CHAPTER 14

PUBLIC KEY CERTIFICATES

16 And he causeth all, both small and great, rich and poor, free and bond, to receive a mark in their right hand, or in their foreheads:
17 And that no man might buy or sell, save he that had the mark, or the name of the beast, or the number of his name.

— Revelation of St. John the Divine 13:16–17 *King James Version*

IN THIS CHAPTER

We use public key certificates to reliably associate a particular identity to a particular public key. Most of the examples here have to do with authenticating Web servers, but the concepts also apply to authenticating people's public key certificates.

- Public key masquerade and the basics of certificates
- Strategies for issuing public key certificates
- Revoking public keys
- Using certificate-based authentication with Kerberos

14.1 TYING NAMES TO PUBLIC KEYS

If we want to authenticate a person or computer off-line, then we need a copy of his or her public key. As long as we can get a copy of the right public key for Cathy or Tim or the Acme.com Web site, then we can accurately authenticate them across a network. However, attackers can undermine this by providing bogus credentials.

The simplest and most obvious approach is for someone named Henry to convince people and software that his own public key is the one used by some important computer, like a bank's server.  *A-84*
Henry might be able to profit even if he tricks only one or two peo-

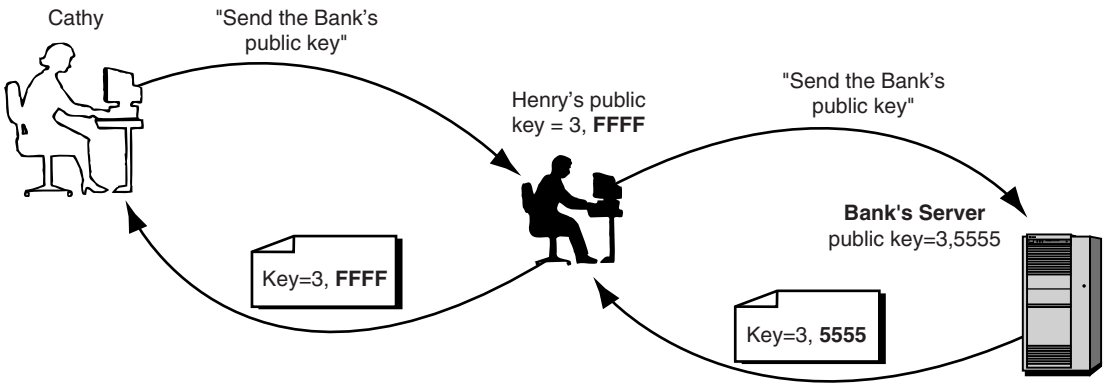

FIGURE 14.1: *Henry performs a "man in the middle" attack.* Cathy asks for a copy of the bank's public key so that she can establish a secure connection to it, perhaps using SSL. Henry intercepts the bank's reply and substitutes his own public key for the bank's. Whenever Cathy uses that public key, Henry intercepts the message and reencrypts it with the bank's public key. Neither Cathy nor the bank can detect Henry's presence.

ple this way. For example, Henry could send a message to Cathy pretending to be the bank, and give Cathy the "bank's public key" when it's really his own key. If Cathy then tries to send secret information to the bank, like password-protected messages, Henry can masquerade as the bank.

Cathy will probably uncover this scam when she realizes that the bank really hasn't received any of her transactions. But this isn't enough to guarantee that Henry's fraud will always be detected. There's also a risk that Henry could systematically participate in all of Cathy's transactions with the bank, substituting his own public key for the bank's. This produces yet another the man in the middle (MIM) attack (Figure 14.1).


A-85

To implement the attack, Henry places himself (or more likely, some software of his) between Cathy and the bank, and intercepts every message they send back and forth. When the bank sends Cathy its public key, Henry substitutes his own key for the bank's key. If Cathy sends the bank some secret information encrypted with its public key, she unintentionally uses Henry's key instead. So Henry intercepts the data, decrypts it with his own private key, reencrypts it with the bank's own public key, and sends it on. Neither Cathy nor the bank can tell the difference. But Henry can eavesdrop on everything Cathy does with the bank.

The solution to this problem is attributed to Leon Kornfelder who, as an MIT undergraduate, proposed the use of *certificates* to thwart such attacks. Figure 14.2 shows the essential contents of a public key certificate. The certificate is a data item containing a public key, the name of the key's owner, and a digital signature. Figure 14.3 shows how a Web browser displays a certificate. We verify a certificate's authenticity by checking its digital signature. Certificates help us detect public key substitutions and MIM attacks. The attacker can't insert a bogus public key if we can verify the key's owner.

D-84

see Note 1.

CERTIFICATE AUTHORITIES

The first question posed by certificates is to decide who signs them. The signer must fulfill two requirements: the signer must be reliable, and the signer's public key must be available to check signatures. We call the signer the *certificate authority*.

By "reliable" we mean that the authority must not sign a certificate without being confident that the name on the certificate really goes along with the public key. Imagine what would happen if Henry could acquire a certificate containing the bank's name along with his own public key: he could sign electronic documents and authenticate himself to remote systems using the name "www.bank.com." This problem actually struck Microsoft Corporation in early 2001: someone posing as a representative of Microsoft submitted public

A-86

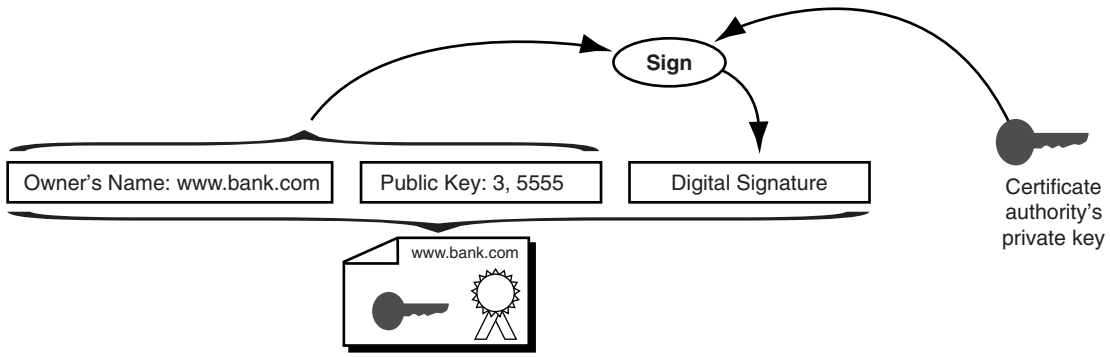

FIGURE 14.2: *Basic public key certificate.* Here is a basic certificate for the bank. Like all public key certificates, it is a data item contains the owner's name ("www.bank.com"), the public key ("3,5555"), and a digital signature. The certificate is produced by a trustworthy third party, the "certificate authority," whose private key is used to sign the certificate. To verify a certificate, we must use the public key belonging to the authority that signed the certificate.

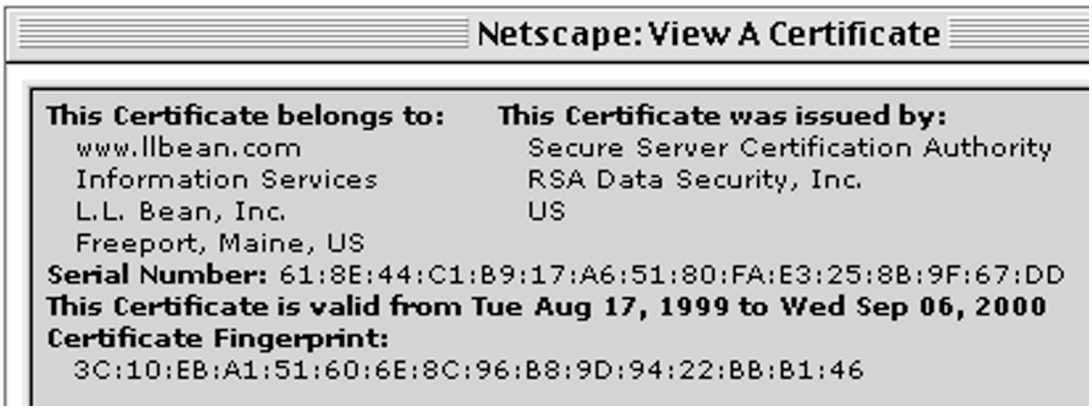

FIGURE 14.3: *A public key certificate displayed by a browser.* This certificate was issued to L. L. Bean of Freeport, Maine, to authenticate their Web server when customers use the server to buy merchandise.

keys to Verisign, a major certificate authority, and Verisign issued certificates claiming that Microsoft owned those keys. It later turned out that the corresponding private keys were not in fact in Microsoft's possession. The resulting certificates would allow the owner of the private keys to produce software that could masquerade as legitimate software released by Microsoft itself.

see Note 2.

The obvious defense is to enforce that first requirement for a certificate authority: the authority must ensure that the name really goes along with the key. This generally means that authorities must ensure that the owner of a particular public key pair is really entitled to use the name. Authorities do this by establishing rules describing how they verify that the person has rights to the name. If Cathy asks for a certificate identifying her as Vice President of Art at Acme Corporation, then she needs to show that she really holds that position. With respect to the Microsoft incident, at the time of this writing it is not yet clear how Verisign was tricked into issuing the bogus Microsoft certificates; Verisign's rules and procedures should have prevented it.

D-85

The second requirement for signing certificates is that the certificate authority's public key must be available to check signatures. To be practical, the key can't simply be "published information" that's available after some digging, no matter how little. The key must be

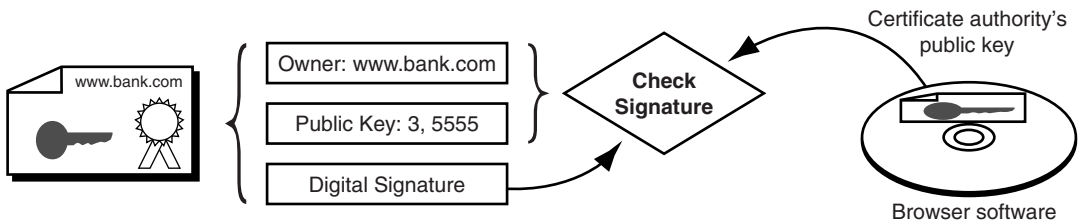

FIGURE 14.4: *Browser authenticating a public key.* This was the original strategy in Netscape Navigator for authenticating a public key. All keys were in certificates, and all certificates were signed by a single certificate authority. The authority's key was embedded in the software. Today, browsers usually contain many such keys in an "authority list," as described in Section 14.4.

immediately at hand so that public key software can use it to authenticate certificates issued by that authority.

Netscape faced this problem when first deploying software to use SSL for Internet electronic commerce. The software ensemble consisted of Web server software, the "Commerce Server," and Netscape's well-known "Navigator" browser software. A server would authenticate itself to a browser by sending a public key certificate. Netscape arranged for RSA Data Security, Inc., to serve as the certificate authority and issue the public key certificates. RSA established a separate organization to do this, which eventually became Verisign, Inc. The authority used a particular public key pair to sign all of those early certificates. Since all certificates were signed with that particular key, Netscape embedded the corresponding public key in Navigator's SSL software, as shown in Figure 14.4. The browser used that key to authenticate certificates used by SSL. *see Note 3.*

USING THE RIGHT CERTIFICATE

Since the public keys are large and the processing is complicated, we have to rely on our workstation to do the authentication processing. This can be a double-edged sword. For example, it's not enough for our browser to receive an authentic public key certificate when we contact an e-commerce site. We need to receive the certificate for the site we think we're using.

For example, an attacker might be the owner of a legitimate e-commerce site and have a legitimate public key certificate for the site's server. What happens if he implements a second server that  *A-87*

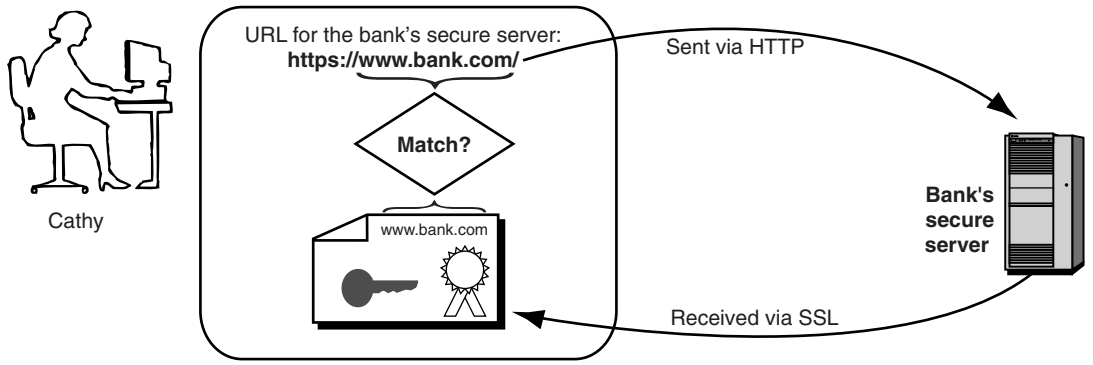

FIGURE 14.5: *Browsers verify the certificate's name.* When Cathy contacts the bank's secure server, the bank's host name appears in the URL. The bank sends back its public key certificate as part of the SSL handshake protocol. The browser compares the host name in the certificate against the domain name in Cathy's URL to verify that it is speaking to the right host.

masquerades as www.bank.com, and intercepts Cathy's messages to the bank? We are essentially back to the problem that opened this section.

In part, this problem arises because the SSL security software operates at one software level, ignoring the data it carries, while the Web server or other application software operates at a completely different level. The solution to the problem is for the application software itself to verify that the name in the certificate goes along with the messages being handled by the application. In the browser environment, Netscape solved this by checking the name in the certificate against the server being asked for in the HTTP (Figure 14.5). This is like checking the name of the account holder on an electronic check against the name in the public key certificate. The check is legitimate if the certificate name matches the account's name.

D-86

Figure 14.6 shows how a browser responds if the check shown in Figure 14.5 fails. The browser has tried to open an SSL connection to the site "www.sprintpcs.com" and received a certificate to use to establish the connection. However, the certificate it received did not contain the host name "www.sprintpcs.com" in it. So the browser displays a dialog that briefly identifies the host name, some basic identifying information on the certificate, and asks the workstation's user to decide if the certificate really goes with the site. This type of problem can occur if the site's proprietor orders a certificate that

FIGURE 14.6: *Browser warns of a certificate containing the wrong name.* Here, Netscape Navigator has been given a certificate to secure a connection, but the site name in the URL does not appear in the certificate. This may have been caused by installation errors, or it could indicate an attempted MIM attack. Verify that the name on the certificate is appropriate for the site, and that the certificate authority (indicated above by "Signed by") is well known and trustworthy.

contains the wrong host name or no host name, or if the host name needs to be changed later for technical or administrative reasons. However, the problem may also show up as part of a public key MIM attack. Essentially, the user must decide if it's likely that the certificate authority who signed the certificate ("RSA Data Security") would have issued a certificate with the name "Sprint PCS" to someone other than the owner of "www.sprintpcs.com." In this case, the certificate seems legitimate, and might be accepted safely. On the other hand, it might be risky to accept a certificate belonging to "Henry's Internet Company" when we try to connect to "www.bank.com." There is no obvious relationship between the URL and the name in the certificate, and that's a bad sign.

Microsoft faces a variant of the certificate-matching problem with their mechanism to verify that downloaded software really comes from Microsoft. This mechanism is intended to prevent attackers from distributing subverted software that claims to be from Microsoft. Major components that install or run downloaded software include procedures to verify a digital signature on the software. The procedure contains a list of Microsoft certificates to verify those signatures. Verisign unintentionally created some bogus certificates

to verify such signatures, but those certificates are not in the list of accepted certificates. So, even though there exists a bogus code-signing certificate that contains the name "Microsoft Corporation," nobody is likely to be tricked by that certificate into loading bogus software. *see Note 4.*

14.2 Creating Certificates

There's no special trick to creating a public key pair if we have the appropriate software handy, but we usually create the certificate separately. Figure 14.7 shows Cathy creating her key pair and acquiring a certificate. First, she creates her key pair within the confines of her computer and keeps her private key secret. Then she transmits a copy of her public key, along with her name, to the certificate authority that will issue her a certificate. The authority, if it is satisfied that the key really goes along with Cathy's name, will combine the name and key into a certificate data structure and affix its digital signature. After that, the authority can transmit a copy back to Cathy, publish the certificate in its own directory, if desired, or distribute the certificate in other ways.

Modern Web browsers can generate public key pairs and submit the public key for certification. One can visit the Web site of a certificate authority, generate a private key within one's browser, and submit the corresponding public key to the authority for certification. On the other hand, some authorities require higher confidence in a person's identity than one gets on-line. For example, certificates for e-commerce applications often require notarized documents attesting to the applicant's right to use the specified domain name. The applicant would need to submit the paperwork along with a machine-readable copy of the public key via a delivery service.

An alternative strategy that is used in some situations is for the proprietor to generate all key pairs and certificates and to handle the distribution of both. Thus, a user like Cathy needs only to receive her key pair and certificate from the site's proprietor and she doesn't need to bother with the key generation process herself. She only has a single task to perform: to install her key material once she receives it. Lotus Notes traditionally uses this approach for distributing public key pairs. This carries an increased risk simply because the private key is created and a copy may be archived by

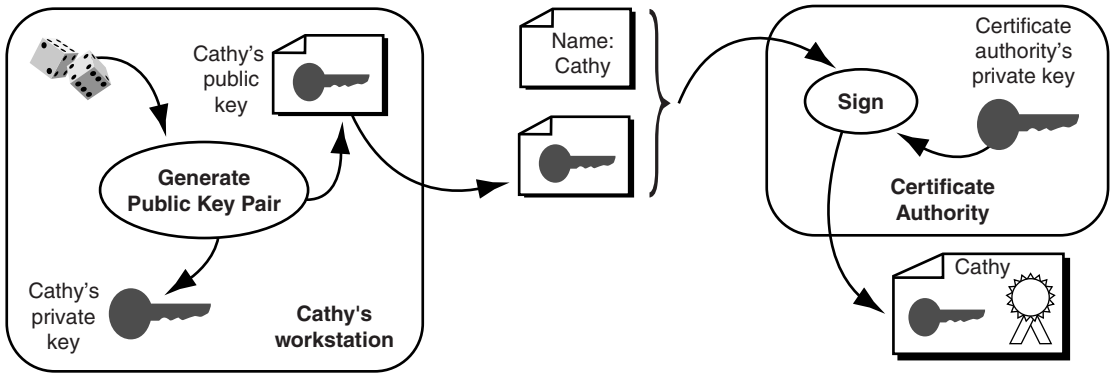

FIGURE 14.7: *Creating a public key certificate.* First, Cathy produces her public key pair within her workstation. Next, she transmits a copy of her public key along with her identifying information to the certificate authority who issues the certificate. The authority verifies that Cathy has the right to use her name, and that she in fact owns the corresponding private key. Then the authority constructs the certificate and affixes its digital signature.

the proprietor. On the other hand, it simplifies configuration for end users and reduces the risk of losing important information if a user's private key is accidentally destroyed.

As a practical matter, modern certificates contain a good deal more information than just the key, the owner's name, and the signature. At least, they need to identify the certificate authority so that we know what key to use to verify the signature. In addition, certificates tend to carry other information to make them easier to use reliably. These include:

- Date of issue
- Expiration date
- Version of this certificate's format
- Indications of types of keys contained
- Owner's name in various formats
- Other properties of the owner (address, for example)
- Owner's rights and privileges

The question of what information really belongs in a certificate is subject to dispute. Some of the information has evolved for practical reasons: the validity dates help minimize problems with invalid keys since they ensure that all keys "die" eventually. The owner's name

has to appear in domain name format so that browsers can compare it against domain names in URLs, while "official" owner names tend to appear in a different, incompatible format. Some experts argue that certificates work best if simply restricted to the job of associating names with keys. Others argue in favor of added features, like the owner's citizenship, date of birth (for age information), or other personal data that may be useful in Web applications. Others have suggested adding fields to certificates to store information about computer system privileges and access permissions. Standards for public key certificates can support a number of extensions, but most certificate products omit personal data and privilege information from their certificates.

CERTIFICATE STANDARDS

Public key certificates today are usually based on X.509, a standard originally established for X.400 electronic mail systems. This seemed like a good choice in the early 1990s, since a number of major organizations had announced their intention to use X.400 for future electronic mail systems, including the U.S. Department of Defense. Today, X.400 is primarily of historical interest, except for its impact on public key certificate formats. *see Note 5.*

In particular, X.509 has had two major impacts on public key certificates: *Abstract Syntax Notation #1* (ASN.1) and *distinguished names*. X.509 defines the contents and arrangement of data in a certificate using ASN.1 statements. Figure 14.8 gives an ASN.1 example from a 1993 Internet certificate specification. The individual statements provide labels for identifying the different fields and type specifications to show what type of information is stored in particular fields. Software developers use ASN.1 compilers to generate the code to build the corresponding data structures and to extract data from them. *see Note 6.*

A distinguished name is a name containing several separately named elements that make the name identify a particular person uniquely. Often, the elements represent levels of a hierarchy. For example, a distinguished name could be based on geography, so that the highest level identifies the country, then the state, city,

street address, and then the individual. For John Doe, this might yield the following distinguished name:

/C="US"/SP="Minnesota"/L="Red Wing"/PA="123 Main Street"
/CN="John Doe"

This hierarchical approach gives us a good way of assigning certification responsibilities and of ensuring uniqueness. We can assign certification authorities to be responsible for different "suffixes" of the distinguished name. For example, there could be a "Red Wing" authority that issues to all Red Wing addresses, and a separate one for each city of each state. These authorities would ensure uniqueness among their own certificates. Since each authority issues certificates with a different city and state, no certificates will be duplicated if each authority avoids local duplicates.

```
Certificate ::= SIGNED SEQUENCE{
       version [0]           Version DEFAULT v1988,
       serialNumber          CertificateSerialNumber,
       signature             AlgorithmIdentifier,
       issuer                Name,
       validity              Validity,
       subject               Name,
       subjectPublicKeyInfo   SubjectPublicKeyInfo}

Version ::= INTEGER {v1988(0)}

CertificateSerialNumber ::= INTEGER

Validity ::= SEQUENCE{
       notBefore             UTCTime,
       notAfter              UTCTime}

SubjectPublicKeyInfo ::= SEQUENCE{
       algorithm             AlgorithmIdentifier,
       subjectPublicKey      BIT STRING}

AlgorithmIdentifier ::= SEQUENCE{
       algorithm             OBJECT IDENTIFIER,
       parameters            ANY DEFINED BY algorithm OPTIONAL}
```

FIGURE 14.8: *ASN.1 syntax for a public key certificate.* This definition is taken from RFC 1422, the standards for Privacy Enhanced Mail (PEM), and uses ASN.1 to describe a simple certificate.

CERTIFICATES AND ACCESS CONTROL

This is an open problem—if our software can process a broad range of certificates from numerous sources, then we can reliably authenticate a large population of people and computers. On the other hand, we must then contend with the problem of access control. Note how Netscape would accept any secure server as long as the certificate name matches the host domain name. That level of access control isn't sufficient for every application. It's like using any mag stripe on any card as a credit card. What if the card is a premises-access badge or a driver's license? Neither say anything about the creditworthiness of the owner. Authentication is one problem, and access control is a completely different one.

14.3 CERTIFICATE AUTHORITIES

We've talked briefly about how certificates are created and formatted, but we have only slightly touched on the whys and wherefores of certificate authorities. What makes a particular person or organization into a good choice for a certificate authority? We examine that question here.

The fundamental question is a trade-off between risk and convenience, like so many other security questions. Errors in certificates will yield mistakes in authentication—the system will recognize data as coming from someone who did not create it. On the other hand, extra caution in certification will increase operating costs.

One way to consider this trade-off is to look at a similar system that has evolved over the past 40 years: the credit card system. Credit cards provide third-party authentication for valuable transactions: John Doe shows his credit card to a clerk, and the card attests to the fact that he will pay for a purchase in the future. Credit cards have gone through several phases over the past decades that parallel some issues facing certificate issuance:

- **Phase 1:** Pioneering organizations like American Express produced special cards for charging travel and entertainment expenses from certain merchants.
- **Phase 2:** Every oil company, department store, and airline issued its own credit card for the convenience of its customers. A creditworthy person might have dozens of credit cards.

- **Phase 3:** Cross acceptance in which companies merged their credit operations and accepted one's cards at the other's stores.
- **Phase 4:** Consolidation in which major cards like American Express, Visa ("Bank Americard"), and MasterCard ("Master Charge"), arranged to be accepted by numerous vendors, making individual cards less necessary. The major cards competed with one another by touting the vast number of places where they were accepted for payment.

Credit cards are worth comparing to public keys because, in an ideal world, an individual's public key might serve many purposes much as a Visa or MasterCard does today. While general-purpose certificates might have large benefits, they also pose practical problems not unlike those faced in the evolution of credit cards.

PROPRIETORS AS CERTIFICATE AUTHORITIES

If we follow the traditions of older authentication systems, we would expect a computing site's proprietor to be fully responsible for issuing certificates. So, if Acme Corporation wishes to issue certificates to its employees, then it purchases the necessary hardware and software and issues them from its information services department. This is similar to Phase 2 of our credit card history: the era when individual organizations issued their own cards.

The big benefit of handling the entire process in-house is that the proprietor could control costs. This was also true of credit cards. A company could issue credit cards for free to anyone they wanted if they controlled the issuance. They could ask whatever questions they wanted on the credit application in order to reduce their risks, or ask no questions at all. They could control how long the cards were valid before they expired, and how much people could use the cards before paying.

Much the same is true for a company issuing its own certificates. The company can structure the enrollment and distribution process to meet its own targets for administrative costs. The company can take the risks it thinks appropriate and establish the requirements needed to keep risks in line.

On the other hand, the process requires an infrastructure that the company must pay for. Credit card operations became rather large

at some companies and, thanks to interest payments, they actually became profit centers in some cases. It's not clear how certificate authorities would ever become profit centers, since they have the potential to require a somewhat costly infrastructure. The U.S. government spent as much as $40 per telephone per year to maintain public key certificates for STU III secure telephones in the mid-1990s. *see Note 7.*

The shortcoming of issuing one's own keys is that the approach can be hard to scale. If we need Acme's certificates to be accepted by most organizations, then most organizations need a copy of Acme's public key. If every company in the Fortune 500 has its own certificate authority and we must check their certificates with those keys, then public key software systems will need to keep track of a huge number of separate authorities.

COMMERCIAL CERTIFICATE AUTHORITIES

Commercial certificate authorities provide a service akin to major credit cards: they're established entities recognized in most places. If a Web site has a certificate issued by a well-known authority, like Verisign, then the certificate will be automatically recognized by most users' browsers. Otherwise, the browser has to take extra steps to accept the certificate, and must generally ask the workstations' operator for permission. Naturally, it's best to avoid the need for such interactions.

Commercial Web sites generally acquire certificates signed by commercial authorities because most browsers will automatically recognize such certificates. This is obviously a good thing. But what about a site that's issuing certificates for individual users? For example, what if a site is using a public key protocol to authenticate people? What does it mean for such a site to hire a commercial certificate authority?

In that case, the proprietor is essentially delegating the enrollment process to the commercial certification authority. Since the authority controls the signature that appears on certificates, the proprietor must depend on the authority doing enrollments correctly. The proprietor must be willing to accept some level of risk associated with incorrect enrollments by the authority. In some cases, the risk might be acceptable in exchange for the convenience of using exist-

ing browser configurations. In other cases, like if forged credentials pose a serious risk, then a commercial authority might not be the right choice.

Since certification is a relatively new type of business, commercial authorities are cautious about the services they offer. In general, they will provide certificates when presented with convincing evidence of identity, but they don't in general accept any financial liability for making incorrect decisions. This situation is generally set forth in the authority's *certification practices statement*. Such statements are intended to provide customers with a clear explanation of the services the authority provides. In general, the statement also explains that the authority does not offer the services such that it takes on the potential liability from errors in authenticating people. Ultimately, the liability rests with the proprietor who incurs the loss caused by an improperly issued certificate. This is the principal shortcoming of commercial authorities. *see Note 8.*

14.4 Public Key Infrastructure

The *public key infrastructure* (PKI) is the collection of organizations, mechanisms, protocols, and procedures that create, certify, and distribute public keys. A PKI may provide public keys for a small user community, for a single enterprise, or possibly for a nation. A properly working PKI provides the facilities to create certificates and to reliably validate them. We've mentioned at least two PKIs already: the National Security Agency created a PKI to support the STU III secure telephones, and RSA Data Security created a PKI to support SSL-based Web security.

End users need authentic copies of certificate authorities' keys in order to validate certificates off-line. The most convenient approach might seem to involve a single certificate authority whose key resides in all end-user software, much like Netscape's original approach. However, this doesn't scale well as the number of certificates grows.

The natural approach is to delegate certification. We make this work by issuing certificates to these certificate authorities and signing those certificates with the key belonging to a higher level certificate authority. End-user software can then validate keys from

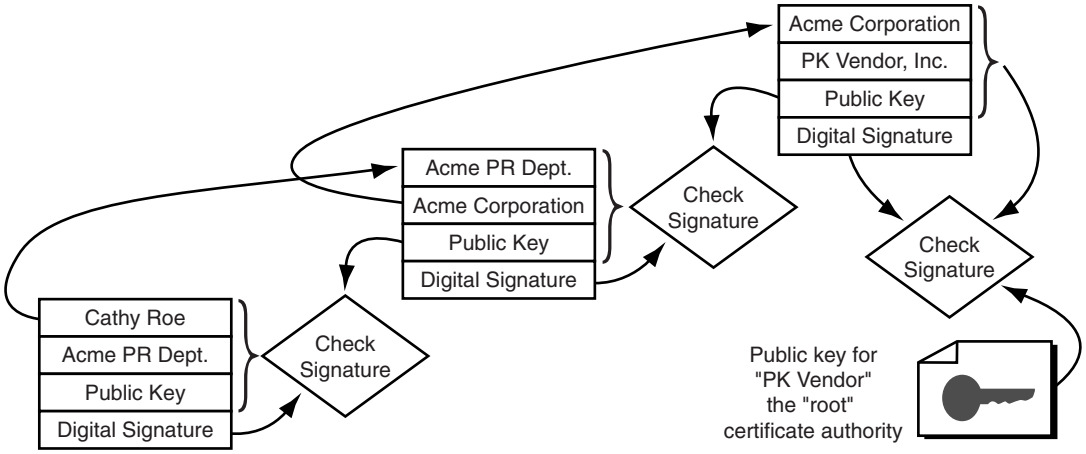

FIGURE 14.9: *Authenticating a certificate chain.* We verify Cathy Roe's certificate by checking its signature with the public key of the "Acme PR Dept" certificate authority. We verify the certificate for the Acme PR Dept by checking its signature with the public key of the "Acme Corporation" authority, and we check that certificate using the key from the authority that issued that certificate. Ultimately, the chain leads to a public key that we already trust, or the authentication fails.

numerous different authorities by validating those authorities' own certificates.

This yields the notion of a *certificate chain*. Each certificate in the chain validates the next earlier certificate, until we reach a "root" authority whose public key we already have. In Figure 14.9, John gets a copy of Cathy Roe's certificate. Note that each certificate has four important entries: the owner's name, the authority that created the certificate, the public key, and the certificate's signature. So John can validate a certificate by retrieving the public key for the authority that created that certificate.

He starts with Cathy's certificate, which was signed by the "Acme PR Department." He retrieves the certificate for the department and uses it to validate the signature on Cathy's certificate. Now he knows that Cathy's certificate is genuine, but only if the PR Department's certificate is also genuine. The department's certificate was issued by Acme Corporation itself, so John must retrieve Acme's corporate certificate to verify the department's certificate. Once John has verified the department's certificate, he still needs to verify the corporation's certificate. That certificate was signed by a commercial certificate authority named "PK Vendor." John already has a

reliable copy of the PK Vendor public key, so he can use it to verify Acme's corporate certificate. Now that all the certificates in the chain have been validated, John can believe that Cathy's certificate is valid.

Originally, it seemed that the world of computing would probably be incorporated into a single, universal PKI. There would be a single "root" certificate authority that all chains led to. This did not work out well in practice, and today there are many "roots" that certificate chains might lead to. Applications like browsers contain lists of authorities in order to deal with this situation. A slightly different strategy is for different authorities to cross-certify each others' keys so that chains might still lead to other roots. These alternatives are discussed below.

CENTRALIZED HIERARCHY

A single, centralized hierarchy would seem to provide a straightforward PKI. There would be a single "root" authority that would sign all certificates for other authorities, and all certificate chains would lead to this root. Software vendors could embed the root authority's key in all software, making it easy to verify any certificate.

Such a hierarchy was proposed for the Internet as part of the Privacy Enhanced Mail (PEM) specification. The root authority would be called the "Internet Policy Registration Authority" and it would issue certificates to "Policy Certification Authorities" (PCAs). Each PCA would have its own policy for issuing certificates, somewhat akin to a certification practices statement. PCAs would in turn issue certificates to certificate authorities, who in turn could certify lower-level authorities or end users.

The PCAs were included because it was obvious that certificates would be issued for different purposes, and there needed to be a systematic way of distinguishing between them. In theory, user software could detect which PCA appeared in a particular certificate chain, and could accept or reject a certificate on that basis. *see Note 9.*

In practice, the centralized hierarchy did not prove practical. Part of the problem was cost. The arrangement forced everyone to purchase a certificate from a PCA, even for prototypes and small-scale tests. Another problem was that no organization had enough good-

will in the community to be able to hold the Internet root key without arousing suspicions from some quarter.

AUTHORITY LISTS

Although browsers started off with all certificates hanging from a single tree, this changed as newer browsers were released. Today, browsers contain a list of certificate authorities whose certificates they recognize. The Netscape Navigator calls this the list of "certificate signer certificates." Figure 14.10 shows part of Netscape's list, as distributed in mid-2000. At that time, the list contained 70 separate keys (13 from Verisign alone). Typically, an authority's key is itself stored in a certificate signed by the authority's own key.

The list structure allows the browser to add new keys as well as recognize the keys already installed. This can happen somewhat automatically when a user visits a site whose certificate can't be authenticated. For example, let's assume that John is browsing a site belonging to a new e-commerce vendor, and the site's certificate was signed by an unfamiliar certificate authority. The SSL protocol generally provides the entire certificate chain when sending certificates to a browser. If the browser doesn't already recognize the highest authority's certificate in that chain, the browser can ask John if it should add that authority to its list.

It is typically a multistep process to add an authority to the browser's list. First, the browser displays the authority's certificate, and then it asks John if and how it should accept the certificate. John can choose to reject the authority entirely, although that will prevent him from being able to open a secure connection with that site. More often, John will add the authority to the browser's built-in list. If John connects to other sites whose certificates were issued by that authority, his browser will recognize the certificates automatically. However, John can also decide to accept the authority only for this particular connection and to discard it afterwards.

Unfortunately, this convenient process isn't necessarily a safe one. It makes things very easy for an attacker who wants to present bogus certificates to a browser. All the attacker has to do is create a public key pair for a certificate authority, give it an honest-sounding name, and sign certificates with that authority's key. The attacker

A-88

could even create a legitimate site or two whose certificates are signed by the bogus authority. Once John or some other unsuspecting person visits one of those sites, his browser will try to install the bogus authority's certificate. If John adds the authority to his browser's list, then the bogus site can trick John into accepting forged certificates that claim to be from legitimate sites.

For example, the attacker could create a series of bogus sites that masquerade as well known commercial sites like Amazon, Barnes and Noble, or Wells Fargo. When John downloads the site's certificate, signed by the bogus authority, his browser will treat the certificate as genuine. This is because John's browser has essentially been "infected" by the bogus authority's certificate. Of course, it's probably not practical for an attacker to masquerade as a large, complicated commercial site, but the attacker could use a bogus certificate to implement an MIM attack, like we described in Section 14.1.

This attack essentially brings us to the limit of what we can do with indirection. We can only establish confidence in a piece of digital data if we can authenticate it against data we already trust. If we receive a completely new public key, we have no way of establishing any certainty about that key's real identity.

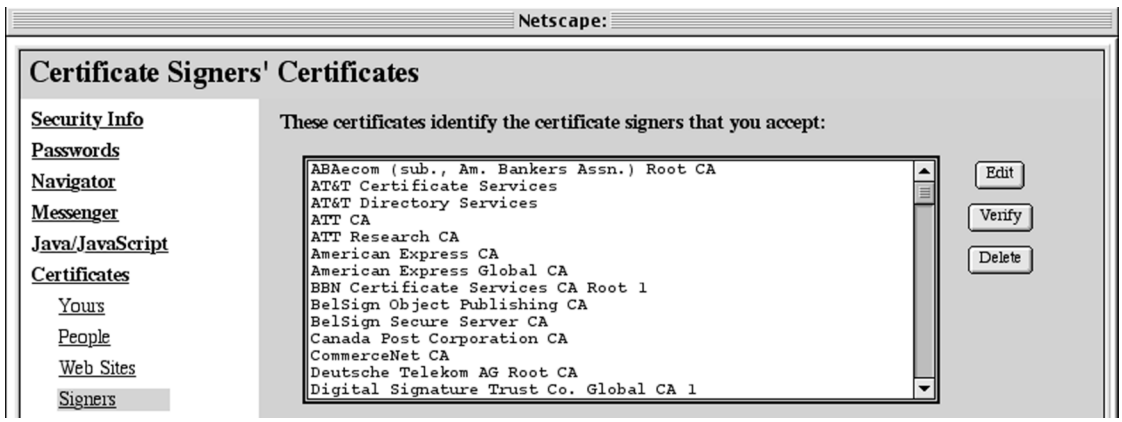

FIGURE 14.10: *List of certificate authorities in the Netscape browser.* This shows part of the list of certificate authorities whose certificates will be automatically authenticated by the browser if received while processing an SSL connection or other secure operation. The user may examine or delete authorities as desired by using the buttons on the right.

CROSS-CERTIFICATION

Although certificate chains were originally intended to support hierarchies of authorities, they also can allow authorities to certify each others' public keys. This allows us to use a smaller number of authority keys to authenticate a broader range of certificates. Moreover, this reduces the risk, since the authority is essentially vouching for the other authority's legitimacy by signing its key.

The notion of cross-certification was introduced in the Distributed Authentication Security Service (DASS) developed by Digital Equipment Corporation. DASS identified specific certificate authorities that could issue certificates containing other authorities' public keys, even though those authorities weren't strictly subordinates. The result was that workstations could get by with a small number of keys belonging to authorities that were allowed to cross-certify other authorities. *see Note 10.*

Although cross-certification holds promise, it also poses some problems. In particular, it's not clear what liability an authority takes on when certifying another authority's keys. If an authority certifies keys from a subverted authority, then this subverts the entire public key system. Moreover, there's the question of exactly what is being implied by cross-certification. Perhaps an authority would like to certify another's keys for some purposes but not for others. Various researchers have been looking at this problem in the context of "trust management systems."

14.5 PERSONAL CERTIFICATION

The world of certification and authorities does not appeal to everyone in the potential user population. Many people do not want a third party to produce a costly cryptographic credential when they themselves have the necessary software on hand. The complexity of large-scale PKI systems does little to promote grass-roots applications of public key technology. However, the benefits of public key technology don't all require a large-scale infrastructure. Pretty Good Privacy (PGP) is a pioneering encryption package that uses public keys without a sophisticated infrastructure. Instead of relying on certificate authorities to serve as independent, trusted third parties, PGP lets people certify their own keys, and the keys of their friends, colleagues, and acquaintances. *see Note 11.*

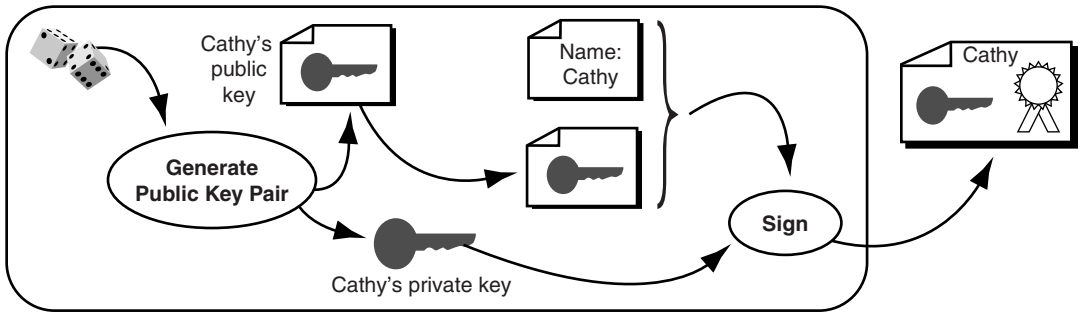

FIGURE 14.11: *A self-signed public key certificate.* Cathy generates a public key pair and generates a self-signed certificate by using her own private key to sign her own certificate. This is a practical approach in environments that establish identity and reputations over time.

The PGP approach can provide a lot of confidence in the authenticity of keys in two particular cases, described below. In the first case, a person's self-signed key can establish authenticity through the "reputation" associated with the key's use in public. In the second case, a key can establish authenticity because other people have vouched for its authenticity.

CERTIFIED BY REPUTATION

The Internet has created a unique public forum in which people can discuss ideas, argue points, and establish reputations based entirely on the strength of one's public statements. Individuals become known and respected for their words, and people refer to this level of respect as *reputation capital*. Most people participating in Internet discussions have no particular public reputation, but many still establish reputation capital based on their performance in the discussions.

A number of people who participate in these discussions will place digital signatures on their messages. The public key for those signatures resides in a *self-signed certificate*. Such a certificate contains the person's name as used on the Internet and is signed with the public key embedded within the certificate. This is illustrated in Figure 14.11.

We can see how this works if we look at what happens when Cathy Roe develops reputation capital based on a series of signed messages. Cathy creates a public key pair and publishes a pubic key

certificate that she signs with that same public key. Anyone who gets a copy of that certificate can verify that the key contained therein was used to sign the certificate, and that the certificate contains Cathy's name. As Cathy publishes signed messages, anyone can verify them with the public key in the certificate she has published.

As Cathy's messages are read by other participants, they develop an understanding of Cathy's identity, and they establish certain assumptions concerning her knowledge and capabilities. This understanding of Cathy is based on the messages that she has signed with her public key. Everyone can verify that all messages were generated by a person who holds that particular public key, and that the person claims in the certificate to be Cathy Roe.

Now, what happens if someone else generates a self-signed certificate claiming to be Cathy Roe? Let's assume that Henry has decided to try to ruin Cathy's reputation by forging messages in her name. First, he creates a self-signed certificate containing Cathy's name. Then he publishes the certificate and uses it to sign messages claiming to be from Cathy.

While some readers might be tricked by Henry's stratagem, Cathy can respond by publicly denouncing the certificate in a message signed with her own key. Anyone can use her older certificate to authenticate her message. Henry won't be able to sign messages with that old key; all of his messages must be signed with his bogus key. So people can use the self-signed certificate to reliably distinguish between Henry's forgeries and Cathy's legitimate messages.

CERTIFIED BY A WEB OF TRUST

In a large-scale PKI, the only entities that can sign certificates are the certificate authorities. In PGP, anyone can sign any certificate. In fact, certificates can contain numerous signatures. The signatures serve roughly the same purpose as a certificate authority's signature: it attests to the fact that the given name and key go together. But PGP relies on individuals to certify each others' keys instead of relying on formally established third-party authorities.

In PGP, as in other public key systems, we authenticate a key by checking the certificate's signature against a signature we trust. If John gets a copy of Cathy's certificate directly from Cathy, then he's

going to treat it as a genuine certificate. If he gets a copy of Tim's certificate via e-mail, and Cathy has signed Tim's certificate, then he'll probably believe that Tim's certificate is genuine. Now what happens if John gets a certificate from someone named Bob that has been signed by Tim? John essentially has a certificate chain leading from Bob to Tim and then to Cathy, which allows him to authenticate Bob's certificate. This type of chain is called a *web of trust*.

14.6 Certificate Revocation

Indirect authentication is a very convenient thing to be able to do, but we pay for that convenience when we try to revoke a certificate. If someone loses their private key to an attacker, we have no easy way to revoke his or her certificate. We have no way of knowing how many copies of that certificate might reside in various browsers and workstations. Since the software uses the certificate to authenticate people off-line, the software won't necessarily find out that the certificate shouldn't be used.

A-89

There are three general strategies suggested to address this problem. First, there is the *certificate revocation list* (CRL) which publishes a list of all certificates that should be revoked. Second, there is the notion of checking certificates on-line. Third, there is a notion of using short-term certificates to minimize the risk of stolen credentials. These alternatives are described below.

CERTIFICATE REVOCATION LIST

In the world of formal certificate authorities, the revocation list is cited as the standard approach to revocation. Every authority is responsible for producing a list of certificates that should be revoked, and for providing that list to end users on a regular basis. Every certificate contains an expiration date, and a revoked certificate remains on the revocation list until it expires. Unfortunately, revocation lists are often one of the last pieces of PKI technology to be implemented, and computers may be vulnerable to bogus certificates under those circumstances.

D-87

see Note 12.

The principal benefit of the revocation list strategy is that it lets us do off-line certificate checking. Major credit cards used a similar strategy to handle revoked credit cards in the 1970s and 1980s, until on-line credit checking became common. Each card company

would periodically publish a list of revoked credit cards and distribute the list to all merchants who accepted their card. In some cases, the companies offered a bounty for every revoked card a merchant might find, giving merchants an extra incentive to check the cards.

The credit card list had an obvious shortcoming: a thief could make heavy use of a card between the time it was stolen and the time the next list of stolen cards was published. The same problem applies to CRLs: the theft may have occurred since the last CRL was issued, or the end user might not have been able to retrieve the latest CRL for some reason. This uncovers one of the places where off-line authentication systems are brittle: an attacker can provide "most" of the evidence to make a key look legitimate, and some end users may be satisfied that authentication has come "close enough" even though partial evidence may well be entirely bogus.

Public key products don't implement CRLs because they represent a complicated mechanism that is rarely used. Early versions of Web browsers ignored CRLs entirely, though recent versions of major browsers do support them. Unfortunately, this doesn't guarantee that CRLs will in fact revoke bogus certificates, since the mechanism depends on detailed technical cooperation between certificate authorities and software product vendors. Vendors might not even know there is a problem until they actually need to use the CRL mechanism.

For example, when Verisign issued a bogus certificate containing Microsoft's name, it uncovered a gap in their CRL processing that prevented the CRL from being used. Microsoft's software relied on a pointer in the certificate to locate the CRL, but Verisign's certificates didn't contain that pointer. So Microsoft issued a specific software patch that installed Verisign's CRL on each computer and forced the certificate-checking software to examine that CRL, ensuring the certificates would be revoked. *see Note 13.*

ON-LINE REVOCATION

This approach is more akin to what happens today with almost all credit cards: the merchant checks the card's status on-line before accepting it for payment. This approach isn't as reliable as true off-line authentication, since failures in the communication system or other components can cause the transaction to fail. The advantage

D-88

of on-line revocation is that it ensures completely accurate information. *see Note 14.*

In the public key certificate environment, an end user would verify a certificate on-line by contacting the certificate authority. The authority could verify that the specified certificate has not yet been revoked. However, not all certificate authorities support real-time certificate verification; at most, host computers would be able to contact the authority for its most recent CRL. In fact, not all public key applications support CRLs, so there are environments in which it isn't really possible to revoke a certificate.

TIMELY CERTIFICATION

Ron Rivest, coinventor of RSA, has proposed a different approach to the revocation problem which relies on recently issued certificates instead of long-lived certificates. Rivest believes that the risk falls most heavily on the host that accepts a certificate, and that the way to control that risk is to control how recently the certificate has been validated. Depending on the application, a host might want the certificate validated fairly recently. From this point of view, the CRL strategy is wrong because the certificate authority, not the recipient, determines how fresh a certificate's validity data can be. *see Note 15.*

Rivest proposes that it should be up to the signer to provide evidence that a certificate is valid, and that such evidence should be a certificate with a recent creation date. Certificate authorities should be able to issue updated copies of valid certificates on demand. If John needs to provide a certificate to his bank in order to perform a transaction, he first retrieves a new copy of his certificate, which will contain a recent time stamp. The bank will be able to look at that certificate and tell from the time stamp that it is probably a valid certificate, since it was issued recently.  *D-89*

If Henry manages to steal John's private key, John can tell the certificate authority about the compromise. When Henry, or anyone else, asks the authority for an up-to-date copy of John's certificate containing the compromised key, the authority will report that the certificate has been revoked. Meanwhile, John can file a new public key with the authority and start distributing certificates with his name and a new key.

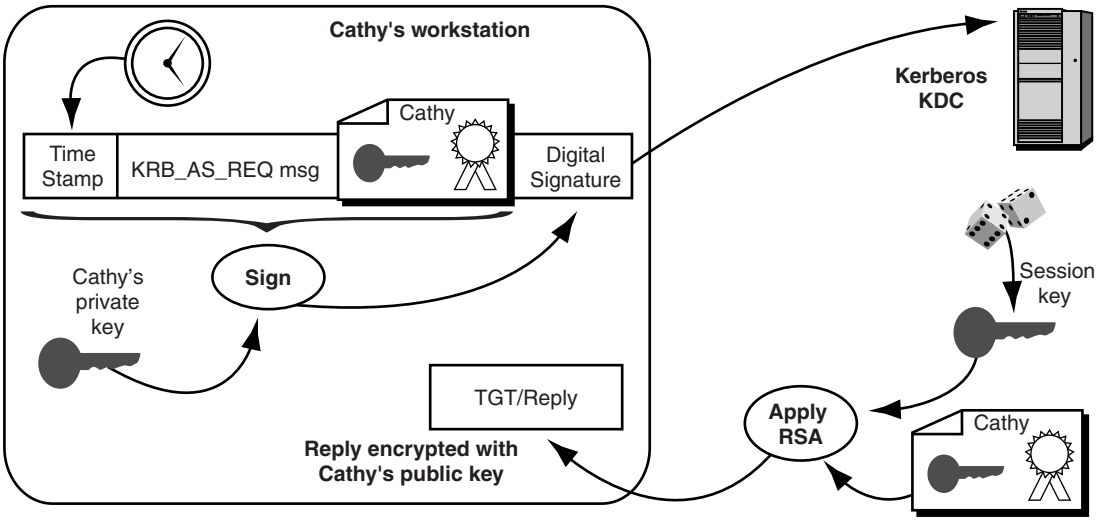

FIGURE 14.12: *Kerberos preauthentication with public keys.* This approach, called PKINIT, uses Cathy's public key certificate instead of a memorized password to authenticate Cathy to the Kerberos KDC. The reply uses a public key to encrypt the TGT's session key.

14.7 CERTIFICATES WITH KERBEROS

In Chapter 12, we briefly noted that there are ways to integrate public key cryptography with Kerberos. While these techniques don't necessarily convert a Kerberos environment into a true off-line authentication architecture (after all, the whole process will always depend on the presence of a Kerberos KDC), it does eliminate the need for a shared secret based on a reusable password.

Figure 14.12 illustrates the technique called PKINIT, for "public key initialization." PKINIT uses Cathy's public key pair in a special version of the preauthentication process described in Section 12.3. Cathy logs on to her workstation and provides her private key. The workstation contacts the KDC, sending a preauthenticated request for a TGT. The request contains the usual information along with a copy of Cathy's public key certificate, and the whole request is digitally signed with her private key.

see Note 16.

Upon receipt, the KDC first tries to validate Cathy's certificate. It must be issued by an authority recognized by the KDC or it will be rejected. Next, the KDC generates the TGT for Cathy and encrypts the corresponding session key with Cathy's public key. The entire

response is then signed with the KDC's own private key so that Cathy can verify its integrity upon receipt. Once Cathy decrypts the session key, she can use it with the TGT to authenticate herself to other servers. She should not need her private key again until the next time she logs on.

This is not the only way PKINIT can work. It can also use temporary Diffie-Hellman keys in order to generate a shared secret. In that case, Cathy's preauthentication request contains a temporary Diffie-Hellman key. Cathy must still sign the request with a separate key, and must usually provide a copy of the certificate for her signature key. The KDC then verifies her certificate and the signature on the preauthentication data before generating a TGT based on a Diffie-Hellman shared secret.

An important property of PKINIT is that it can eliminate the KDC's role in authenticating users. Instead, the KDC relies entirely on user certificates. The authority that issues certificates will bear the responsibility for verifying user identities. Once the KDC accepts a certificate, it has essentially authenticated the corresponding user. For this reason, the KDC must scrupulously check certificates and only issue tickets if the certificate is one that the KDC can honestly recognize.

Windows 2000 uses PKINIT to integrate public keys with its Kerberos authentication environment. The Kerberos TGT request contains a copy of the user's certificate and is signed with the user's private key. The Windows 2000 KDC confirms that the certificate is valid and that it was issued by an authority recognized by that KDC. Then the KDC verifies the preauthentication time stamp and digital signature. *see Note 17.*

Once the KDC has validated the preauthentication data, it constructs a TGT that includes Windows 2000–specific authorization data, primarily consisting of security identifiers for the user and groups the user belongs to. Then the KDC encrypts the response with the public key in the user's certificate and signs it with the KDC's own private key. Upon receipt, the user's system decrypts and verifies the response, and uses the TGT in accordance with the Kerberos protocols to request access to other servers.

14.8 Summary Tables

TABLE 14.1: *Attack Summary*

Attack 	Security Problem	Prevalence	Attack Description
A-84. Public key forgery	Recover or modify hidden information	Trivial	If a recipient accepts an unauthenticated public key, the attacker simply substitutes his own key for the right one.
A-85. Man in the middle	Recover or modify hidden information	Sophisticated	Attacker substitutes own public key for another, and reencrypts messages between two entities
A-86. Bogus name on certificate	Masquerade as someone else	Trivial	Attacker puts the victim's name on the application for a public key certificate.
A-87. Substitute certificate	Masquerade as someone else	Trivial	Attacker uses a legitimate certificate to implement SSL on a bogus site that is masquerading as a different site
A-88. Bogus certificate authority	Masquerade as someone else	Common	Attacker uses a bogus certificate authority to create bogus certificates, and induces browsers to accept his authority key
A-89. Exploit private key	Masquerade as someone else	Software	Attacker relies on off-line authentication to exploit a stolen private key

TABLE 14.2: *Defense Summary*

Defense	Foils Attacks	Description
D-84. Public key certificates	A-84. Public key forgery A-85. Man in the middle	Publish the assignment of a given public key to a given owner, and sign this publication with a trustworthy digital signature
D-85. Key ownership requirement	A-86. Bogus name on certificate	Issue certificates only to the person who owns the requested name
D-86. Validate certificate's host name	A-87. Substitute certificate	Compare the name on the certificate against the name of the host computer on the SSL connection
D-87. Certificate revocation list	A-89. Exploit private key	Issue a periodic list of all certificates that have been revoked
D-88. On-line certificate revocation	A-89. Exploit private key	Provide a mechanism to query an authority to verify that a given certificate has not been revoked
D-89. Timely certification	A-89. Exploit private key	Require that all certificates be issued recently, and provide a mechanism so that authorities can issue such certificates if the certificate has not been revoked

RESIDUAL THREATS

A-88. Bogus certificate authority—The user interface of typical modern browsers provides no reasonable protection against this attack, since there is no way to tell the difference between a new key from a legitimate authority and a key from a bogus authority.