Federated Kerberos

Each Kerberos server is responsible for providing secure identity and trust management to a single realm. A realm has well-defined network boundaries and is made of a finite number of participating entities, such as hosts and applications. A large network may suffer from the bottleneck exhibited by a single Kerberos server managing identity trust for the entire network. Scalability of Kerberos can be an issue for large networks. Kerberos addresses this problem by dividing a large network into separate domains; each is supported by its own Kerberos server. Cross-domain relationships are provided by the inter-realm trust feature of Kerberos. This feature enables a client from one realm to obtain a ticket for a service that resides in another realm, referred to as a *foreign realm*. The aggregation of all realms in this fashion makes it seem like a single large domain of trust.

Interrealm trust in Kerberos is based on sharing secret keys between ticket-granting services of cooperating Kerberos domains. Recall that each TGS is like any other entity with respect to its local KDC. A client obtains a ticket for a server in a foreign realm by first obtaining a TGT to the remote TGS from its own local KDC. Figure 3.5 illustrates the protocol steps used by Kerberos V5 in support of the cross-domain trust relationship. It is assumed that the client is already in possession of a TGT to its local TGS.



(1) Client $\rightarrow$ TGS_{local}: $\{A_c\}\ K_{c,\,tgs}, \{T_{c,\,tgs}\}\ K_{tgs}, tgs_{remote}$
(2) $\text{TGS}_{local} \rightarrow$ Client: $\{K_{c,\,tgs\ remote}\}\ K_{c,tgs}, \{T_{c,\,tgs\ remote}\}\ K_{tgs\ remote}$
(3) Client $\rightarrow$ TGS_{remote}: $\{A_c\}\ K_{c,\,tgs\ remote}, \{T_{c,\,tgs\ remote}\}\ K_{tgs\ remote}, s_{remote}$
(4) $\text{TGS}_{remote} \rightarrow$ Client: $\{K_{c,\,s}, remote\}\ K_{c,\,tgs\ remote}, \{T_{c,\,s\ remote}\}\ K_{s\ remote}$
(5) Client $\rightarrow$ Server_{remote}: $\{A_c\}\ K_{c,\,s\ remote}, \{T_{c,\,s\ remote}\}\ K_{s\ remote}$

FIGURE 3.5 Kerberos protocol steps for cross-realm establishment of trust

A Topology of Kerberos Federations

Bidirectional interrealm trust in Kerberos requires a pairwise of key exchanges. Applying this arbitrarily to a set of n realms yields $O(n^2)$ key exchanges. This topology can be modeled by a directed-complete graph in which the nodes represent the realms and the edges represent key exchanges, as shown in Figure 3.6 for five realms.

To alleviate the problem of having to deal with a large number of key exchanges, a Kerberos Version 5 specification recommends organizing the realms in a hierarchical structure. Key exchanges across ticket-granting servers from various realms are performed only along this hierarchy structure. Specifically, key exchanges take place across realms that are directly descending or ascending from one another. Exceptions to this rule are referred to as *shortcuts* where two realms unrelated by the hierarchy relationship are directly joined via a key exchange to optimize heavily used paths. A hierarchy defined along domain names of the participating realms is a natural fit. The number of key exchanges required by this topology is $O(\log(n))$. Figure 3.7 illustrates the hierarchical interrealm trust in Kerberos. The dotted edge represents a shortcut.

When an application needs to send requests to a server in a foreign realm, it traverses the tree upward, downward, or through shortcuts until the destination realm is reached. In each step of this traversal, a TGT is acquired for the next foreign TGS.

Ticket Forwarding

Kerberos supports authentication forwarding, also referred to as *delegation in the form of impersonation*. Here an entity that has authenticated to the KDC

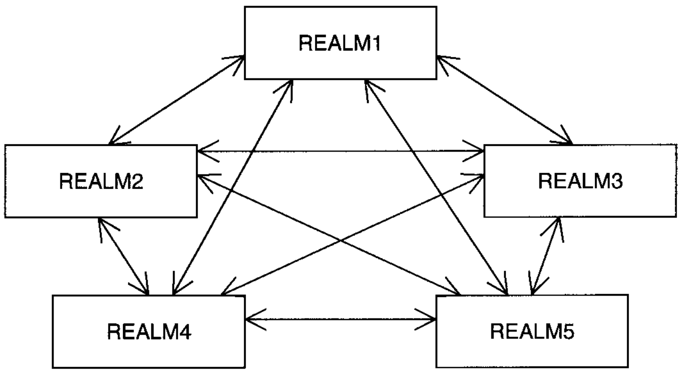


FIGURE 3.6 A pairwise key exchange across five realms modeled using a complete graph

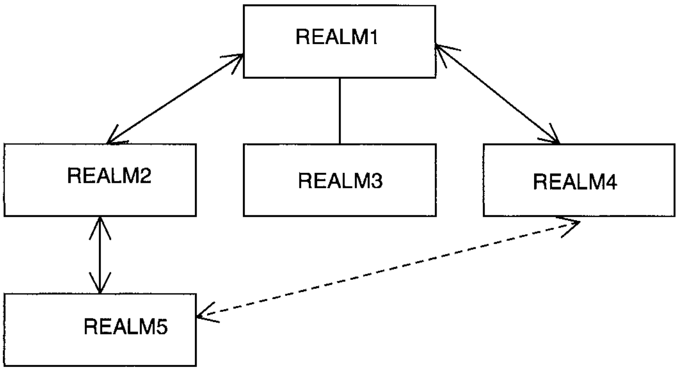


FIGURE 3.7 Cross-realm hierarchical key exchange

(i.e., holds a valid TGT) delegates its authenticated context to another entity on a local or remote host. Thereafter, the delegated entity impersonates the original entity and may acquire tickets to downstream servers on its behalf. An example where delegating credentials is useful is the case of a server that needs to access a file stored on a network file system that is accessible by the client only. Such may be the case of a print server, for instance.

Delegation in Kerberos is enabled by way of the client forwarding its TGT to a server. During the initial TGT acquisition, the client requests that the ticket be marked forwardable. The session key established between the client and the TGS is also forwarded to the target server so that it can form a fresh authenticator as it attempts to acquire a service ticket from the TGS.

Entitlement Attributes in Kerberos

In addition to serving the purpose of authenticating clients to target services, a Kerberos ticket may contain a set of authorization privileges that are associated with the holder of the ticket. The following definition expressed in Abstract Syntax Notation 1 (ASN.1) illustrates the structure of authorization information contained in a Kerberos ticket.

```
Ticket :: = [APPLICATION 1] SEQUENCE {
                              tkt-vno[0]        INTEGER,
                              realm[1]          Realm,
                              sname[2]          PrincipalName,
                              enc-part[3]       EncryptedData
}
EncTicketPart :: =  [APPLICATION 3] SEQUENCE {
                flags[0]          TicketFlags,
                key[1]            EncryptionKey,
                crealm[2]         Realm,
                cname[3]            PrincipalName,
                transited[4]        TransitedEncoding,
                authtime[5]         KerberosTime,
                starttime[6]        KerberosTime OPTIONAL,
                endtime[7]          KerberosTime,
                renew-till[8]       KerberosTime OPTIONAL,
                caddr[9]            HostAddresses OPTIONAL,
                authorization-data[10]
                               AuthorizationData OPTIONAL
}
AuthorizationData ::= SEQUENCE OF SEQUENCE {
                ad-type[0]          INTEGER,
                ad-data[1]          OCTET STRING
}
```

Authorization information is marshaled in a Kerberos ticket as a sequence of (*ad-type, ad-data*) value pairs with ad-type representing the parameterization factor. This parameter is an integer that classifies the value of the authorization attribute with which it is associated. Negative values are reserved for local use. Nonnegative values are reserved for registered use (i.e., one that is known to the Kerberos community at large). The fact that the data type of an authorization attribute is a stream of octets allows it to be extensible and dynamic.

Cross-realm support in Kerberos enables the federated management of user entitlements over widely distributed computing resources. Principal entitlements are maintained by the Kerberos service associated with the realm in which the target service resides. This is expressed by the fact that a principal obtains a service ticket directly from the TGS of the target service's realm. Authorization privileges and user-profile attributes fit well with the local management paradigm in which access control is performed by the local resource managers. In this approach, the semantics of entitlement attributes are locally scoped, and thus ambiguity and collision among attribute names are prevented. The security model enabled by Kerberos therefore follows the paradigm of global authentication and local management of authorization. The latter encompasses the semantics of access privileges and provides resource-access control. Adherence to this paradigm is an important aspect of identity and trust management in highly distributed computing models.

A Kerberos service ticket carries information about the home realm of its holder in the crealm field. This field indicates the name of the realm in which the client is registered (i.e., with which the client explicitly authenticates). Resource managers that receive service tickets from principals in foreign realms can further qualify the semantics of the access privileges and entitlements by the foreign realm. This adds another parameterization factor that can be used to scope or distinguish among entitlements for local versus foreign principals. For instance, attribute A from a foreign user's profile may require more stringent trust-verification procedures than when that same attribute is associated with a principal that is local to the realm of a service.

A Kerberos identity is always qualified with the name of the realm in which it is defined. Even when two principal names from different realms are identical, they differ when qualified by the respective realms. Principal name collisions across realms are therefore eliminated. The partitioning of Kerberos naming space along realms plays an important role in the federated trust of Kerberos. This information is reliably and securely carried in the encrypted portion of a Kerberos ticket.

Explicit Third-Party Authentication Paradigm

The third-party authentication method via entity introductions is a novel approach that advanced the state of art in the field of authentication, particularly with the development of Kerberos. A number of aspects, however, characterize this model with some level of rigidity. For one thing, it requires all participating entities to adhere to a predefined authentication protocol. Programmers need to abide by a relatively advanced programming model, and the protocol has a degree of infrastructure complexity built into it. The predominant alternate approach is a much simpler one, easy to use but of lesser strength and eloquence. This approach uses an explicit authentication scheme in which the authenticating entity does not manage its own user registry; instead, it calls out to a third-party service or subsystem.

The explicit paradigm of third-party authentication is based on the principle of outsourcing the authentication process within a distributed environment to a third party that manages an identity repository, performs authentication, and dispenses entity entitlements. Typically, an application server directly receives an authentication credential such as an identity and a password from a requesting client. The credentials are then forwarded to the third party for authentication as well as the retrieval of entitlements. Various forms of third-party entities have been used for this purpose. An example is a database system against which a user credential is validated (e.g., by attempting to connect to a database using the user's credential). A widely used third-party registry is the hierarchical X.500 directory service exposed through the LDAP protocol [HOWE03, WAHL97, HOWE95]. Here an identity is established by way of a successful bind operation to the directory using the credential supplied by the client.

This trust model is characterized by being loosely coupled in that the interacting entities are not required to participate in a well-defined protocol sequence. The client communicates with the target service using application-level interfaces. Similarly, the server engages the third-party entity using interfaces specific to that third party. The target-application service, in particular, needs to secure the communication channel used for the transmission of credentials between the client and the application, on one hand, and the application and the third party, on the other hand. Typically, a secure socket-layer (SSL) [FREI96] channel is used for that purpose. This model offers the advantages of simplicity and extensibility. Connectors to various third-party identity services can be incrementally built and used.

Plugging an application server with a third-party identity and trust manager in this fashion is exploited by a number of evolving Web application servers (WAS) such as IBM's Websphere [IBMC03]. Websphere further generalizes this approach by abstracting the third-party authentication services and repositories in what is referred to as a *pluggable authentication mechanism*. This can be represented by an LDAP service or some native operating system repository such as IBM's RACF or one that is customized. Figure 3.8 illustrates the third-party explicit authentication paradigm.

The Public-Key Infrastructure Approach to Trust Establishment

Public-key cryptography was developed with a revolutionary concept—that of establishing trust without having to share secrets. The premise of freely disseminating a public key, however, remains a proposition that nevertheless comes with cost, as well, perhaps only less than that of distributing secret keys. Security services, particularly origin authenticity, rely on the single foundation that a particular public-key material is indeed bound to its legitimate user. The public-key establishment problem relates to trust in the binding that exists between a subject and a public key. The novel paradigm brought about by public-key encryption relies on the fact that public keys are intended to be universally accessible. As long as the binding of a public key can be securely established, the key material can be distributed over secure and nonsecure channels and stored in public repositories. An established public key is one that exhibits the property of being securely and unambiguously associated

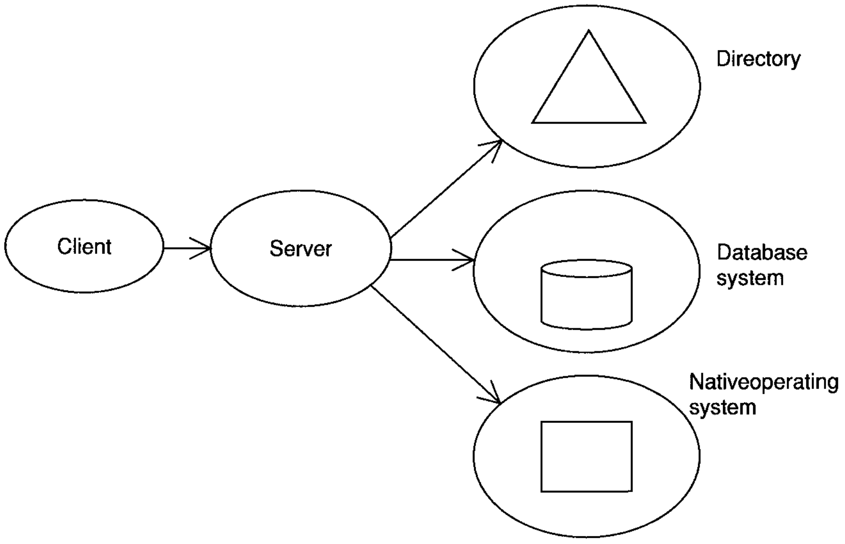


FIGURE 3.8 Layout of an explicit third-party authentication scheme

with its legitimate owner. This association should remain invariable no matter the transport over which the key is being communicated or the storage medium in which it resides or an execution runtime where it is processed.

In the Internet world, public-key establishment is defined through the X.509 digital certification performed by a trusted third party known as the *certificate authority* (CA) [BENA02]. The result of this certification process is a data construct in the form of an X.509 certificate representing a cryptographic binding between the public key material and its holding entity referred to as a *subject*. The foundation of such certification rests on the digital signature of the authoritative CA vouching for the trustworthiness of the certified public key and hence the associated private key. We begin by taking a brief overview of public-key cryptography, pointing out its underlying strength in representing trust. An instance of that is expressed by the capability of public-key cryptography in realizing digital signatures. We subsequently elaborate on the trust elements that form the foundation for the Internet public-key trust.

Foundations of Public Key-Cryptography

Public-key cryptography emerged in the mid-1970s with the work published by Whitfield Diffie and Martin Hellman [DIFF76a, DIFF76b] as well as by Ralph Merkle [MERK78]. The concept is simple and eloquent yet it has had far-reaching impacts on the science of cryptography and its applications as a whole. 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. Data encrypted using the public key can be decrypted only using the associated private key and vice versa.

In the following, we consider a simple example that illustrates the dual key concept of public-key cryptographic systems. We restrict our plaintext to 27 characters drawn from the 26-letter English alphabet plus the blank character. We then assign numerical equivalents to our plaintext alphabet sequentially from the integral domain of [0...26] with the blank assigned the numerical 26. We consider our encryption function E to be the affine transformation that takes in a plaintext character P and maps it into a ciphertext C as follows:

$$E(P) = (a * P + b) \bmod 27 = C,$$

with a and b being fixed integers. Solving for P in terms of C in the prior equation yields the inverse transformation, decryption D:

$$D(C) = (a'* C + b') \bmod 27 = C, \text{ where}$$

$$a' = a^{-1} \bmod 27, \text{ and}$$

$$b' = -a^{-1} * b.$$

For a to be invertible while computing in $\boldsymbol{Z/27Z}$, it is necessary and sufficient to have a and 27 relatively prime. That is to say, there is no number that divides both a and 27 but for the trivial divisor of 1. Note that this condition guarantees a one-to-one mapping between P and C. $\mathbf{Z}/27\mathbf{Z}$ is the set of equivalence classes (residue classes) with respect to the relationship of congruence *modulo* 27.

The parameterized affine transformation in the example, and its inverse can be used for a basic public-key cryptosystem with the private and public keys being (a, b) and (a', b'), respectively. An example would be to have $a = 2$ and $b = 1$, resulting in $(a', b') = (14, -14)$. The premise here is for an entity to maintain secrecy of the private key while freely distributing the public key. An encryption performed using the public key can be decrypted only using the corresponding private key. Since the owner of a public-key pair is presumed to be the sole entity with knowledge of the private key, encrypting information using the private key leads to establishing data-origin authenticity. Furthermore, with tamper-proof storage and manipulation of private keys, nonrepudiation can be established as well. Besides the provision for data integrity and confidentiality, public-key encryption is about establishing authenticity without having to disseminate or manage secrets.

In practice, however, the public-key cryptographic system in our example is easily defeated, even with its generalization to longer blocks instead of single characters. A block of size s yields a ciphering transformation that maps each block to a value in the range $[0...N^s - 1]$, where N is the size of the alphabet. The weakness of this algorithm rests in the ease by which a decryption key can be deduced from an encryption key in a deterministic fashion, using very simple operations (multiplication and additions *modulo* $(N^s - 1)$). But first and foremost is the fact that the encryption function admits a deterministic inverse function.

The premise behind public-key cryptography is that it should be computationally infeasible to compute the private key by simply knowing the public key. Along this key premise, we discuss some of the mathematical foundations of the processes by which modern public-key cryptosystems derive their strength and reliability when it comes to the generation of public and private key pairs. Figure 3.9 is an illustration of the duality between corresponding public and private keys.

Modern public-key cryptography derives from eloquent mathematical foundations that are based on the one-way trapdoor functions existing in the abstractions of number theory. Encryption is the easy one-way trapdoor. Decryption is the hard direction. Only with knowledge of the trapdoor (the private key) can decryption be as easy as encryption. Three of these currently known trapdoor one-way functions form the basis of modern public-key cryptography, and we discuss them in the next sections.

The Problem of Factoring Large Numbers

The first of the well-known trapdoor one-way functions is based on the ease of multiplying two large prime numbers, while the reverse, factoring a very

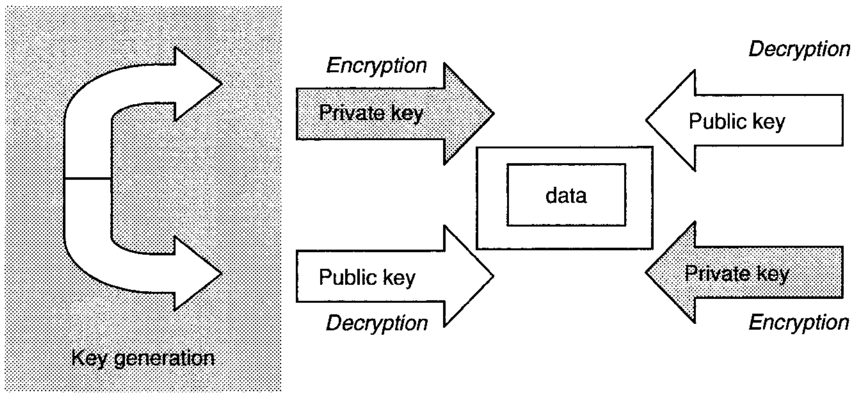


FIGURE 3.9 The duality between public and private keys in public key cryptosystems

large number is a far more complex task. Factoring an integer n is the process of finding a series of prime factors, such that their products together yields n. A prime number, by definition, is one that has no divisors other than 1 and itself; otherwise, a number is called *composite*. Factoring large numbers (over 1,024 bits) is known to be computationally infeasible with today's computers and technology. Modular arithmetic renders the multiplication of such numbers a far easier task. Consequently, the one-way trapdoor problem here is to make a very large number a public knowledge and secretly maintain its prime factors. Note that the trapdoor function discussed here in essence requires deciding on whether a randomly picked very large number is prime. Primality testing is a much easier task than the actual factorization [GORD85].

A number of methods have been devised to determine the primality of an odd number N. The most trivial of which is to run through the odd numbers starting with 3 and determine if any of such numbers divides N. The process should terminate when we reach $\sqrt{N}$. Due to the time complexity that this method requires, in practice it is stopped much earlier before reaching $\sqrt{N}$ and is used as a first step in a series of more complicated primality test methods.

The best example of this class of public-key cryptosystems is the Rivest-Shamir-Adleman public-key algorithm, known by its acronyms of RSA [RIVE78].

Computing Discrete Logarithms in a Large Finite Field

The second well-known trapdoor one-way function that exists 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 a much harder problem. 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 a of GF(p) is guaranteed to have a multiplicative inverse a^{-1} that is also in GF(p), such that

$$a * a^{-1} = 1 \bmod p.$$

The time complexity required for the computation of $f(x) = a^x = y$ in $\boldsymbol{Z/pZ}$ 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 $\boldsymbol{Z/pZ}$ as opposed to the much easier continuous problem in the set of real numbers, for instance (hence the use of the term discrete in qualifying this problem).

The one-way trapdoor function as defined by the discrete logarithm problem can be stated as follows:

Knowing a and x, it is an easy operation to compute a^x in $\boldsymbol{Z/pZ}$ (using the repeated-squaring method). On the other hand, if we keep x secret and hand someone the value y that we know is of the form a^x and ask to determine the power of a that gives y, they can use up all the computing resources that they have available but will indefinitely fail to hand back a response.

A number of modern public-key cryptographic algorithms are based on the discrete logarithm one-way trapdoor function. Most notable is the Diffie-Hellman key exchange algorithm [DIFF76b] and the El Gamal cryptographic system [ELGA95].

Elliptic Curves over Finite Fields

Elliptic curves over finite fields have been proposed for use with existing public-key cryptographic systems [KOBL87, MILL86]. Given a point P from an elliptic curve E, defined over a finite field, and an integer a, the one-way function here consists of the ease of computing the product a^*P, while the inverse of finding a such that a^*P results in a point over E is intractable. Elliptic curves as such form a reliable and secure source for computing public keys. The elliptic-curve analogs of existing algorithms that are based on the discrete log problem, such as Diffie-Hellman and ElGamal, can be deduced in a straightforward manner. The discrete log problem on elliptic curves is likely to be harder to tract than its counterpart on finite fields. This property has led to the adoption of elliptic cryptosystems in many situations requiring stringent security measures.

Digital Signatures

The advent of public-key cryptography combined with the strength and reliability of intractable one-way hash functions gave rise to the digital signing of a document. This process inherently enables data-origin authenticity and can be strengthened to further withstand repudiation. Using the private key of a public-key pair to encrypt a data stream automatically binds the subject with whom the key is associated to the data. The cost of encrypting an entire document to simply establish this binding can be prohibitive, particularly in light of the compute-intensive public-key cryptosystems. Fortunately, the alternative is eloquent and is computationally affordable as it does not require encrypting an entire document. Two of the well-known digital signature algorithms are the RSA and the DSA [NIST94]. We briefly outline the RSA algorithm below.

RSA Signature

The RSA digital signature algorithm proceeds along two main steps:

- ❑ Using one of the common hashing algorithms such as MD5 or SHA-1 [RIVE92, [NIST95], a document is first digested into a much smaller representation, a hash value.
- ❑ Encryption is applied to the hash instead of an entire document

Provided there is no need for a confidentiality service, the signed document is then transmitted in its cleartext form, and the signature is provided to the recipient for verification. Figure 3.10 illustrates the RSA signature computation and verification procedures.

Trusting a Public Key

From the outset, public-key cryptography seems to eloquently solve the key distribution and management problem introduced by secret key cryptography.

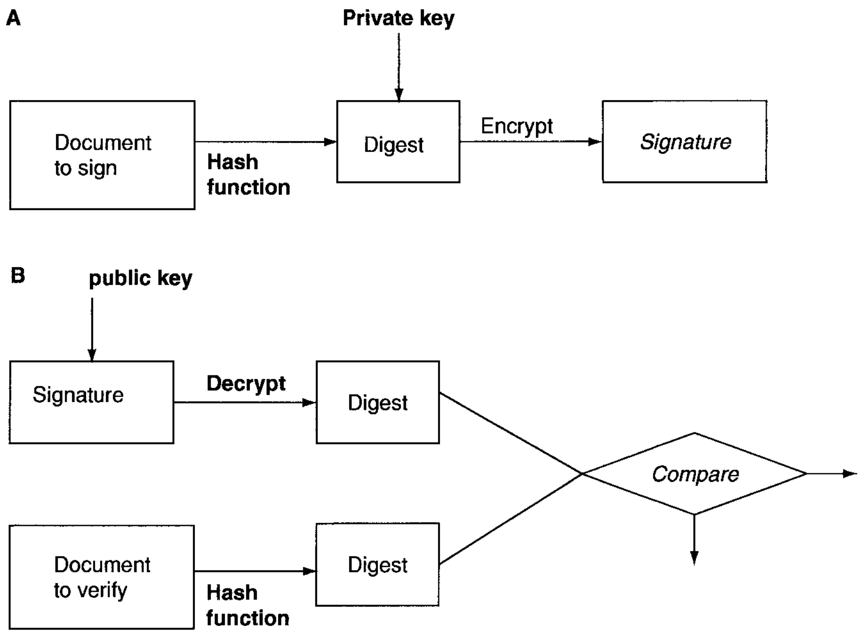


FIGURE 3.10 A Generating a RSA signature and B verifying the signature

Anyone can use the public key to encrypt data, but only the owner of the private key can decrypt it. A community of users that wishes to communicate in confidentiality can adopt a public-key cryptosystems, publish the public keys of its community members in a directory, and completely dispel any concerns that may otherwise arise when distributing secret keys. Unfortunately, the secure binding of a public key to its legitimate holder remains a critical problem on which trust is completely dependent. In a sense, the authenticity of a public key with respect to its holder is at issue.

One promising answer to the question of assurance in a public key 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 outlined earlier. Parties relying on public keys confine their trust in a single entity, known as the *certifying authority* (CA). Before a user's public key is disseminated, the underlying high-assurance CA uses its own private key to digitally sign the user's key, which is then distributed to a public repository. The concept of a verifiable public-key certification can be traced back to the work published in [KOHN78].

A relying party securely installs the public key of the trusted CA and uses it to verify the signature of each user's public key that might thereafter be used. Only on a successful verification does the reliant party initiate a communications channel. This simple method of certification thwarts against an attacker who does not have a public key signed by the same CA as that of the two communicating parties but fails when the attacker is in possession of a key signed by the same CA.

To yield a reliable assurance, a comprehensive public-key certification process necessitates more security elements than simply signing an encryption key. These elements are embodied in the data construct that is to be certified. For the Internet realm this construct is called an *X.509 Version 3* certificate, and the secure infrastructure that makes it is the public-key infrastructure for X.509 (PKIX) [HOUS99a, HOUS99b]. We discuss the main PKI trust elements in the next section.

Foundations of Trust in PKI

An Internet public-key certificate (PKC) provides a high degree of assurance in the public key that it certifies. At the core of this assurance is a trusted issuing authority that is either the signer of the PKC or one situated along a chain of certificates leading to that PKC. Such a chain is called a *trust path*; its meaning will become clear in the next sections. The trust provided by PKI is demonstrated by a provable binding between the public-key material and its associated subject and hence the private key. Recall that the public and private keys are mathematically related values that are associated with one another. In addition to the public-private key pair, the certified binding implicates a set of attributes that a subject may possess. Such attribute may include an X.500 *distinguished name* (DN), an electronic mail address, or further yet

{-- the signed portion

Version number	**v3**
Serial number	xxxxxxxxxxx
Signature algorithm	xxxxxxxxxxxx
Issuer name	xxxxxxxxxxxxxx
Validity period	xxxxxxxxxx
Subject name	xxxxxxxxxxxxxx
Subject public key	xxxxxxxxxxxxxxxx
Issuer unique identifier	xxxxxxxxxxxxx
Subject unique identifier	xxxxxxxxxxxxxxx
Extensions	xxxxxxxxxxxxx

}

Signature algorithm	xxxxxxxxxxxx
Signature value	xxxxxxxxxxxxxxx

FIGURE 3.11 Data elements of the X.509 v3 certificate

a customized personal attribute profiling the certificate holder. Figure 3.11 illustrates the major elements that are implicated in a certified public key using X.509 V3 certificates.

The trust model in PKI is anchored through the degree of assurance in the public-key certificate of the issuing CA. The public key of the issuing CA as determined from its own PKC is, in turn, used to verify the digital signature of that CA in the user's PKC. That signature is computed over the data elements of the certificate as illustrated in the bottom part of Figure 3.11 including, of course, the public key material. Given the assurance in the PKC of the issuing CA, a successful verification of this signature establishes trust in the binding of the public key being verified and hence the corresponding private key to the end entity that holds the PKC.

The need for the secure verification of an end entity's public key is likely due to the involvement of that entity in a public-key-based security protocol or simply in data signing or encryption. Besides the signature verification step, establishing trust in a PKC is foremost based on the certificate itself being valid. Two key factors are decisive in determining the validity of a certificate:

❑ *Revocation of the certificate* First the certificate is checked for membership in a *certificate revocation list* (CRL). A revoked certificate is invalid regardless of its signature being valid. A PKC may be revoked before at any time before expiry arrives. Various revocation policies may be instituted based on circumstances. A CRL is the second major data construct that is available for PKI consuming entities. It attests that the PKCs to which it refers are no longer valid for use. Like for PKCs, CRLs are constructs that are digitally signed by certificate authorities. Below we shed more light on the links between a PKC and its entry in a CRL.

- *Time of use* The certificate use has to be valid with respect to its designated lifetime as indicated in the PKC itself.

The elements that contribute to the validity or invalidity of a PKC are all included in data over which the PKC digital signature is computed. A number of aspects can affect the level of trust in a PKI. Below we discuss two such aspects. The first is the serial number embedded in a PKC and its relation to a CRL. Subsequently, we shed light on the element that is without a doubt the cornerstone of trust in PKI—that of protecting the private key of a certificate signing authority.

Identification Links Between a Certificate and a CRL

As it is shown in Figure 3.12, the certificate serial number is about the only field that identifies a certificate membership in the list of revoked certificates contained by a particular CRL. A collision in certificate serial numbers therefore may lead to erroneous decisions by validating entities. Since it is only within the confines of a particular certificate authority that the serial-number-generation process can be controlled, it becomes an implicit requirement that a certificate be revoked by the same authority that had issued it. Furthermore, assuming that the serial numbers are generated in some incremental fashion, the serial-number-generation functions need to maintain a persistent representation of the current number over the lifespan of the authority. Due to the importance of using a unique number for each certificate, the persistent form of the current serial number may need to be encrypted while it is saved in auxiliary storage.

Certificate membership in a CRL needs to be decided by the identification parameters as represented by both the serial number as well as the issuer name.

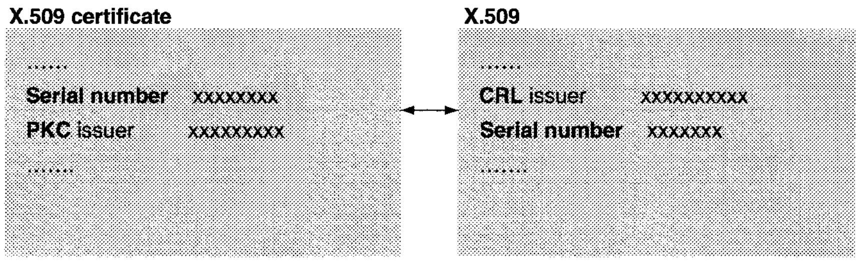


FIGURE 3.12 Identification links between a certificate and a CRL

Protecting the CA Signing Key

The CA private key deserves being the object in need of most protection possible within a public-key infrastructure. After all, the verification of assurance in the certification process is entirely dependent on the security of this key. Indeed, once a CA signing key is compromised, the whole infrastructure and any relying entities and applications are breached. A compromised CA key can lead to all sorts of attacks. Issued and published certificates can be modified. Others can be illegitimately revoked. Most dangerous is that certificates can be issued under the auspices of the compromised CA to subjects that are not entitled to certificates. It is prudent measure to treat the CA signing key with particular care. Software solutions can provide an increasing degree of security to the signing key through encryption. However, because the key must be exposed to generate signatures, it may become vulnerable to interception and capture.

One approach that affords the CA key a high level of security is the use of tamper-resistant hardware in the form of PCI-based cards to store cryptographic keys and perform encryption and signing operation without exposing the key. One reliable product in this category is the IBM 4758 coprocessor card that is delivered with a high level of assurance and manufacturing certification. This cryptographic coprocessor provides a simple access interface using the IBM Common Cryptographic Architecture (CCA) APIs as well as the RSA Laboratories PKCS #11 interfaces (cryptoki) [RSA99]. It relies on a key-encrypting key, the master key, stored in a tamper-resistant circuitry that withstands physical attacks.

The IBM 4758 provides a whole set of cryptographic operations such as random number and key generation, hashing, encryption, generating message-authentication codes (MACs) as well as signing and verifying signatures. These operations are based on common cryptographic algorithms such as SHA-1, MD5, DES, Triple-DES (DES3), RSA, and DSA. In addition to the cryptographic hardware engine, the card includes a small general-purpose processor. The access-control module serves as an authentication mechanism used to log on users to the coprocessor as well as performing access-authorization checks based on the different roles a user might assume. Enforcing access policies as such is achieved by the hardware and protected software. The coprocessor manages DES and public-key algorithm (PKA) keys separately.

PKI Trust Topologies

Trust verification in PKI may involve more than one CA certificate. Depending on the trust topology in use, the validation process can become a recursive process involving a chain of CA certificates. We outline the trust topologies commonly found in PKI in the sections below.

Hierarchical Trust

A hierarchical topology is one that maps the trust layout of an organization top down into a tree structure [HOUS99a]. At the top of the tree is the root certificate authority. Extending branches may lead to leaf nodes that represent end entities in the organization or may lead to other subauthorities. The rational for the partitioning may stem from the need to manage a large organization as a set of smaller entities, each with its own authoritative CA. Figure 3.13 shows an example of a hierarchy structure. Generally, there is no requirement that one CA certify end entities only or other CAs only. A particular CA may issue certificates to end entities as well as to other certificate authorities. But for all practical purposes, however, the role of each CA may be best managed by requiring that it certify subordinate CAs only or end entities only. Such a separation enforces the authoritative hierarchy structure of an enterprise and points out the controlling elements of trust.

The hierarchical trust topology enables the delegation of trust down to subordinate authorities. The root, high-trust authority becomes concerned with the trust-delegation task down to a smaller number of subordinate authorities. The fact that the top CA is concerned with the dissemination of trust to a small number of entities allows for managing the strict controls and policies that need to apply at this highest level. One such policy may require the offline distribution of the root CA certificate in a highly secured fashion to the immediate subordinate CAs that it manages. There is a fundamental reason behind the secure distribution of the top certificate; the process of building a trust chain begins at the root CA.

Building a trust chain consists of backtracking the path from an end entity certificate all the way to the root-trusted CA. This backtracking process

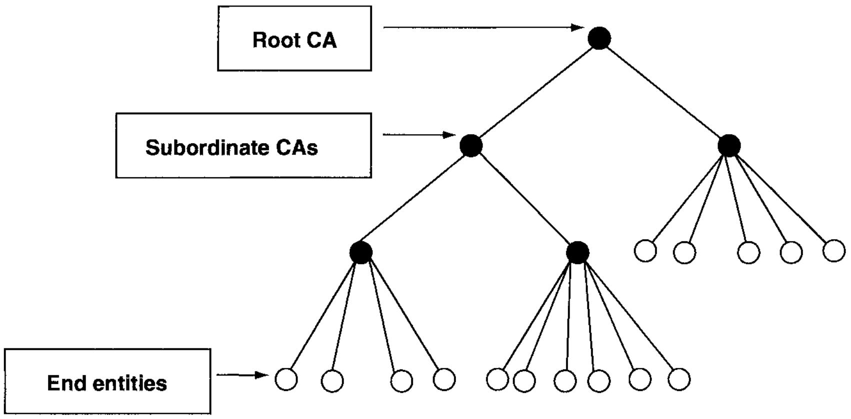


FIGURE 3.13 A hierarchical trust topology with one root governing a two- and a three-level hierarchy

entails a number of validation steps, two of which are fundamental. The first is the determination of the chain by starting at the leaf end-entity certificate, associating an issuer name at this level with a subject name in a certificate of an authority at the immediate upper level until the root is reached. Figure 3.14 depicts this process of computing a trust path. For each subject name determined as such, the corresponding CA certificate is retrieved, perhaps from a repository such as a directory service or one referred to through some URI.

The second step consists of validating the series of cryptographic signatures in the previously computed trust chain. This process begins with the certificate of the root trusted CA and proceeds until it reaches the leaf end-entity certificate.

As illustrated in Figure 3.14, the determination of the path via the back-tracking of issuer and subject names is computed in a bottom-up fashion starting with the end-entity certificate. By contrast, the signature-validation process is performed in a top-down fashion beginning with the certificate of the trusted authority.

Signature validation is the process during which the fundamental trust of a certificate is built. It is all based on the basic assumption that the public key of the root CA is trusted. Recall that assurance in this assumption is based on the secure distribution of the root CA certificate. This distribution process defines what can be termed as the "boot-strap" of trust.

The high-assurance public key of the root is used to validate the signature value in the CA certificate immediately below it in the hierarchy as determined by the path. Once this is validated, the immediate subordinate CA

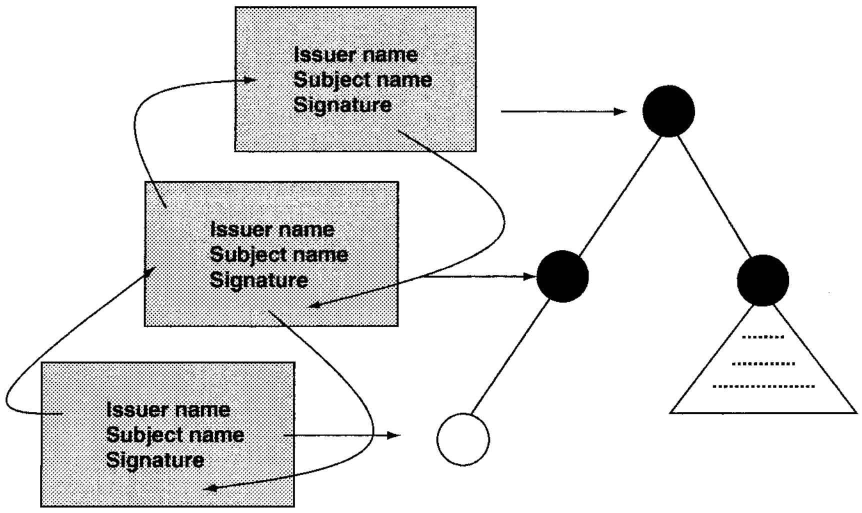


FIGURE 3.14 Computing a trust path in a hierarchical trust model

implicitly inherits the highly assured trust property and becomes the trust root. This procedure continues recursively until the signatures in the leaf end-entity certificate is validated. A special case of this path-validation scheme is one in which there is only one level of hierarchy, and thus the self-signed root CA certificate is used to directly validate the signature in the end-entity PKC.

The fundamental element of trust in a certificate chain rests in the secure distribution of the root CA certificate to all of the entities below it in the hierarchy. The dissemination of the root CA certificate may involve an offline distribution method to increase security. For instance, the certificate can be mailed to the respective human entities in a nonvolatile medium such as a diskette or a compact disk. On receipt, each entity computes a digest of the certificate using, for example, SHA-1 or MD5 and then calls the human trusted with the administration of the CA to confirm the digest value and hence this distribution process.

The notion of a single point of trust does not necessarily concern the root CA only. Rather, it can be applied down the tree hierarchy in a delegated fashion. The property that makes this delegation stand is that the recursive signature-validation scheme, as described, can also be started at some highly trusted intermediate CA. Any compromise in the signing keys above this intermediate CA will ultimately be detected once validation reaches the trusted intermediate CA. The trust path therefore requires the existence of at least one high-assurance authority along the path irrespective of its position in the tree hierarchy. A delegation scheme of this kind lends itself well to situations in which end users of some global enterprise need only to be aware of "regional" certificate authorities that directly manage their part of the business but need to be concerned with the corporate CA.

The advantage of setting up a multilevel trust hierarchy is to bridge multiple organizations (public-key infrastructures within, say, a large organization) without having to reissue the public-key credentials already deployed within each of the individual organizations. Let us assume that an enterprise that has grown due to a merger decides to join its existing and distinct public-key infrastructures into a single hierarchy so that services in one organization can be accessible to the members of the other organization and vice versa.

The hierarchical scheme of trust can provide a solution in this case by having each of the disjointed CAs become subordinate to the root CA, one that is perhaps designated and managed at the corporate level. Figure 3.15 illustrates a hierarchy consisting of two intermediate CAs and joining two different organizations.

The procedural steps required to effect this merge may consist of the following:

- ❑ Have each subordinate CA revoke its existing self-signed certificate and publish it in a certificate revocation list, actually an *authority-revocation list* (ARL). This will ensure that a trust path should always lead to the new root CA.

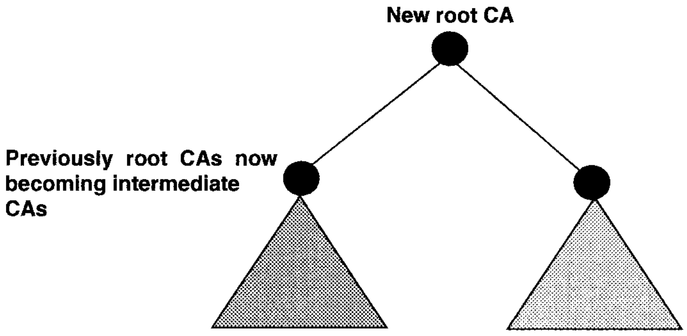


FIGURE 3.15 Joining two organizations using the hierarchical trust model

- ❑ Have each subordinate CA acquire a new certificate from the new root CA. To avoid a CA key-update process, each CA may use its current public key when requesting the new certificate.
- ❑ Distribute the new root CA certificate in a secure fashion to all of the end entities in the merged organizations including the two subordinate Cas, and have each entity replace this certificate for the old trusted root.

The net effect of this join operation is the dissemination of trust across the two previously disparate organizations via the new root CA that represents the trust anchor for the larger organization. Note that if so desired one can split the two organizations by reversing each of the steps in the join operation as described. To accomplish this, first, each CA requests revocation of its own certificate from the root CA. Each subordinate CA then uses its current public key to issue a self-signed certificate for itself and push it down to each of the entities it certifies through a highly assured channel.

Joining existing public-key infrastructures by building a single multilevel hierarchy results in a unified trust model. In this model, a single authority represents trust in the entire organization. Similarly, the affected trust join operation enables the organization to continue delegating to each subordinate CA the PKI management tasks for its own domain of operation.

The use of multilevel hierarchies, however, extends a certificate trust path and thus may affect performance of the certificate validation process. To mitigate the extent of this problem, a PKI deployment as such may resort to computing and then pushing the trust paths to each end entity's local environment ahead of any validation processing.

Cross-Certification

The proliferation of PKIs, particularly in the Internet space, ultimately leads to the need for extending the benefits provided by public-key certification across the boundaries of certification domains. Such domains may span disparate organizations and departments within a single enterprise. In many cases, the requirement for automated interaction across multiple organizations is what drives the need to maintain the benefits of PKI-based security in applications that bring about those interactions. The basic issue here is that of joining independently deployed PKIs with a minimum disruption and a maximum transparency to end users. Most important, in joining disparate PKIs it is sometimes desirable to maintain the independence characteristic that each domain enjoys whereby each certification authority remains the sole authority for its own domain of operations.

Functionally, the hierarchical scheme that we previously discussed can be sufficient for bridging two certification domains, the result of which is tightly linked organizations, virtually becoming a single domain. The drawback of the hierarchical merge is that end entities will not be completely shielded from the join operation. Cross-domain certification, on the other hand, achieves similar trust semantics in joining disparate PKIs, yet it maintains a complete transparency of the process with respect to end entities.

Cross-certification is a method of joining two disparate PKIs without incurring any effect on the end entities and without subordination of either infrastructure to a new authority. It is a peer-to-peer contract between two CAs to honor certificates exchanged, through security protocols, on service requests crossing each other's domain. Each end-entity member in the communities joined via a cross-certification process remains in possession of the certificate of its respective trusted root CA prior to the merge taking place. This is contrary to the hierarchical scheme in which end entities are to acquire the certificate for the new root CA. The trust model remains invariable in the cross-certification case while it takes a different form in the hierarchical scheme.

A CA A that issues a cross-certificate to authority B underscores the fact that end entity certificates issued by B to its own community members are now trusted for use within the domain certified by authority A. Similarly, authority B may issue a cross-certificate for authority A, and thus domains A and B are said to be mutually cross-certified, also referred to as a *two-way cross-certification.* In essence, a two-way cross-certification is equivalent to joining two domains under a single trusted root CA but without a direct impact on end users.

It is worth noting that structurally a cross-certificate is simply an X.509 v3 certificate with a base constraint extension indicating that it is a CA certificate and in which the subject and issuer names represent two different CAs. It certifies the public key of an already operating subject CA as a signing key used for issuing certificates.

Cross-Certification Grid

Given a network of CAs, the cross-certification process can be modeled as a direct graph whose nodes represent the participating CAs while the edges represent the direction of the certification. A directed edge from A to B indicates a one-way cross-certification of authority B by authority *A*. Figure 3.16 illustrates a cross-certification grid comprised of five CAs.

Note that because the cross-certification in one direction is a transitive relationship, CA2 becomes implicitly engaged in a two-way cross-certification with CA5. This is because CA2 is explicitly cross-certified by CA5. Meanwhile, CA2 cross-certifies CA1, which in turn cross-certifies CA3, and hence CA2 indirectly cross-certifies CA3. In turn, CA3 cross-certifies CA5 and thus CA2 implicitly cross-certifies CA5. In that sense, the respective communities of CA2, CA1, CA3, and CA5 are now entitled to interact across the domains represented by these CAs. For a purist, such communities are defined by the strongly connected component in the directed graph representing the cross-certification network of trust [DIES00].

Hub-Based Cross-Certification

Because of the transitivity property exhibited by the cross-certification operation in each direction, a common hublike CA can be used to bridge a network of CAs, thereby establishing a complete cross-certification grid (one in which each CA is cross-certified with each other CA in the network). In this trust topology, every CA is mutually cross-certified with the hub CA only. Trust is then disseminated by way of the transitivity property. Figure 3.17 depicts this topology. Note that the advantage here is that the number of cross-certifications performed in this case is linear in the order n of the number of CAs involved, while in the previous case it is in the order of n^2.

Hybrid Model

The hybrid model is a trust scheme that combines the hierarchical and the cross-certification methods. A multilevel hierarchy can be the result of merging of two organizations, while the cross-certification process might be driven

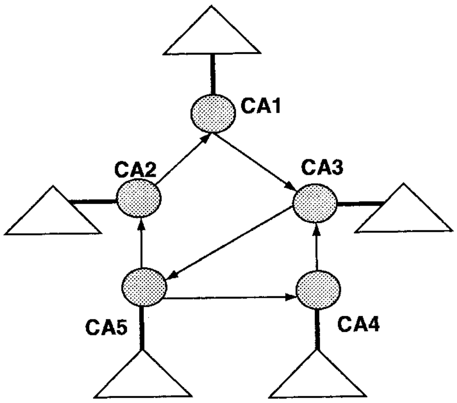


FIGURE 3.16 An example of a cross-certification network

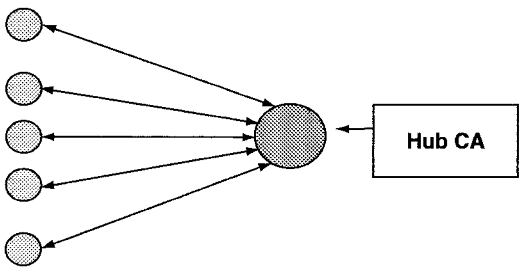


FIGURE 3.17 A network of CAs mutually cross-certified through a hub CA

by the need to extend the trust to a third-party business partner in one direction or another. The complexity of a federation formed by a hybrid configuration may directly affect the performance of constructing a trust path. Implementations may need to optimize path construction by caching constructed paths for subsequent uses. Figure 3.18 shows a trust path between two communicating entities. The path spans two domains in a hybrid scheme of trust.

Web-of-Trust Model

The web model evolved with the advent of the SSL as a security protocol between two HTTP endpoints, mainly the client browser and a target Web server. It uses a more relaxed trust model in which a user can pick and choose

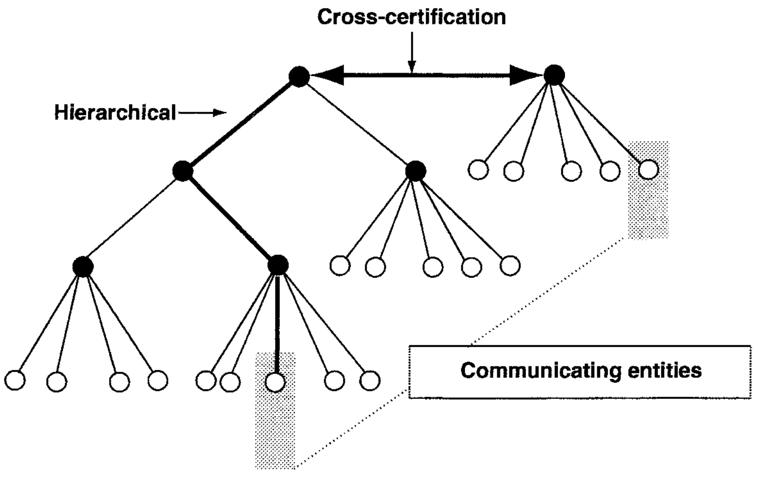


FIGURE 3.18 An example of a hybrid trust scheme bridging two entities

among the trust anchors that he or she deems worthy of being root CAs. An end entity in the web-trust model maintains one or more root CA certificates in its local environment (the browser's key store, for example). Validating a certificate as such consists of finding a trust path to one of the trusted CAs. Generally, these trust paths are shallow and in the most part consist of two certificates, the end entity's and that of the root CA from the local key store. The reason for this is to achieve high performance of the web-based applications. Figure 3.19 illustrates a web-trust model of completely disjointed CAs.

A variant of this trust model is defined by the pretty good privacy (PGP) web of trust. PGP, which evolved into a family of software, was initially developed by Philip Zimmermann as an email encryption program [CALL98]. It uses public key encryption for the distribution of strong secret encryption keys. The trust scheme in PGP known as the PGP web of trust is a simplistic model founded on the discretionary trust of individuals. There is no concept of an authoritative entity that certifies public keys in PGP. An individual user generates a public-private key pair that he or she binds to a unique identifier usually in the form of (name, emailaddress) and is responsible for its distribution to other individual entities or key distribution services. The simplistic information model of PGP certificates is intended for the main purpose of securing email exchanges. Each user maintains a set of public keys of other individuals deemed trustworthy. Furthermore, a key can be signed by a trusting entity and distributed to other individuals. The signing entity is referred as an *introducer*. Trust in the PGP model like in the Internet PKI is not transitive. The fact that A trusts B as an introducer and in turn B trusts C does not necessarily establish that A trusts C. This basic trust scheme has evolved from real-life behaviors. Because PGP has gained popularity mostly as an email encryption tool, its web-of-trust model has naturally

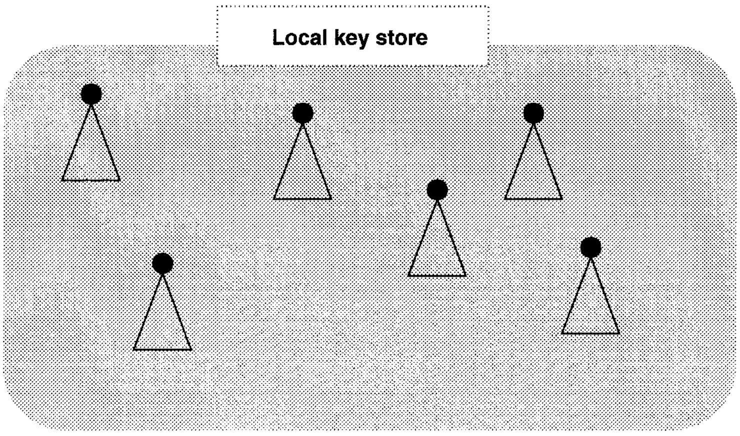


FIGURE 3.19 The web-trust model: Discretional trust of certificate authorities

evolved along a paradigm that mimics trust in human relationships. For this, it is sometimes referred to as a model of the grassroots in which authority is equally distributed across all participating entities.

The PGP web of trust can be modeled by a directed graph $G = (N, E)$ where the set of nodes N represents the collection of entities participating in a PGP web of trust, and edge $e \in E$ from entity A to entity B represents the fact that A trusts the public key of B.

Proxy Certificates: Delegated Impersonation in PKI

Impersonation, the simplest form of delegation, allows an entity A to grant to another entity B the right to establish itself as if it were A. In that process entity B generally inherits a subset of privileges of A. In computational terms entity A may represent an end user, while entity B can be a programming agent running on the user's behalf. Similarly, the initiating entity A can be an identifiable programming agent as well. The use of inherited privileges can be subject to various constraints that may result in what is referred to as restricted impersonation, a benefit of which may be to limit damage from a potential compromise. Impersonation can be recursively applied along a chain of requests, where, for example, a sequence of computing tasks are composed then executed in the course of servicing an end-user request.

Proxy certificates have recently been advanced by the IETF as the mechanism by which chained impersonation can be accomplished in a PKI using X.509 certificates. They were originally introduced by the Globus Project (www.globus.org) as a means for providing single sign-on and delegation in what has come to be known as the *grid security infrastructure* (GSI), a key element of grid computing.

The main motivation behind proxy certificates appears to be the strong requirement imposed in the public-key arena for safeguarding the private key associated with a public-key certificate. Excessive use of the private key increases the probability of exposure and hence compromise. The proxy certificate (PC) concept remedies this problem by allowing an entity that initiates a distributed multitasked request to access its private key only once during initiation. Processes and tasks involved thereafter all impersonate the same initiator yet without having to access its private key.

The Proxy-Certificate Approach

A PC is a public-key certificate that conforms to the X.509 profile [HOUS99a] and has the following properties:

- ❑ The signer (issuer) of a PC is either a holder of an *end-entity certificate* (EEC) or another PC. A PC-holding entity that issues another PC is a participant in an impersonation chain.
- ❑ It contains its own public- and private-key pair, distinct from any other certified key pair.

- It can be used to sign another PC but not an entity certificate (i.e., an EEC).
- A PC certificate chain must have a signing end-entity root certificate, which is a PKC. This underscores the fact that impersonation is controlled by a single delegating entity at the root of the chain.
- An EEC acting as a proxy issuer must have a nonempty subject name.
- A PC does not stand on its own in binding an identity to the certificate.
- A PC inherits its identity from the subject field of a signing end-entity certificate. This may possibly be inherited from the subject alternate name extension of the EEC.
- The subject field of a PC is used as a unique identifier in tracing back the chain of certificates leading up to the original signer. It does not define a new identity by its own.

Typically, a proxy certificate is generated along a delegation chain. An entity B that is authorized to impersonate A generates a public-private key pair, forms a PC and signs it using the private key corresponding to its own PKC. Similarly, a PC that is received by another entity C, during the authentication of a cascaded request, can be used by C to issue another PC, thus further extending the impersonation chain. The entity issuing a PC is called a *proxy issuer* (PI). A PI represents either an end entity or another PC. One key difference between a CA signing a certificate and a PI signing a PC is the fact that the CA performs a unique key to name binding, while the PI does not. Recall that the identity associated with a PC has to be traced back to an EEC. Figure 3.20 illustrates an example of an impersonation chain using proxy certificates.

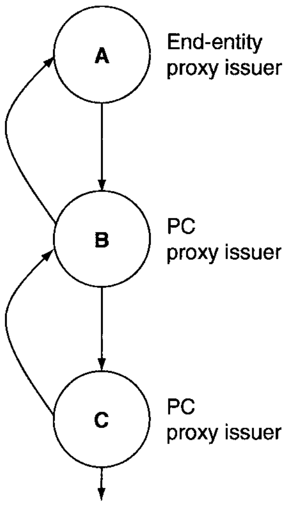


FIGURE 3.20 Proxy certificate chain

Elements of the X.509 Proxy Certificate

A proxy certificate conforms to the X.509 profile. Two elements make this profile dynamic and flexible. The first is the specification of optional fields that may or not be present in a certificate. The second and the most important one is the extensions field intended to be exploited by various PKI-based applications. Besides being simply an X.509 PKC, the characterizing elements of a PC are described below.

- ❑ *The PC extension* The PC profile describes a new X.509 certificate extension designated to identify a PC and to place constraints on its use. This extension, called the ProxyCertInfo, must be present and marked critical in every PC. Its pC field of a Boolean data type must be set to TRUE.
- ❑ *Naming requirements* Because a PC does not represent a name binding of its own, it must not contain the issuerAltName extension. The subject field of a PC must be a sequence of one or more proxy identifiers concatenated together. A proxy identifier is a common name (CN) attribute and should be unique among all PCs issued by one proxy issuer. This characteristic is an important element in tracing back a path of a PC chain when evaluating trust. For example, if the proxy issuer of a PC is an EEC, the subject field must be one single proxy identifier—say, id_1. When that same PC becomes a proxy issuer, the subject field is the concatenation of id_1 and id_2, where id_2 is the unique identifier of the PC (the entity that became a proxy issuer). The proxy identifier value can be the same as the PC serial number. Finally, the subject of PC should be used for path validation only and not for name binding or for use in authorization decision for instance.
- ❑ *Extended key usage* Because a PC inherits the attributes of its issuer, if the issuer certificate includes the extKeyUsage extension, then the PC must include that same extension. The key contained in the PC cannot be used for any purpose for which the issuer certificate is not designated for. Key usage in the PC must be a subset of the issuer's key usage. If the issuer certificate does not contain the extKeyUsage extension, then the PC may or may not include such extension. The criticality of this extension must be preserved top down along a chain of PCs.
- ❑ *Basic constraints* The basic constraints extension that is used to designate a CA certificate must not have the cA field set to TRUE.

Computing Trust in Proxy Certificates

A PC is a representative of some end-user entity with an actual EEC. Ultimately, the binding of a PC to an identity has to involve the root EEC. Validation of a chain of PCs needs to trace back a PC to an EEC. To make the appropriate PCs and the EEC available for path validation, an authentication protocol using a PC may pass the entire PC and EEC chain as part of that protocol.

Computing a PC trust path consists of tracing an issuer name in the PC being validated to a subject name in the issuer's certificate until an EEC is reached. The EEC, in turn, is subjected to the standard trust-path validation that we outlined before to arrive at a trusted root authority CA_r. After the EEC is validated, its subject name can then be used for authorization purposes. Figure 3.21 illustrates the construction of a PC trust path.

In computing a PC trust path, the issuerCertSignature part of the ProxyCertInfo extension found in a PC can be used to add accuracy to the computed path. The optional issuerCertSignature field, when present, can be used during path validation to ensure that each PC path starting with an EEC and ending at the PC is unique. If certificate N+1 in a certificate path is a PC, then issuerCertSignature is used to verify that certificate N is actually the PI that issued it and not some other certificate with the same name and public key. Without this field, if a PI were to issue two different proxy certificates (P_1 and P_2) with the same subject and public key but different proxy restrictions or validity time constraints, then the path-validation algorithm would accept a path in which P_2 appears as the issuer of a certificate that in reality was issued by P_1.

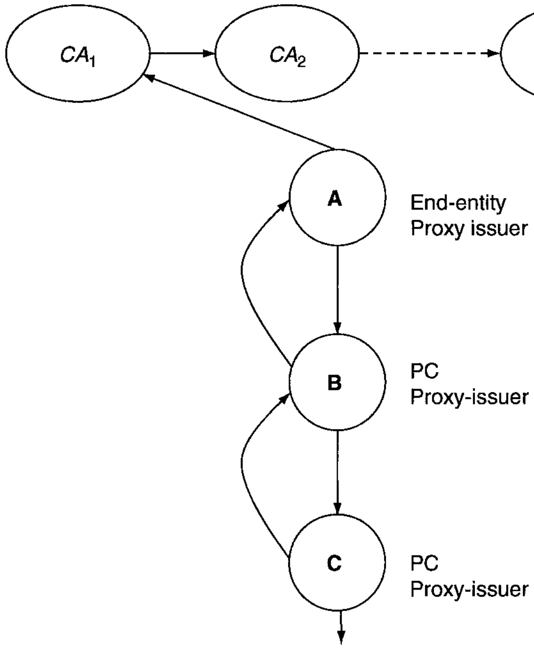


FIGURE 3.21 Constructing a PC trust path

Attribute Certificates: Entitlement Management in PKI

An X.509 PKC is signed and issued by a CA. It binds an identity with a public-private key pair. An *attribute certificate* (AC) is a data construct that is similar to a PKC; it is signed and issued by an attribute authority (AA). The main difference between a PKC and an AC is that an AC contains no public key. Instead, an AC carries with it a set of attributes associated with its holder. These attributes may specify privileges in the form of group membership, roles, a security clearance, or any information profiling its holding user. In essence, an AC binds a user with a set of authorization attributes, capabilities, or in general terms a profile.

Authorization attributes of an entity can be placed in the extensions field of its PKC. The key arguments against this proposition stem first from the fact that certificate extensions are intended for describing certificates and thus expressing user attributes in certificate extensions overloads the semantics of X.509 extensions. The second argument is due to the difference in lifetime between a PKC and an AC. Given that a PKC binds its holder with a public key, its validity period is likely to outlast the lifetime of an AC. User entitlements are much more of a dynamic nature and are constantly subject to change. In contrast, a PKC is likely to remain unchanged and valid for a long period of time. Extending a PKC to include user privileges therefore may increase the cost and complexity of managing the underlying PKI.

Elements of Attribute Certificates

Among pieces of key information contained in an AC is a set of user attributes, a validity period, and a signature certifying the integrity of the AC and establishing the authenticity of its issuing authority. Except for the signature information, all attributes are encapsulated in the AttributeCertificateInfo data type as expressed by the ASN.1 notation of Figure 3.22.

Binding Information

To enable an AC verifier to assert trust, AC binding information defines the association between an AC, its issuer, and its holder. The following data fields represent this binding:

- ❑ *Issuer* The issuer of an AC is represented by its X.500 distinguished name. All AC issuers must have nonempty distinguished names. It is up to the AC verifier to appropriately map the issuer name to a PKC for the issuer before asserting trust.
- ❑ *Holder* In an environment where the AC is passed in an authenticated message or a protocol session in which authentication is based on the use of X.509 PKCs, such as is the case with TLS/SSL, the holder field should contain the holder's PKC serial number and issuer (it asserts the

```
{ -- the signed portion
    AttributeCertificateInfo ::= SEQUENCE {
    Version                     v2,
    Holder                      Holder,
    Issuer                      AttCertIssuer,
    Signature                   AlgorithmIdentifier,
    SerialNumber                Certificate Serial Number,
    AttrCertValidityPeriod      AttCertValidityPeriod,
    Attributes                  SEQUENCE OF Attribute,
    IssuerUniqueID              UniqueIdentifier OPTIONAL,
    Extensions                  Extensions OPTIONAL
}

Signature algorithm         xxxxxxxxxxx
Signature value             xxxxxxxxxxxxxx
```

FIGURE 3.22 Elements of the X.509 v2 attribute certificate

holder in way analogous to establishing its security context). The holder can also be expressed as the subject name or the subject alternate name from its corresponding PKC. This binding leads to establishing an authenticated security context in which the AC can be used to perform authorization checks.

- *Serial number* The serial number assigned to the AC. For any conforming AC, the (issuer, serial number) pair must be unique.

Attribute Information

This field contains a sequence of uniquely identifiable attributes. Each contains a set of key-value pairs. Privilege attributes that are designated for use in access control form the basis of an AC. At least one attribute must be present in an AC. Evidently the absence of attributes altogether defeats the basic purpose of an AC. To foster interoperability across various security domains, a number of AC attributes have been standardized. The following is a brief description of some of them:

- *Service authentication information* This attribute identifies the AC holder to a target service by name. It may also include optional service-specific authentication information. Typical application of this attribute is to communicate the holder's identity and password to a legacy application service. An encryption scheme is likely to be used to provide for the security of the password. The use of the target service's public key to encrypt such information lends itself well for the protection of authentication information. As shown in Figure 3.23, the verifier of an AC, a target service, first establishes the trust path to the holder's PKC. It then uses its private key to decrypt any authentication information. The latter can be passed to a legacy application that is based on such authentication information to establish the identity represented by this attribute.

- *Charging identity* This attribute identifies an identity that can be used by the AC holder for charging purposes. Such attribute can be exploited by a billing service for example.
- *Role* Used to specify a role that the AC holder is capable of assuming. Additionally, it may specify the name of the authority issuer of the role specification as a reference.
- *Clearance* It carries clearance information associated with the AC holder. This attribute can be exploited by systems enforcing multilevel security. The clearance is scoped within an associated policy identifier field in which the semantics of the clearance are defined.

A Note About AC Attributes

The data types used to describe an attribute are designed to provide a high degree of flexibility and extensibility through a parameterization that describes an attribute as a (type, value) pair expressed by the following ASN.1 syntax [BENA02]:

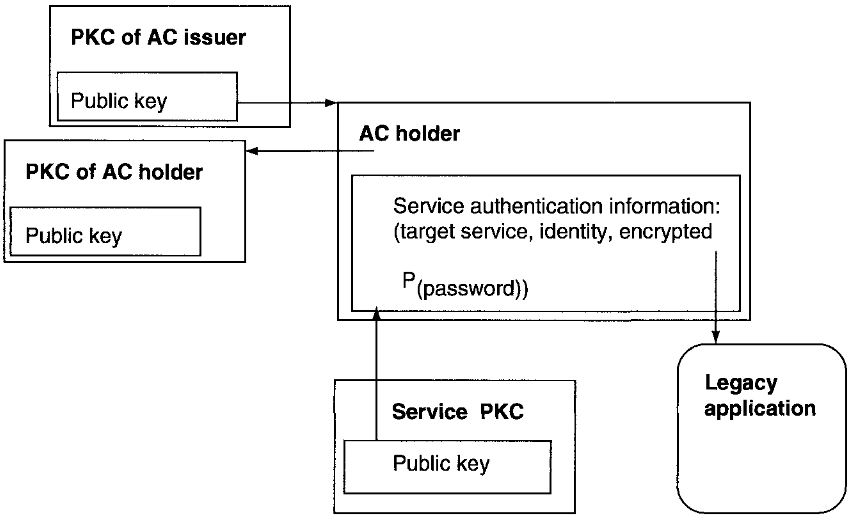


FIGURE 3.23 View of trust verification elements for an AC and its service attributes protected using the PKC of the service

```
Attribute :: = SEQUENCE {
               type      AttributeType,
               values    SET OF AttributeValue
               — at least one value is required
}
AttributeType :: = OBJECT IDENTIFIER
AttributeValue :: = ANY DEFINED BY AttributeType
```

The extensibility of AC attributes is due to the opacity of an attribute's value with respect to the structure of the AC itself. Entities can exploit an attribute embedded in an AC only when they are capable of interpreting both its type and value—of course, provided they are also able to verify any trust elements associated with that attribute. The syntactic and the semantics scope of AC attributes is unbounded and thus can be exploited by various applications.

Extensions

Although most PKC extensions provide information about the certificate itself instead of its holder, some extensions defined for ACs provide a way for associating additional information with holders. Below we enumerate some of the AC extensions relating to identity management and trust:

- ❑ *AC targeting* An AC may be designated for use by a specific target entity. The AC targeting extension is intended for that purpose. Target information may specify multiple services. Relying parties not explicitly named in this extension must reject the AC. This targeting information can be useful in the transactional web. The absence of this extension is an indication that the AC can be used by any relying party.
- ❑ *Audit identity* To satisfy cases where data privacy laws, for example, require that audit trails not reveal or even contain records that identify individuals, an audit identity extension can be added to an AC. This extension allows the logger of an audit trail to use an identity designated by the value of this extension. This value along with the AC issuer name or the AC serial number should be used for audit or logging purposes
- ❑ *Trust-related extensions* By this we mean not one specific extension but a set of AC extensions relating to the evaluation of trust in an AC. These are all defined by the X.509 v3 certificate profile [HOUS99a]. The first is the *authority-key identifier*, which can be used to assist the AC verifier in validating the signature of the AC. The second is the *Authority-information access*, and the third is the *CRL distribution points*. Both of these can be used by a relying party to verify the revocation status of the AC.

Generalized Web-of-Trust Model

The web-of-trust scheme that we discussed under the public-key models can be generalized as a mechanism by which heterogeneous cross-enterprise identity models are joined in a federated web. The building block of this federation is the trust relationship that can be established across heterogeneous identity and trust-management systems using secure network-authentication protocols, some of which we have previously discussed. The trust protocols used can be negotiable between each of two domains entering into a relationship as such. Trust can be one-way or mutual. The potential advantage of this comes from the incremental weaving of trust across domains that builds on existing heterogeneous trust and identity management schemes that may exist in each participating domain. The basic element of trust here relies on the principal of trust by introductions in which entity A that trusts entity B may also trust entities presented to it by B, provided A establishes a trust relationship with B in a secure and verifiable manner.

Federated domains that are based on the generalized web-of-trust model that we propose are characterized by the following:

- ❑ Cross-domain identity-management systems are joined through a negotiated trust mechanism in which an agreed on authentication and trust protocol is used. Authentication is performed between agents of two domains entering in a trust relationship. The direction of trust (one-way or mutual) is based on the policies of the participating domains.
- ❑ Subjects are registered to their, respective, generally local domains. Subject authentication and profile management is performed with its domain of registration only.
- ❑ Subjects authenticate to their respective domain of registration but can seamlessly access services and resources managed by other domains via the trust relationships established across these domains.
- ❑ Identity profile information can be used across domains that have established trust relationships, provided its syntax and semantics are similarly interpreted. Translation of profile information in any direction can be performed by gateways local to each domain.
- ❑ Identity information of a subject remains attached to its original domain of registration as it is passed across domains. The identity of the home domain is attached to this information as it is passed across domains with established trust relationships.
- ❑ Secure transports such as those based on strong cryptographic channels are required for exchanging profile and identity information. These channels depend on the trust scheme adopted between each two domains.

Figure 3.24 illustrates this concept of the generalized web of trust, which can be modeled by a directed graph where the edge directionality represents trust (i.e., edge (x, y) represents trust of y by x). The transfer of profile information for subject s is shown across three domains.

Transitive trust may be used at the discretion of the security policies implemented by each domain. Domain A that enters into a trust relationship with domains B and C may apply the transitive trust policy with domain B

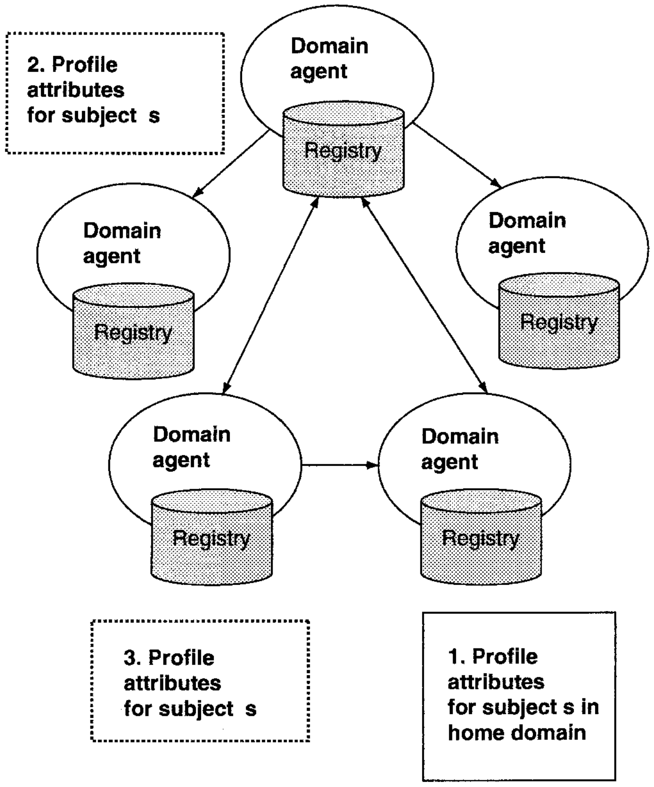
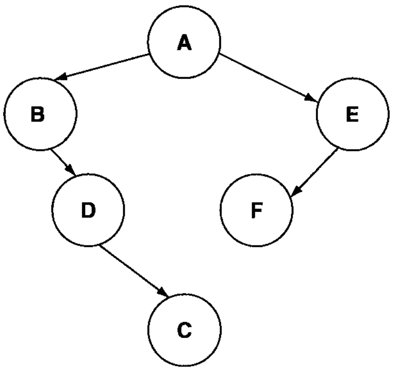

2. Profile attributes for subject s
Domain agent
Registry
Domain agent
Registry
Domain agent
Registry
Domain agent
Registry
Domain agent
Registry
3. Profile attributes for subject s
1. Profile attributes for subject s in home domain

FIGURE 3.24 An example of the generalized web of trust model federating five domains

but not with domain C. Once a trust relationship between domains A and B is designated as transitive, all domains reachable through B for example can be trusted by A. Similarly, the depth of such transitive trust can be limited if so desired. Figure 3.25 illustrates an example of a generalized web-of-trust model in which trust relations are all transitive. Trust paths in this case correspond to the transitive closure of the graph representation.

Examples of Trust-Exchange Mechanisms over the Web

Web services are at the leading edge of deploying highly distributed software components that can be published, discovered, and invoked seamlessly. They build on two of existing technologies, HTTP and XML, which are widely accepted and expected to dominate computing at least in the foreseeable future. Due to the higher level of abstracting the programming components of



	A	B	C	D	E	F
A	1	1	1	1	1	1
B	0	1	1	1	0	0
C	0	0	1	0	0	0
D	0	0	1	1	0	0
E	0	0	0	0	1	1
F	0	0	0	0	0	1

FIGURE 3.25 Graph representation of a web of trust across six heterogeneous domains adopting the transitive trust policy. The resulting transitive closure matrix is shown

network computing, web services appear to lay the foundation for composing service elements together to provide complex services. This composition capability may potentially revolutionize computing. It has all the aspects of achieving seamless web navigation in a way analogous to what users have experienced with the advent of manual navigation of the Web through browsers. Such composite computations over the seemingly unbounded frontiers of the Web further highlight the need for strong and reliable computational trust.

We look at three emerging mechanisms for the exchange of security constructs to enable trusted and secure Web computing, all of which are complementing each other. The first is a method for exchanging trust enabling constructs on Web service calls, web services security (WS-Security). The second one is a standard method for how to express trust and identity constructs in the computing web, the security assertion markup language (SAML). The third one represents a way to establish security sessions between a client and a remote service, Web cookies. A programming model in which these three techniques are used together expresses trust elements using SAML; transports the SAML statements using WS-Security and then maintains a session using Web cookies that contain SAML constructs.

Web-Services Security

Recently IBM, Microsoft, and VeriSign, Inc. have cooperated on the development of a Web-services security (WS-Security) specification submitted to the Organization for the Advancement of Structured Information Standards (OASIS) [OASI03]. Web services are at the leading edge of deploying integrated Web software components that can be published, discovered, and invoked seamlessly. Furthermore and due to their higher level of abstraction, Web services appear to lay the foundation for composing service elements together to provide complex services. This composition capability may potentially revolutionize computing. It has all the elements of achieving seamless Web navigation in a way analogous to what users have experienced since the advent of manual Web navigation driven through the end-user browser. Such composite computations over the seemingly unbounded frontiers of the Web further highlight the need for computational trust that can be established with reliability.

WS-Security is an attempt to retrofit security in the design of the Web-services protocol referred to as the simple-object access protocol (SOAP). It builds on existing mechanisms to generate security tokens for use across SOAP interlocutors referred to as *actors*. Data transfer in SOAP is based on exchanging XML documents. From a high perspective, such documents all adhere to a well-defined XML schema [W3CO02a] that governs the structure of SOAP messages. This structure consists of an enclosing envelope within which are nested zero or more control headers, followed by one body containing the application-level message payload.

Because WS-Security is an attempt to fit security into an already specified Web-service document format, the header portion of the document seems like a natural fit. The header element <Security> provides a means for attaching security-related information that can be targeted for a specific receiving entity. The latter can be an intermediate node traversed by the Web service or some other endpoint target.

A SOAP message can have multiple <Security> elements embedded in its header. Each of such elements may be designated to target a particular receiver specified through the S:actor attribute. Security information targeted to different receivers is required to appear within different <Security> elements. The omission of a S:actor attribute from a security element indicates that it is intended for consumption by all intermediate hopes of the message including the endpoint. Only one <Security> header block can omit the S:actor attribute, and no two elements can have the same S:actor attribute. This enforces a consistent rule in which security information that is targeted to all recipients or that is intended for a specific target is all structured respectively in a single <Security> element.

Security elements can be dynamically added to a Web-service message as it navigates the Web. Figure 3.26 depicts two examples of embedding security information within the <Security> elements of a SOAP message. In A we illustrate an acceptable syntax in which two <Security> elements are inserted, one targeted to a specific SOAP actor, while the second one is intended for all recipients. In B we show an invalid insertion syntax caused by having two <Security> elements targeted for consumption by all recipients.

A

```
<S:Envelope>
    <S:Header>
          ...
        <SecurityS:actor="weburi"
              S:mustUnderstand="TRUE">
          ...
        </Security>
        <Security S:mustUndertsand="TRUE">
          ...
        </Security>
    </S:Header>
...
</S:Envelope>
```

B

```
<S:Envelope>
     <S:Header>
              ...
         <Security S:mustUndertsand="TRUE">
              ...
         </Security>
         <Security S:mustUndertsand="TRUE">
              ...
         </Security>
         </S:Header>
...
</S:Envelope>
```

FIGURE 3.26 Inserting security elements in a SOAP message

As subelements are incrementally added to the <Security> header block, they are prepended to existing ones. The header therefore is an ordered sequence of elements combining security tokens, XML signatures, as well as encryptions. The processing of the security elements by a recipient is likely to be performed in accordance to this sequencing rule where no forward dependency across security subelements is permitted. When a subelement refers to a key placed in another subelement, the security token containing the key should be prepended following the subelement using that key. An example of that is a key-bearing subelement that contains an X509 certificate used for a signature. The X509 token in this case should be prepended following the signature subelement.

The security mechanisms that can be used in WS-Security may span technologies ranging from simple user identifier and password to more sophisticated constructs such as X.509 certificates and Kerberos tickets. Security elements may also contain signatures and encryptions computed over particular elements of the exchanged SOAP document. They also provide a natural transport for SAML assertions that can be attached to Web-services requests. We discuss the details of SAML shortly.

Identity and Trust Tokens

WS-Security provides an extensibility mechanism that can be exploited to embed any type of identity token. Three specific types of tokens are currently defined. You may attach a simple user-identifier token that consists of a user name and password, an X.509 v3 certificate, or a Kerberos v5 ticket. The types of tokens that can be used are classified in two categories: simple username tokens and binary tokens.

Simple User Name Token A user name token has the following XML structure:

```
<wsse:Security>
   <UsernameToken  Id ="...">
   <Username>
   ...
   </Username>
   <Password       Type ="...">
   ...
       </Password>
   </UsernameToken>
</wsse:Security>
```

The ID attribute can be optionally used to label the token. Username is a required element that specifies the identity of the token holder. The optional password element is intended to establish Username. Password information includes a type and a value. Protecting the password may require at least some level of transport security. Two formats for the password are currently defined by the optional Type attribute: a plaintext form and a bse64 encoding of the SHA-1 digest of the UTF8-encoded password.

Binary Tokens Binary tokens provide a way to embed cryptographic identity and privilege tokens in the security header block of a soap message. The parameterization of these tokens is based on two factors. The first one defines the type of encoding used. This allows the token to be handled appropriately. Two encoding types are currently specified:

❑ Base 64 encoding (wsse:Base64Binary) and
❑ Hex encoding (wsse:HexBinary).

The second parameter defines the type of the token's value. Three such types have been defined:

❑ X509 v3 certificate (wsse:X509v3),
❑ Kerberos v5 TGT (wsse::Kerberosv5TGT), and
❑ Kerberos v5 service ticket (ST) (wsse:Kerberos5ST).

wsse is the name space defined specifically for WS-Security. An X.509 certificate and its data components such as the public key can also be embedded in a <ds:KeyInfo> element defined by the XML name space of the digital signature standard [W3CO02b]. Below is an example illustrating the inclusion of an X509 v3 certificate as a binary security token within a <Security> element.

```
<wsse:Security>
      ...
<wsse:BinarySecurityToken
xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
      Id="myX509Token"
      ValueType="wsse:X509v3"
      EncodingType="wsse:Base64Binary">
      MITEZzIQEmt9CgCCAJZ0cqr5ihk...
</wsse:BinarySecurityToken>
      ...
</wsse:Security>
```

Referencing Security Tokens A token may be embedded in a security element by reference instead of value. Referencing a security token consists of specifying a URI for its location. The token can then be pulled by a relying party. This approach affords the advantage of having to marshal less data on a Web-services request. The following XML snippet illustrates the syntax of specifying tokens by reference:

```
<SecurityTokenReference
         Id="...">
      <Reference URI="...">
      </Reference>
</SecurityTokenReference>
```

SAML Approach: Unifying Trust and Identity Constructs

The security markup language (SAML) is an evolving standard that defines the syntax and semantics for XML-encoded statements that represent security assertions about a user or some programming entity [OASI02]. Assertions can be constructed by an initiating entity or can be acquired from a third party and presented to another entity where they are validated based on a predefined trust model. The unifying approach undertaken in SAML stems first from its generality and second from the fact that it represents a higher level of abstraction above any underlying security mechanisms, trust paradigms, transport, or the security protocols being used. Furthermore, SAML can be applicable irrespective of the trust model adopted whether it is a two-party or a third-party scheme. It lends itself to forming trust federations as assertions may span a large web of network endpoints and intermediaries.

With SAML, security decisions are not computed based on the traditional security context established by a controlling process in which an application executes. With SAML, an application acts as a container and provides a conduit for the security context associated with the underlying entity. This context therefore becomes exposed to the transaction level as opposed to the traditional paradigm in which contexts are managed and kept by control programs. Being part of the transaction's constructs, a SAML context follows the network routes taken by a Web application. As such, the flow of SAML constructs over a network may follow an arbitrary topology dictated only by the chain of requests with which they are associated. The depth of such request chains can be unbounded.

The vision of the network as a computer has indeed arrived with the federated Web-based applications that can be limited only by the scope of the Internet. The seamlessly unbounded journey of a network service request requires single sign-on of the initiating endpoint and transparent forwarding of user trust elements, such as authentication and authorization credentials. Furthermore, an adaptive dissemination of the user's profile elements that can be enforced by a dynamic and adaptive security policy is a key requirement for privacy control.

The SAML approach defines three types of identity management and trust assertions:

- *Authentication* The subject specified by the assertion was authenticated by a particular mechanism at a particular time. Authentication assertions merely state acts of authentication that happened in the past.
- *Authorization* The specified subject is either allowed or denied access to a particular resource.
- *Attribute* The specified subject is associated with the list of attributes provided in the assertion. Attribute elements define what is commonly known as a *user profile.*

An assertion may optionally be accompanied by one or more conditions constraining its validity. Assertions have a nested structure in which an outer generic element provides information common to all assertions. A series of inner elements representing authentication statements, authorization decision statements, and attribute statements all describe the specifics of the assertion. Instead of duplicating the statements issued via other assertions, one assertion may simply refer to those assertions via their unique identifiers (e.g., by a URI). Entities consuming assertions with external references to other assertions are responsible for resolving and validating those references as well as the assertions that they contain.

To broaden the scope of SAML and make it independent of any particular trust model, the concept of a SAML authority is introduced. SAML assertions are issued by SAML authorities that are distinguished based on the type of assertions they can issue. A SAML authority can be an authentication authority, an authorization authority, or an attribute authority. This distinction is conceptual and logical but is not necessarily physical as all types of assertions can be issued by a single authoritative entity. SAML distinguishes among three actors—a requester, a relying party, and an authority. The relying party is the entity that consumes and validates SAML assertions. The requester is the entity responsible for initiating the acquisition of assertions. A requester may also be considered a relying party, and thus one might broadly distinguish two main entities: an asserting party (an authority) and a relying party (consumer of SAML assertions). Figure 3.27 provides a conceptual view of the relationships across SAML entities. A dotted arrow linking an assertion type with a SAML authority indicates that the authority makes use of the assertion to issue new assertions. For instance, an authorization authority requires one or more authentication assertions to issue one or more authorization-decision assertions.

SAML authorities rely on various information sources to issue assertions. Most important, an external registry containing policy information may be consulted by an authority before an assertion is formulated. Additionally, SAML authorities may rely on previously issued and verified assertions to compute new ones. Requesting entities send existing assertions to SAML authorities when acquiring new assertions. Similarly, a SAML authority may pull referenced assertions from specified network URIs. In that respect, SAML authorities consume and produce assertions at the same time. On the other hand, clients, requestors, or relying parties can only be consumers of SAML assertions.

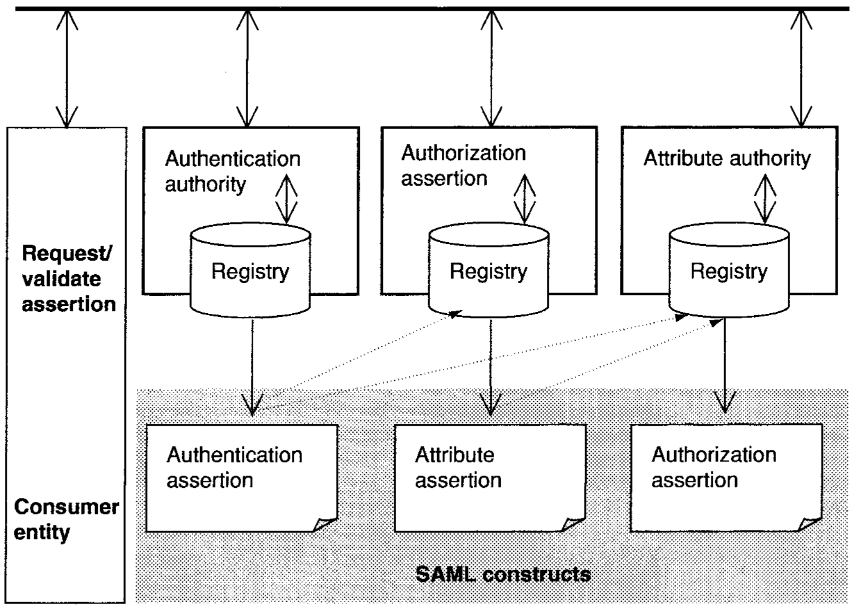


FIGURE 3.27 A conceptual view of the relationships across SAML entities

In addition to the syntactic and semantic definition of assertions, SAML defines a basic request and response protocol for the acquisitions of assertions.

SAML Constructs

Computations in SAML are performed over assertions. Each assertion is composed of a nonempty set of XML statements characterizing a particular subject with a temporal fact, such as an act of past authentication, an attribute, or a decision on whether access is allowed to a specific resource. The following is a discussion of major data elements of SAML.

Assertion An assertion is described by *AssertionType*, which is an XML complex type. This type specifies the basic information that is common to every assertion including the following attributes:

- ❏ *MajorVersion* A required attribute designating the major version of this assertion,
- ❏ *MinorVersion* A required attribute indicating the minor version of this assertion,
- ❏ *AssertionID* A required attribute uniquely identifying this assertion (a URI, for instance, can be used for such identification)
- ❏ *Issuer* A required attribute that unambiguously identifies the SAML authority that issued this assertion (an issuer might be identified by a URI), and
- ❏ *IssuerInstant* A required attribute specifying the time of issue in UTC.

Conditions This is an optional element that adds constraints to an assertion. The use of the assertion is subject to the constraints specified in this element. For example, a time constraint may set the validity of an assertion to some future time. Similarly, the validity of an assertion may be set to expire after a specified time.

Advice An optional element containing additional information that aids in processing an assertion.

Signature An optional element for marshalling XML signatures.

Statement This defines an extension point allowing the derivation of other statement constructs by an assertion-based application.

Subject Statement Defines an extension point from which other subject-related statements can be derived by various assertion-based applications. It contains a <Subject> element that defines a single entity associated with the statement. <Subject> encompasses two other elements: <NameIdentifier>, which identifies the subject by name and security domain, and an optional <SubjectConfirmation> element, which contains authentication information establishing <NameIdentifier>.

Authentication Statement This element is used by an issuing authority to indicate that the subject of the statement was authenticated by a particular authentication method and at a particular time in the past. An example of such assertion is shown below:

```
<saml:assertion    MajorVersion="1" MinorVersion="0"
    AssertionID="128.9.164.32.132547698"
    Issuer="Company.com"
    IssuerInstant="2003-04-26T11:03:00Z"
    <saml:Condition
                NotBefore="2003-04-26T11:03:00Z"
                NotAfter=""2003-04-26T11:10:00Z"
    <saml:AuthenticationStatement
                AuthenticationMethod="password"
                AuthenticationInstant=
                "2003-04-26T11:03:00Z"
                <saml:Subject>
                        SecurityDomain="Company.com"
                        Name="JohnDoe"
                </saml:Subject>
    </saml:AuthenticationStatement>
</saml:Assertion>
```

Authorization Decision Statement This element provides a statement by the issuer to the fact that the named subject is granted or denied access to a resource which is unambiguously specified by means of a URI. An example of an authorization decision assertion is shown below:

```
<saml:assertion     MajorVersion="1" MinorVersion="0"
    AssertionID="129.9.164.32.132547690"
    Issuer="Company.com"
    IssuerInstant="2003-04-26T11:03:00Z"
    <saml:Condition NotBefore="2003-04-26T11:03:00Z"
                    NotAfter="2003-04-26T12:10:00Z"
    <saml:AuthorizationDecisionStatement
          Decision="Permit"
          Resource="http://Travel.com/Servlet/reserve"
               <saml:Action
                     Namespace="http://WellknownURI">
                  Execute
               </saml:Action>
               <saml:Subject>
                     <saml:NameIdentifier
                         SecurityDomain="Company.com"
                         Name="JohnDoe"
                     </saml:NameIdentifier>
               </saml:Subject>
    </saml:AuthorizationDecisionStatement>
</saml:Assertion>
```

Attribute Statement This element underscores a statement by the issuer that the specified subject is associated with the attributes indicated. The following is an example of an attribute assertion:

```
<saml:assertion   MajorVersion="1" MinorVersion="0"
    AssertionID="130.9.164.32.132547691"
    Issuer="Company.com"
    IssuerInstant="2003-04-26T11:03:00Z"
    <saml:Condition NotBefore="2003-04-26T13:03:00Z"
                    NotAfter=""2003-04-26T13:10:00Z"
    <saml:AttributeStatement
          <saml:Subject>
                    SecurityDomain="Company.com"
                    Name="JohnDoe"
          </saml:Subject>
          <saml:Attribute>
                <saml:AttributeDesignator>
                AttributeName="Department"
                AttributeNamespace="http://Company.com"
                </saml:AttributeDesignator>
                <saml:AttributeValue>
                    Sales
                </saml:AttributeValue>
          </saml:Attribute>
    </saml:AttributeStatement>
</saml:Assertion>
```

Note how attributes are parameterized by names. This parameterization exemplifies the degree of flexibility in SAML. Furthermore, the name of an attribute is accompanied with a URI for the namespace in which the attribute is defined. Thus the semantics of an attribute is resolved to its defining source, which prevents ambiguity and collisions.

Trust Elements of SAML

SAML assertions are consumed by relying entities to establish subject identities and confine the use of resources to predefined policies. Affirming such assertions manifests itself through trust relationships that can be established between a relying party and the authority issuing the assertion. Trust establishment and verification in SAML is based on various constructs expressed through SAML assertions. In the following, we enumerate the major such elements that contribute to trust.

Digital Signatures The XML element <ds:Signature> may optionally be part of an assertion. When present, it represents an XML digital signature computed over the statements carried by the assertion. An assertion signed by an asserting party (AP) such as a SAML authority provides support for the integrity of the assertion, its authenticity, and possibly allows for nonrepudiation when a tamper-proof public-key mechanism is used. An assertion can also be part of a request message made to a SAML authority. Likewise, the signature over the assertion in this case supports data integrity, origin authenticity, and possibly nonrepudiation between the message originator and the destination authority.

User Confirmation A <SubjectStatement> contains a <Subject> element used to describe an active entity. In turn, the <Subject> element consists of two nested elements: <NameIdentifier>, which specifies a subject by name in accordance with a particular naming scheme such as in X.509 [HOUS99a], or an email address based on IETF RFC2822 [RESN01]. The second element is <SubjectConfirmation>, used to provide data allowing the subject to be authenticated. This element may encapsulate any authentication token or credential that can lead to establishing the named identity.

Authority Binding Information The <AuthorityBinding> element may optionally be part of an authentication statement. It can be used to indicate to a relying party that a SAML authority may be available to provide additional information about the subject of an assertion. This authority is specified by location and through its supported protocol binding.

Authorization Evidence An authorization statement may optionally contain an <Evidence> element that carries an assertion used by the issuer in making the authorization decision. This assertion can be specified either by value or by reference. Authorization evidence may also be supplied by an entity requesting an authorization decision from a SAML authority.

Other Trust Elements of SAML

Other elements of trust in the SAML definition for an assertion include the name of the issuer <Issuer>. A name in the form of a URI allows a relying party to inquire further information about the subject of the attribute to verify a particular trust relationship. The time of issuance of the assertion <IssueInstant> as well as a validity interval as defined by the <Condition> element allow for the timely usage of an assertion. Additionally an <Advice> element may encompass further trust-related information about the assertion.

A Note on Federated Trust in SAML

Federated SAML authorities are expected to play a key role in the proliferation and success of the SAML constructs over the Internet. Forwarding SAML authentication and authorization assertions across security domains without re-authentication requires the existence of a well-defined trust across participating SAML authorities. SAML in itself has not introduced a new federated trust paradigm; rather, it relies on existing models of trust such as those based on PKI or Kerberos for instance. Trust verification in this case will ultimately involve the low-level mechanisms producing the SAML constructs.

Web Cookies

The HTTP protocol that made the World Wide Web a household name is stateless and simple. The statelessness of HTTP precludes the need for managing persistent sessions and all the complexities that may arise thereof. Users connect anew and identify themselves whenever needed, each time they navigate a Web link even with the same server. Although they face a number of reliability and security issues, cookies were invented as an ad-hoc mechanism to establish continuity and sate on the Web. Cookies are data constructs that are initially sent from a Web server to the client's browser environment, referred to as a *user agent* and subsequently exchanged between the browser and Web servers visited by the user. They can serve many purposes from the basic functions of keeping track of the display mode that a user selects (e.g., graphic frames or text only) to representing the current state of a shopping cart for a Web store buyer. The concept of cookies is an interesting one in that it simplifies managing HTTP states by involving the client yet in a seamless manner. An end user is generally unaware of cookies placed in his or her machine. The server maintains no state constructs in its runtime except for when they arrive through client cookies. The server is said to forget about the client until the latter reminds it of who he or she is.

Structure of Cookies

Cookies have a flat data structure that is simple and easy to manipulate. A cookie is a sequence of attribute name and value pairs as defined in the IETF RFC 2965 [KRIS00]. A few control attributes are introduced by the standard. The most important aspect, however, is the generality of attribute-value pairs that can be marshaled into a cookie. Application-level attributes can be arbitrarily defined as indicated by the following syntax:

```
av-pairs   =   av-pair(";" av-pair)*
av-pair    =   attr ["=" value];optional value
attr       =   token
value      =   token | quoted-string
```

Attribute names, instances of attr, are case-insensitive. While the above syntax shows value as optional, evidently most attributes will have values associated with them. Figure 3.28 illustrates the structure of a generic cookie.

Server Role

A server application that needs to establish a cookie-based session with a particular client returns cookie information in the HTTP response header preceded with the label of "Set-Cookie2" as shown by the syntax below.

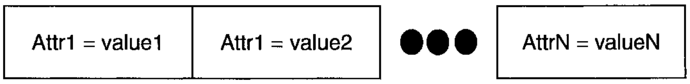


FIGURE 3.28 Generic structure of a Web cookie

```
set-cookie          =     "Set-Cookie2:" cookies
cookies             =     1#cookie
cookie              =     NAME "=" VALUE(";" set-cookie-av)*
NAME                =     attr
VALUE               =     value
set-cookie-av       =     "Comment" "=" value
                    |     "CommentURL" "=" <"> http_URL <">
                    |     "Discard"
                    |     "Domain" "=" value
                    |     "Max-Age" "=" value
                    |     "Path" "=" value
                    |     "Port"[ "=" <"> portlist <"> ]
                    |     "Secure"
                    |     "Version" "=" 1*DIGIT
portlist            =     1#portnum
portnum             =     1*DIGIT
```

The Set-Cookie2 response header comprises the token Set-Cookie2: followed by a list of one or more comma-separated cookies. In turn, each cookie begins with a required NAME=VALUE pair representing the cookie name, followed by zero or more semicolon-separated attribute-value pairs. Among the standard control attributes we point out the following list, which is to some degree relevant to the security and reliability of the cookie mechanism:

❑ The optional Path attribute specifies the server URLs for which the cookie is applicable.
❑ The optional Port attribute restricts the ports to which a cookie may be returned by a client in an HTTP request header.
❑ The optional Secure attribute (with no value) indicates that the cookie is secure. The security level or mechanism by which the cookie is protected is unspecified and remains application-specific. When the client sends a "secure" cookie back to the server, the level of security as indicated by the server should not be downgraded.
❑ The presence of the optional Domain attribute specifies the domain name for which the cookie is valid. Generally, the domain of the server is the one specified, although cookies can also be generated by one server and consumed by another server located in a separate domain. This attribute is a bit of information that can be used to further extend the generation and consumption of cookies across federated domains.

- The optional attribute Max-Age represents the lifetime of the cookie in seconds. A value of zero means the cookie should be discarded immediately. The absence of this attribute can be interpreted as representing an indefinitely valid cookie.
- The optional attribute of Discard is used to instruct the client program (the browser, for example) to discard the cookie unconditionally when it terminates.
- The optional attribute of CommentURL is used by the server to inform the client of any privacy-related information as well as the intended use of the cookie. The client agent should give opportunity to the user to inspect this information before he or she initiates a request.

Client Role

When a client wishes to continue interacting with a server, it returns cookie information in the HTTP request header based on the Set-Cookie2 data that it had received. The cookie header sent from the client to the server adheres to the following syntax.

```
cookie           =  "Cookie:" cookie-version 1

((";" | ", ")* cookie-value)
cookie-value     =  NAME "=" VALUE [";" path] [";" domain]
                    [";" port]
cookie-version   =  "$Version" "=" value
NAME             =  attr
VALUE            =  value
path             =  "$Path" "=" value
domain           =  "$Domain" "=" value
port             =  "$Port" [ "=" <"> value <"> ]
```

Attributes values returned by the client reflect those sent by the server through Set-Cookie2.

Cookies already stored at the client side can be sent to the server based on the following:

- The host and port designated by the request,
- The URI of the request, and
- The age of the cookie.

Example: Cookies Exchanged Between a Client and a Web Server

The following steps illustrate cookies exchanged between a client and a web server presented through a fictitious URL of http://www.webstore.com. It is assumed that the client has no stored cookies for the server and he just visited the home of webstore.com that displays a login form. The client fills and then submits the form. The server receives client log on information and processes it. Subsequent interactions between the client and that same server result in the following exchange of cookies.

Server → User

```
Set-Cookie2:Customer="JohnDoe";Version="1"; Path="/webstore"
Cookie identifies the client.
```

User → Server User selects an item to order.

```
  Cookie: $Version="1"; Customer="JohnDoe"; $Path="/webstore"
  [form data]
Server → User Shopping basket contains an item.
  Set-Cookie2: Part_Number="Diesel_Engine_101";
  Version="1";Path="/webstore"

User → Server User selects shipping method from form.
  Cookie: $Version="1";Customer="John Doe"; $Path="/webstore";
  Part_Number="Diesel_Engine_101"; $Path="/webstore"
  [form data]

Server → User New cookie contains shipping method.
  Set-Cookie2: Shipping="UPS"; Version="1"; Path="/webstore"

User → Server User chooses to process order.
        Cookie:$Version="1"; Customer="JohnDoe;
  Part_Number="Diesel_Engine_101";
  $Path="/webstore";Shipping="UPS";
  [form data]

Server → User
Transaction is complete.
```

Issues with Use of Cookies

The concept of cookies is controversial in a number of aspects. Foremost is the ability of a Web server to push data constructs into a user's machine. This process may in fact be taking place without the user's full awareness of potential consequences. Nonsavvy users in many cases are not cognizant of what a cookie is. Indeed, this paints an element of intrusion under the auspices of normalcy and thus users will tend to accept cookies. The user's Web navigation behavior can be easily tracked thereby raising concerns over privacy. Malicious servers may attempt to flood a user's machine with cookie files. The transparency of uploading cookies to Web servers, the fact that cookies issued for one host may be consumed by another one, and cookies stored in one machine can be copied and used on another machine all are factors that increase the risks associated with cookies.

The risk factor is further exacerbated with the misuse of nonsecure cookies for identity management, such as authentication, single sign-on, and for carrying entitlements. Although the IETF standard for the use and management of cookies emphasizes the adoption of informed consent where the end user is made aware of cookies, the potential for misuse can be abound, particularly when in fact the user is subsumed by his or her agent, the browser. The fact that a cookie generally tends to have a lifetime that is sufficient enough for an intruder or a malicious user to modify it or completely regenerate it with new information poses a considerable risk. Park and Sandhu [PARK00] classify threats of using cookies into three types: network threats, end-system threats, and cookie-harvesting threats. Network threats can be carried by intercepting HTTP requests and responses, extracting cookies, and implanting them for a malicious use. The use of secure connections such as SSL protects cookies during transport but leaves them in cleartext once they reach an endpoint. End-user threats stem from the fact that cookies can be easily altered and copied from one machine to another. Attackers can therefore forge cookies and perhaps impersonate other users in a scheme of identity theft. An attack for harvesting cookies can be mounted by a Trojan Web site that impersonates a site that accepts cookies from users. The harvested cookies can later be used to compromise all other sites accepting them.

Secure Cookies

The level of security required by cookies depends on the sensitivity of information carried in a cookie, the type of potential threats and risks, as well as the cost incurred in the event of a compromise. Usage of cookies may require data integrity, origin authenticity, and confidentiality. Despite the controversy surrounding it, the cookie paradigm can be securely and reliably exploited to the benefit of Web computing. Sometimes an encrypted transport channel such as one using SSL/TLS is established between a client and a server to encrypt the entirety of a data payload exchanged just because a few bytes of the payload require confidentiality. Instead, one might use cookies with only the sensitive information encrypted.

Any reasonable level of secure cookies will, in all likelihood, require encryption. We distinguish three scenarios in which encryption of cookies may take place.

Use of a Public Key on the Client Side Cookie information can be signed, encrypted, or both signed and encrypted using the private key of the client. Decryption as well as signature verification is performed by the destination server. The public key of the client is established by the server according to a predefined PKI trust scheme. This approach is applicable in situations where the client is sending information that has no risk of exposure but requires integrity and origin authenticity. An example would be the signing of a shopping-cart cookie so that some level of nonrepudiation can be achieved.

Cookies secured in this fashion can be used across multiple servers provided the certified public key of the client is available.

Use of a Public Key on the Server Side In this model, the server uses its private key to sign or encrypt cookies before they are pushed into a client machine. The client may elect to verify signed cookies to establish server authenticity. The server may choose to encrypt sensitive information from the user's profile or other session-related information using its public key. When such a cookie bounces back on the server side, the server uses its own private key to decrypt it and thus the cookie is guaranteed confidentiality, data integrity, and authenticity of the origin server. In this scenario, encrypting a cookie with the server's public key is relevant to sensitive data. Server signing of the cookie enables data integrity, and enforces authenticity of the origin server. Simply encrypting cookies using the server's public key, however, is not adequate since the server's public key can be available to other entities and thus eavesdropping and impersonation may take place. Such encryption should be performed over data that is signed by the server to ensure both confidentiality of cookie information and origin authenticity of the server.

Use of a Shared Secret Key A symmetric encryption key shared between a client and a server may also be used to encrypt cookie information or apply a keyed MAC to cookies requiring data origin authenticity and integrity. When the client origin authenticity is required, however, a shared secret key needs to be distinct for each client-server pair. This does not lend itself to scalability and faces the key distribution issue. A session key established through key exchange protocols such as the encrypted key exchange (EKE) or Diffie-Hellman can also be used [DIFF76a, BELL92].