why would I worry about protecting it during the brief period in which it is being used by the CPU? After all, if someone can get to my volatile memory, I probably have bigger problems than protecting this little bit of data, right?" Not really.

Various independent researchers have demonstrated effective side-channel attacks against memory shared by multiple processes. A *side-channel attack* exploits information that is being leaked by a cryptosystem. As we will see in our discussion of cryptology in Chapter 8, a cryptosystem can be thought of as connecting two channels: a plaintext channel and an encrypted one. A *side channel* is any information flow that is the electronic by-product of this process. As an illustration of this, imagine yourself being transported in the windowless back of a van. You have no way of knowing where you are going, but you can infer some aspects of the route by feeling the centrifugal force when the van makes a turn or follows a curve. You could also pay attention to the engine noise or the pressure in your ears as you climb or descend hills. These are all side channels. Similarly, if you are trying to recover the secret keys used to encrypt data, you could pay attention to how much power is being consumed by the CPU or how long it takes for other processes to read and write from memory. Researchers have been able to recover 2,048-bit keys from shared systems in this manner.

But the threats are not limited to cryptosystems alone. The infamous Heartbleed security bug of 2014 demonstrated how failing to check the boundaries of requests to read from memory could expose information from one process to others running on the same system. In that bug, the main issue was that anyone communicating with the server could request an arbitrarily long "heartbeat" message from it. Heartbeat messages are typically short strings that let the other end know that an endpoint is still there and wanting to communicate. The developers of the library being used for this never imagined that someone would ask for a string that was hundreds of characters in length. The attackers, however, did think of this and in fact were able to access crypto keys and other sensitive data belonging to other users.

More recently, the Meltdown, Spectre, and BranchScope attacks that came to light in 2018 show how a clever attacker can exploit hardware features in most modern CPUs. Meltdown, which affects Intel and ARM microprocessors, works by exploiting the manner in which memory mapping occurs. Since cache memory is a lot faster than main memory, most modern CPUs include ways to keep frequently used data in the faster cache. Spectre and BranchScope, on the other hand, take advantage of a feature called speculative execution, which is meant to improve the performance of a process by guessing what future instructions will be based on data available in the present. All three implement side-channel attacks to go after data in use.

So, how do we protect our data in use? The short answer is, we can't, at least for now. We can get close, however, by ensuring that our systems decrypt data at the very last possible moment, ideally as it gets loaded into the CPU registers, and encrypt it as it leaves those registers. This approach means that the data is encrypted even in memory, but it is an expensive approach that requires a cryptographic co-processor. You may encounter it if you work with systems that require extremely high security but are in places where adversaries can put their hands on them, such as automated teller machines (ATMs) and military weapon systems.

A promising approach, which is not quite ready for prime time, is called *homomorphic encryption*. This is a family of encryption algorithms that allows certain operations on the encrypted data. Imagine that you have a set of numbers that you protect with homomorphic encryption and give that set to me for processing. I could then perform certain operations on the numbers, such as common arithmetic ones like addition and multiplication, without decrypting them. I add the encrypted numbers together and send the sum back to you. When you decrypt them, you get a number that is the sum of the original set before encryption. If this is making your head hurt a little bit, don't worry. We're still a long ways from making this technology practical.

Standards

As we discussed in Chapter 1, *standards* are mandatory activities, actions, or rules that are formally documented and enforced within an organization. Asset security standards can be expensive in terms of both financial and opportunity costs, so we must select them carefully. This is where classification and controls come together. Since we already know the relative value of our data and other information assets and we understand many of the security controls we can apply to them, we can make cost-effective decisions about how to protect them. These decisions get codified as information asset protection standards.

The most important concept to remember when selecting information asset protection standards is to balance the value of the information with the cost of protecting it. Asset inventories and classification standards will help you determine the right security controls.

Scoping and Tailoring

One way to go about selecting standards that make sense for your organization is to adapt an existing standard (perhaps belonging to another organization) to your specific situation. *Scoping* is the process of taking a broader standard and trimming out the irrelevant or otherwise unwanted parts. For example, suppose your company is acquired by another company and you are asked to rewrite some of your company's standards based on the ones the parent company uses. That company allows employees to bring their own devices to work, but that is not permitted in your company. You remove those sections from their standard and scope it down to your size. *Tailoring*, on the other hand, is when you make changes to specific provisions so they better address your requirements. Suppose your new parent company uses a particular solution for centralized backup management that is different from the solution your company has been using. As you modify that part of the standard to account for your platform, you are tailoring it to your needs.

Data Protection Methods

As we have seen, data can exist in many forms and places. Even data in motion and data in use can be temporarily stored or cached on devices throughout our systems. Given the abundance of data in the typical enterprise, we have to narrow the scope of our data protection to the data that truly matters. A *digital asset* is anything that exists in digital form, has intrinsic value to the organization, and to which access should be restricted in some way. Since these assets are digital, we must also concern ourselves with the storage media on which they reside. These assets and storage media require a variety of controls to ensure data is properly preserved and that its integrity, confidentiality, and availability are not compromised. For the purposes of this discussion, "storage media" may include both electronic (disk, optical discs, tape, flash devices such as USB "thumb drives," and so on) and nonelectronic (paper) forms of information.

The operational controls that pertain to digital assets come in many flavors. The first are controls that prevent unauthorized access (protect confidentiality), which, as usual, can be physical, administrative, and technical. If the company's backup tapes are to be properly protected from unauthorized access, they must be stored in a place where only authorized people have access to them, which could be in a locked server room or an offsite facility. If storage media needs to be protected from environmental issues such as humidity, heat, cold, fire, and natural disasters (to maintain availability), the media should be kept in a fireproof safe in a regulated environment or in an offsite facility that controls the environment, so it is hospitable to data processing components.

Companies may have a digital asset library with a librarian in charge of protecting its resources. If so, most or all of the responsibilities described in this chapter for the protection of the confidentiality, integrity, and availability of media fall to the librarian. Users may be required to check out specific resources from the library, instead of having the resources readily available for anyone to access them. This is common when the library includes licensed software. It provides an accounting (audit log) of uses of assets, which can help in demonstrating due diligence in complying with license agreements and in protecting confidential information (such as PII, financial/credit card information, and PHI) in libraries containing those types of data.

Storage media should be clearly marked and logged, its integrity should be verified, and it should be properly erased of data when no longer needed. After a large investment is made to secure a network and its components, a common mistake is to replace old computers, along with their hard drives and other magnetic storage media, and ship the obsolete equipment out the back door along with all the data the company just spent so much time and money securing. This puts the information on the obsolete equipment and media at risk of disclosure and violates legal, regulatory, and ethical obligations of the company. Thus, overwriting (see Figure 6-2) and secure overwriting algorithms are required. Whenever storage media containing highly sensitive information cannot be cleared or purged, physical destruction must take place.

When storage media is erased (*cleared* of its contents), it is said to be *sanitized*. In military/government classified systems terms, this means erasing information so it is not readily retrievable using routine operating system commands or commercially available forensic/data recovery software. Clearing is acceptable when storage media will be reused in the same physical environment for the same purposes (in the same compartment of compartmentalized information security) by people with the same access levels for that compartment.

Not all clearing/purging methods are applicable to all storage media—for example, optical media is not susceptible to degaussing, and overwriting may not be effective when

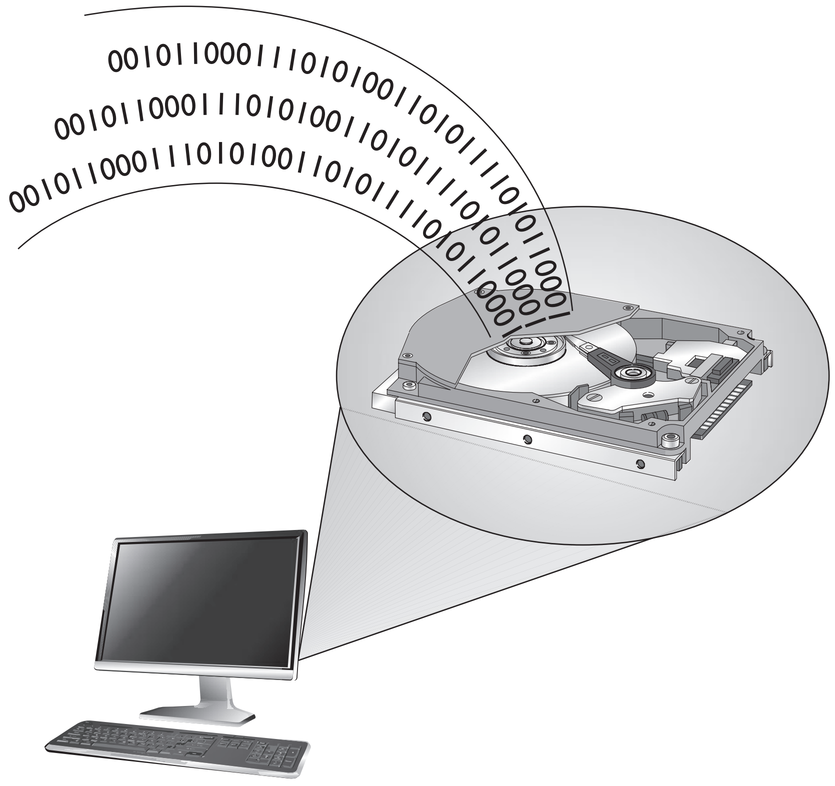


Figure 6-2 Overwriting storage media to protect sensitive data

dealing with solid-state devices. The degree to which information may be recoverable by a sufficiently motivated and capable adversary must not be underestimated or guessed at in ignorance. For the highest-value digital assets, and for all data regulated by government or military classification rules, read and follow the rules and standards.

The guiding principle for deciding what is the necessary method (and cost) of data erasure is to ensure that the enemies' cost of recovering the data exceeds the value of the data. "Sink the company" (or "sink the country") information has value that is so high that the destruction of the storage devices, which involves both the cost of the destruction and the total loss of any potential reusable value of the storage media, is justified. For most other categories of information, multiple or simple overwriting is sufficient. Each organization must evaluate the value of its digital assets and then choose the appropriate erasure/disposal method.

Chapter 5 discussed methods for secure clearing, purging, and destruction of electronic media. Other forms of information, such as paper, microfilm, and microfiche, also require secure disposal. "Dumpster diving" is the practice of searching through trash at homes and businesses to find valuable information that was simply thrown away without being first securely destroyed through shredding or burning.

Atoms and Data

A device that performs degaussing generates a coercive magnetic force that reduces the magnetic flux density of the storage media to zero. This magnetic force is what properly erases data from media. Data is stored on magnetic media by the representation of the polarization of the atoms. Degaussing changes this polarization (magnetic alignment) by using a type of large magnet to bring it back to its original flux (magnetic alignment).

Digital Asset Management

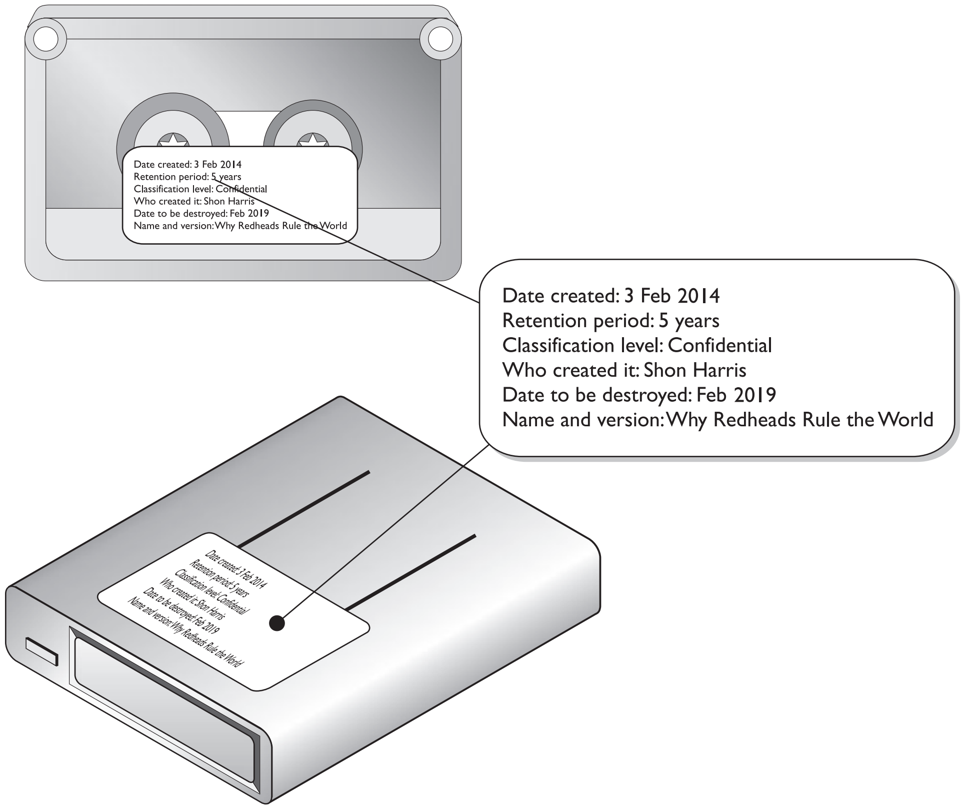
Digital asset management is the process by which organizations ensure their digital assets are properly stored, well protected, and easily available to authorized users. While specific implementations vary, they typically involve the following tasks:

- **Tracking** (audit logging) who has custody of each digital asset at any given moment. This creates the same kind of audit trail as any audit logging activity—to allow an investigation to determine where information was at any given time, who had it, and, for particularly sensitive information, why they accessed it. This enables an investigator to focus efforts on particular people, places, and times if a breach is suspected or known to have happened.
- **Effectively implementing access controls** to restrict who can access each asset to only those people defined by its owner and to enforce the appropriate security measures based on the classification of the digital asset. Certain types of media, due to their sensitivity and storage media, may require special handling. As an example, classified government information may require that the asset may only be removed from the library or its usual storage place under physical guard, and even then may not be removed from the building. Access controls will include *physical* (locked doors, drawers, cabinets, or safes), *technical* (access and authorization control of any automated system for retrieving contents of information in the library), and *administrative* (the actual rules for who is supposed to do what to each piece of information). Finally, the digital media may need to change format, as in printing electronic data to paper, and still needs to be protected at the necessary level, no matter what format it is in. Procedures must include how to continue to provide the appropriate protection. For example, sensitive material that is to be mailed should be sent in a sealable inner envelope and only via a courier service.
- **Tracking the number and location of backup versions** (both onsite and offsite). This is necessary to ensure proper disposal of information when the information reaches the end of its lifespan, to account for the location

and accessibility of information during audits, and to find a backup copy of information if the primary source of the information is lost or damaged.

- **Documenting the history of changes.** For example, when a particular version of a software application kept in the library has been deemed obsolete, this fact must be recorded so the obsolete version of the application is not used unless that particular obsolete version is required. Even once no possible need for the actual asset remains, retaining a log of the former existence and the time and method of its deletion may be useful to demonstrate due diligence.
- **Ensuring environmental conditions do not endanger storage media.** If you store digital assets on local storage media, each media type may be susceptible to damage from one or more environmental influences. For example, all types are susceptible to fire, and most are susceptible to liquids, smoke, and dust. Magnetic storage media are susceptible to strong magnetic fields. Magnetic and optical media are susceptible to variations in temperature and humidity. A media library and any other space where reference copies of information are stored must be physically built so all types of media will be kept within their environmental parameters, and the environment must be monitored to ensure conditions do not range outside of those parameters. Media libraries are particularly useful when large amounts of information must be stored and physically/environmentally protected so that the high cost of environmental control and media management may be centralized in a small number of physical locations and so that cost is spread out over the large number of items stored in the library.
- **Inventorying digital assets** to detect if any asset has been lost or improperly changed. This can reduce the amount of damage a violation of the other protection responsibilities could cause by detecting such violations sooner rather than later, and is a necessary part of the digital asset management life cycle by which the controls in place are verified as being sufficient.
- **Carrying out secure disposal activities.** Disposal activities usually begin at the point at which the information is no longer valuable and becomes a potential liability. Secure disposal of media/information can add significant cost to media management. Knowing that only a certain percentage of the information must be securely erased at the end of its life may significantly reduce the long-term operating costs of the company. Similarly, knowing that certain information must be disposed of securely can reduce the possibility of a storage device being simply thrown in a dumpster and then found by someone who publicly embarrasses or blackmails the company over the data security breach represented by that inappropriate disposal of the information. The business must take into account the useful lifetime of the information to the business, legal, and regulatory restrictions and, conversely, the requirements for retention and archiving when making these decisions. If a law or regulation requires the information to be kept beyond its normally useful lifetime for the business, then disposition may involve *archiving*—moving the information from the ready (and possibly more expensive) accessibility of a library to a long-term stable and (with some effort) retrievable format that has lower storage costs.

- **Internal and external labeling** of each piece of asset in the library should include
  - Date created
  - Retention period
  - Classification level
  - Who created it
  - Date to be destroyed
  - Name and version


Date created: 3 Feb 2014
Retention period: 5 years
Classification level: Confidential
Who created it: Shon Harris
Date to be destroyed: Feb 2019
Name and version: Why Redheads Rule the World

Digital Rights Management

So, how can we protect our digital assets when they leave our organizations? For example, if you share a sensitive file or software system with a customer, how can you ensure that only authorized users gain access to it? *Digital Rights Management (DRM)* refers to a set of technologies that is applied to controlling access to copyrighted data. The technologies themselves don't need to be developed exclusively for this purpose. It is the use of a technology that makes it DRM, not its design. In fact, many of the DRM technologies in use today are standard cryptographic ones. For example, when you buy a Software as a Service

(SaaS) license for, say, Office 365, Microsoft uses standard user authentication and authorization technologies to ensure that you only install and run the allowed number of copies of the software. Without these checks during the installation (and periodically thereafter), most of the features will stop working after a period of time. A potential problem with this approach is that the end-user device may not have Internet connectivity.

An approach to DRM that does not require Internet connectivity is the use of product keys. When you install your application, the key you enter is checked against a proprietary algorithm and, if it matches, the installation is activated. It might be tempting to equate this approach to symmetric key encryption, but in reality, the algorithms employed are not always up to cryptographic standards. Since the user has access to both the key and the executable code of the algorithm, the latter can be reverse-engineered with a bit of effort. This could allow a malicious user to develop a product-key generator with which to effectively bypass DRM. A common way around this threat is to require a one-time online activation of the key.

DRM technologies are also used to protect documents. Adobe, Amazon, and Apple all have their own approaches to limiting the number of copies of an electronic book (e-book) that you can download and read. Another approach to DRM is the use of digital watermarks, which are embedded into the file and can document details such as the owner of the file, the licensee (user), and date of purchase. While watermarks will not stop someone from illegally copying and distributing files, they could help the owner track, identify, and prosecute the perpetrator. An example technique for implementing watermarks is called steganography.

Steganography

Steganography is a method of hiding data in another media type so the very existence of the data is concealed. Common steps are illustrated in Figure 6-3. Only the sender and receiver are supposed to be able to see the message because it is secretly hidden

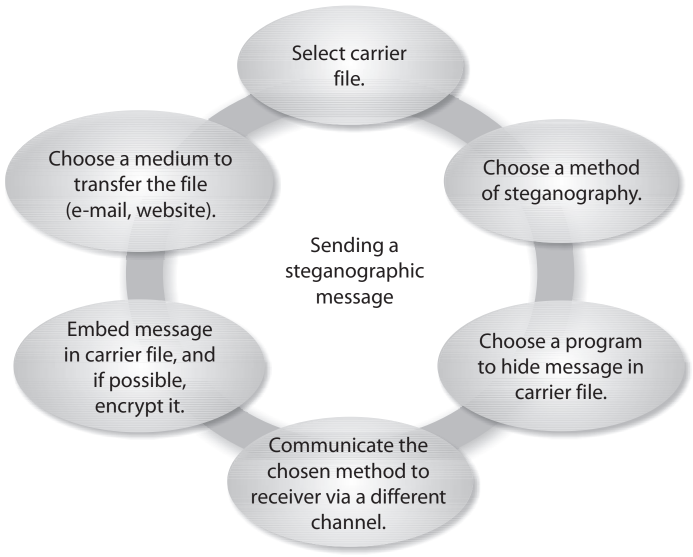

Select carrier file.
Choose a method of steganography.
Choose a program to hide message in carrier file.
Communicate the chosen method to receiver via a different channel.
Embed message in carrier file, and if possible, encrypt it.
Choose a medium to transfer the file (e-mail, website).
Sending a steganographic message

Figure 6-3 Main components of steganography

in a graphic, audio file, document, or other type of media. The message is often just hidden, and not necessarily encrypted. Encrypted messages can draw attention because the encryption tells the bad guy, “This is something sensitive.” A message hidden in a picture of your grandmother would not attract this type of attention, even though the same secret message can be embedded into this image. Steganography is a type of security through obscurity.

Steganography includes the concealment of information within computer files. In digital steganography, electronic communications may include steganographic coding inside of a document file, image file, program, or protocol. Media files are ideal for steganographic transmission because of their large size. As a simple example, a sender might start with an innocuous image file and adjust the color of every 100th pixel to correspond to a letter in the alphabet, a change so subtle that someone not specifically looking for it is unlikely to notice it.

Let’s look at the components that are involved with steganography:

- **Carrier** A signal, data stream, or file that has hidden information (payload) inside of it
- **Stegomedium** The medium in which the information is hidden
- **Payload** The information that is to be concealed and transmitted

A method of embedding the message into some types of media is to use the *least significant bit (LSB)*. Many types of files have some bits that can be modified and not affect the file they are in, which is where secret data can be hidden without altering the file in a visible manner. In the LSB approach, graphics with a high resolution or an audio file that has many different types of sounds (high bit rate) are the most successful for hiding information within. There is commonly no noticeable distortion, and the file is usually not increased to a size that can be detected. A 24-bit bitmap file will have 8 bits representing each of the three color values, which are red, green, and blue. These 8 bits are within each pixel. If we consider just the blue, there will be 2^8 different values of blue. The difference between 11111111 and 11111110 in the value for blue intensity is likely to be undetectable by the human eye. Therefore, the least significant bit can be used for something other than color information.

A digital graphic is just a file that shows different colors and intensities of light. The larger the file, the more bits that can be modified without much notice or distortion.

Data Loss Prevention

Unless we diligently apply the right controls to our data wherever it may be, we should expect that some of it will eventually end up in the wrong hands. In fact, even if we do everything right, the risk of this happening will never be eliminated. *Data loss* is the flow of sensitive information, such as PII, to unauthorized external parties. Leaks of personal information by an organization can cause large financial losses. The costs commonly include

- Investigating the incident and remediating the problem
- Contacting affected individuals to inform them about the incident

- Penalties and fines to regulatory agencies
- Contractual liabilities
- Mitigating expenses (such as free credit monitoring services for affected individuals)
- Direct damages to affected individuals

In addition to financial losses, a company's reputation may be damaged and individuals' identities may be stolen.

The most common cause of data breach for a business is a lack of awareness and discipline among employees—an overwhelming majority of all leaks are the result of negligence. The most common forms of negligent data breaches occur due to the inappropriate removal of information—for instance, from a secure company system to an insecure home computer so that the employee can work from home—or due to simple theft of an insecure laptop or tape from a taxi cab, airport security checkpoint, or shipping box. However, breaches also occur due to negligent uses of technologies that are inappropriate for a particular use—for example, reassigning some type of medium (say, a page frame, disk sector, or magnetic tape) that contained one or more objects to an unrelated purpose without securely ensuring that the media contained no residual data.

It would be too easy to simply blame employees for any inappropriate use of information that results in the information being put at risk, followed by breaches. Employees have a job to do, and their understanding of that job is almost entirely based on what their employer tells them. What an employer tells an employee about the job is not limited to, and may not even primarily be in, the "job description." Instead, it will be in the feedback the employee receives on a day-to-day and year-to-year basis regarding their work. If the company in its routine communications to employees and its recurring training, performance reviews, and salary/bonus processes does not include security awareness, then employees will not understand security to be a part of their job.

The more complex the environment and types of media used, the more communication and training that are required to ensure that the environment is well protected. Further, except in government and military environments, company policies and even awareness training will not stop the most dedicated employees from making the best use of up-to-date consumer technologies, including those technologies not yet integrated into the corporate environment, and even those technologies not yet reasonably secured for the corporate environment or corporate information. Companies must stay aware of new consumer technologies and how employees (wish to) use them in the corporate environment. Just saying "no" will not stop an employee from using, say, a personal smartphone, a USB thumb drive, or webmail to forward corporate data to their home e-mail address in order to work on the data when out of the office. Companies must include in their technical security controls the ability to detect and/or prevent such actions through, for example, computer lockdowns, which prevent writing sensitive data to non-company-owned storage devices, such as USB thumb drives, and e-mailing sensitive information to nonapproved e-mail destinations.

Data loss prevention (DLP) comprises the actions that organizations take to prevent unauthorized external parties from gaining access to sensitive data. That definition has some key terms. First, the data has to be considered *sensitive*, the meaning of which we spent a good chunk of the beginning of this chapter discussing. We can't keep every single datum safely locked away inside our systems, so we focus our attention, efforts, and funds on the truly important data. Second, DLP is concerned with *external parties*. If somebody in the accounting department gains access to internal R&D data, that is a problem, but technically it is not considered a data leak. Finally, the external party gaining access to our sensitive data must be *unauthorized* to do so. If former business partners have some of our sensitive data that they were authorized to get at the time they were employed, then that is not considered a leak either. While this emphasis on semantics may seem excessive, it is necessary to properly approach this tremendous threat to our organizations.

EXAM TIP The terms data loss and data leak are used interchangeably by most security professionals. Technically, however, data loss means we do not know where the data is (e.g., after the theft of a laptop), while data leak means that the confidentiality of the data has been compromised (e.g., when the laptop thief posts the files on the Internet).

The real challenge to DLP is in taking a holistic view of our organization. This perspective must incorporate our people, our processes, and then our information. A common mistake when it comes to DLP is to treat the problem as a technological one. If all we do is buy or develop the latest technology aimed at stopping leaks, we are very likely to leak data. If, on the other hand, we consider DLP a program and not a project, and we pay due attention to our business processes, policies, culture, and people, then we have a good fighting chance at mitigating many or even most of the potential leaks. Ultimately, like everything else concerning information system security, we have to acknowledge that despite our best efforts, we will have bad days. The best we can do is stick to the program and make our bad days less frequent and less bad.

General Approaches to DLP

There is no one-size-fits-all approach to DLP, but there are tried-and-true principles that can be helpful. One important principle is the integration of DLP with our risk management processes. This allows us to balance out the totality of risks we face and favor controls that mitigate those risks in multiple areas simultaneously. Not only is this helpful in making the most of our resources, but it also keeps us from making decisions in one silo with little or no regard to their impacts on other silos. In the sections that follow, we will look at key elements of any approach to DLP.

Data Inventories It is difficult to defend an unknown target. Similarly, it is difficult to prevent the leaking of data of which we are unaware or whose sensitivity is unknown. Some organizations try to protect all their data from leakage, but this is not a good approach. For starters, acquiring the resources required to protect everything is likely cost prohibitive to most organizations. Even if an organization is able to afford this level of protection, it runs a very high risk of violating the privacy of its employees and/or customers by examining every single piece of data in its systems.

A good approach is to find and characterize all the data in your organization before you even look at DLP solutions. The task can seem overwhelming at first, but it helps to prioritize things a bit. You can start off by determining what is the most important kind of data for your organization. A compromise of these assets could lead to direct financial losses or give your competitors an advantage in your sector. Are these healthcare records? Financial records? Product designs? Military plans? Once you figure this out, you can start looking for that data across your servers, workstations, mobile devices, cloud computing platforms, and anywhere else it may live. Keep in mind that this data can live in a variety of formats (e.g., database management system records or files) and media (e.g., hard drives or backup tapes). If your experience doing this for the first time is typical, you will probably be amazed at the places in which you find sensitive data.

Once you get a handle on what is your high-value data and where it resides, you can gradually expand the scope of your search to include less valuable, but still sensitive, data. For instance, if your critical data involves designs for next-generation radios, you would want to look for information that could allow someone to get insights into those designs even if they can't directly obtain them. So, for example, if you have patent filings, FCC license applications, and contracts with suppliers of electronic components, then an adversary may be able to use all this data to figure out what you're designing even without direct access to your new radio's plans. This is why it is so difficult for Apple to keep secret all the features of a new iPhone ahead of its launch. Often there is very little you can do to mitigate this risk, but some organizations have gone as far as to file patents and applications they don't intend to use in an effort to deceive adversaries as to their true plans. Obviously, and just as in any other security decision, the costs of these countermeasures must be weighed against the value of the information you're trying to protect. As you keep expanding the scope of your search, you will reach a point of diminishing returns in which the data you are inventorying is not worth the time you spend looking for it.

NOTE We cover the threats posed by adversaries compiling public information (aggregation) and using it to derive otherwise private information (inference) in Chapter 7.

Once you are satisfied that you have inventoried your sensitive data, the next step is to characterize it. We already covered the classification of information earlier in this chapter, so you should know all about data labels. Another element of this characterization is ownership. Who owns a particular set of data? Beyond that, who should be authorized to read or modify it? Depending on your organization, your data may have other characteristics of importance to the DLP effort, such as which data is regulated and how long it must be retained.

Data Flows Data that stays put is usually of little use to anyone. Most data will move according to specific business processes through specific network pathways. Understanding data flows at this intersection between business and IT is critical to implementing DLP. Many organizations put their DLP sensors at the perimeter of their networks, thinking that is where the leakages would occur. But if that's the only location these sensors are placed, a large number of leaks may not be detected or stopped. Additionally, as we will discuss in detail when we cover network-based DLP, perimeter sensors can often be bypassed by sophisticated attackers.

A better approach is to use a variety of sensors tuned to specific data flows. Suppose you have a software development team that routinely passes finished code to a quality assurance (QA) team for testing. The code is sensitive, but the QA team is authorized to read (and perhaps modify) it. However, the QA team is not authorized to access code under development or code from projects past. If an adversary compromises the computer used by a member of the QA team and attempts to access the source code for different projects, a DLP solution that is not tuned to that business process will not detect the compromise. The adversary could then repackage the data to avoid your perimeter monitors and successfully extract the data.

Data Protection Strategy The example just described highlights the need for a comprehensive, risk-based data protection strategy. The extent to which we attempt to mitigate these exfiltration routes depends on our assessment of the risk of their use. Obviously, as we increase our scrutiny of a growing set of data items, our costs will grow disproportionately. We usually can't watch everything all the time, so what do we do?

Once we have our data inventories and understand our data flows, we have enough information to do a risk assessment. Recall that we described this process in detail in Chapter 2. The trick is to incorporate data loss into that process. Since we can't guarantee that we will successfully defend against all attacks, we have to assume that sometimes our adversaries will gain access to our networks. Not only does our data protection strategy have to cover our approach to keeping attackers out, but it also must describe how we protect our data against a threat agent that is already inside. The following are some key areas to consider when developing data protection strategies:

- **Backup and recovery** Though we have been focusing our attention on data leaks, it is also important to consider the steps to prevent the loss of this data due to electromechanical or human failures. As we take care of this, we need to also consider the risk that, while we focus our attention on preventing leaks of our primary data stores, our adversaries may be focusing their attention on stealing the backups.
- **Data life cycle** Most of us can intuitively grasp the security issues at each of the stages of the data life cycle. However, we tend to disregard securing the data as it transitions from one stage to another. For instance, if we are archiving data at an offsite location, are we ensuring that it is protected as it travels there?
- **Physical security** While IT provides a wealth of tools and resources to help us protect our data, we must also consider what happens when an adversary just steals a hard drive left in an unsecured area, as happened to Sentara Heart Hospital in Norfolk, Virginia, in August 2015.
- **Security culture** Our information systems users can be a tremendous control if properly educated and incentivized. By developing a culture of security within our organizations, we not only reduce the incidence of users clicking on malicious links and opening attachments, but we also turn each of them into a security sensor, able to detect attacks that we may not otherwise be able to.

- **Privacy** Every data protection policy should carefully balance the need to monitor data with the need to protect our users' privacy. If we allow our users to check personal e-mail or visit social media sites during their breaks, would our systems be quietly monitoring their private communications?
- **Organizational change** Many large organizations grow because of mergers and acquisitions. When these changes happen, we must ensure that the data protection approaches of all entities involved are consistent and sufficient. To do otherwise is to ensure that the overall security posture of the new organization is the lesser of its constituents' security postures.

Implementation, Testing, and Tuning All the elements of a DLP process that we have discussed so far (i.e., data inventories, data flows, and data protection strategies) are administrative in nature. We finally get to discuss the part of DLP with which most of us are familiar: deploying and running a toolset. The sequence of our discussion so far has been deliberate in that the technological part needs to be informed by the other elements we've covered. Many organizations have wasted large sums of money on so-called solutions that, though well-known and highly regarded, are just not suitable for their particular environment.

Assuming we've done our administrative homework and have a good understanding of our true DLP requirements, we can evaluate products according to our own criteria, not someone else's. The following are some aspects of a possible solution that most organizations will want to consider when comparing competing products:

- **Sensitive data awareness** Different tools will use different approaches to analyzing the sensitivity of documents' contents and the context in which they are being used. In general terms, the more depth of analysis and breadth of techniques that a product offers, the better. Typical approaches to finding and tracking sensitive data include keywords, regular expressions, tags, and statistical methods.
- **Policy engine** Policies are at the heart of any DLP solution. Unfortunately, not all policy engines are created equal. Some allow extremely granular control but require obscure methods for defining these policies. Other solutions are less expressive but are simple to understand. There is no right answer here, so each organization will weigh this aspect of a set of solutions differently.
- **Interoperability** DLP tools must play nicely with existing infrastructure, which is why most vendors will assure you that their product is interoperable. The trick becomes to determine precisely how this integration takes place. Some products are technically interoperable but, in practice, require so much effort to integrate that they become infeasible.
- **Accuracy** At the end of the day, DLP solutions keep your data out of the hands of unauthorized entities. Therefore, the right solution is one that is accurate in its identification and prevention of incidents that result in the leakage of sensitive data. The best way to assess this criterion is by testing a candidate solution in an environment that mimics the actual conditions in the organization.

Once we select a DLP solution, the next interrelated tasks are integration, testing, and tuning. Obviously, we want to ensure that bringing the new toolset online won't disrupt any of our existing systems or processes, but testing needs to cover a lot more than that. The most critical elements when testing any DLP solution are to verify that it allows authorized data processing and to ensure that it prevents unauthorized data processing.

Verifying that authorized processes are not hampered by the DLP solution is fairly straightforward if we have already inventoried our data and the authorized flows. The data flows, in particular, will tell us exactly what our tests should look like. For instance, if we have a data flow for source code from the software development team to the QA team, then we should test that it is in fact allowed to occur by the new DLP tool. We probably won't have the resources to exhaustively test all flows, which means we should prioritize them based on their criticality to the organization. As time permits, we can always come back and test the remaining, and arguably less common or critical, processes (before our users do).

Testing the second critical element, that the DLP solution prevents unauthorized flows, requires a bit more work and creativity. Essentially, we are trying to imagine the ways in which threat agents might cause our data to leak. A useful tool in documenting these types of activities is called the misuse case. *Misuse cases* describe threat actors and the tasks they want to perform on the system. They are related to *use cases*, which are used by system analysts to document the tasks that authorized actors want to perform on a system. By compiling a list of misuse cases, we can keep a record of which data leak scenarios are most likely, most dangerous, or both. Just like we did when testing authorized flows, we can then prioritize which misuse cases we test first if we are resource constrained. As we test these potential misuses, it is important to ensure that the DLP system behaves in the manner we expect—that is to say, that it *prevents* a leak and doesn't just alert to it. Some organizations have been shocked to learn that their DLP solution has been alerting them about data leaks but doing nothing to stop them, letting their data leak right into the hands of their adversaries.

NOTE We cover misuse cases in detail in Chapter 18.

Finally, we must remember that everything changes. The solution that is exquisitely implemented, finely tuned, and effective immediately is probably going to be ineffective in the near future if we don't continuously monitor, maintain, and improve it. Apart from the efficacy of the tool itself, our organizations change as people, products, and services come and go. The ensuing cultural and environmental changes will also change the effectiveness of our DLP solutions. And, obviously, if we fail to realize that users are installing rogue access points, using thumb drives without restriction, or clicking malicious links, then it is just a matter of time before our expensive DLP solution will be circumvented.

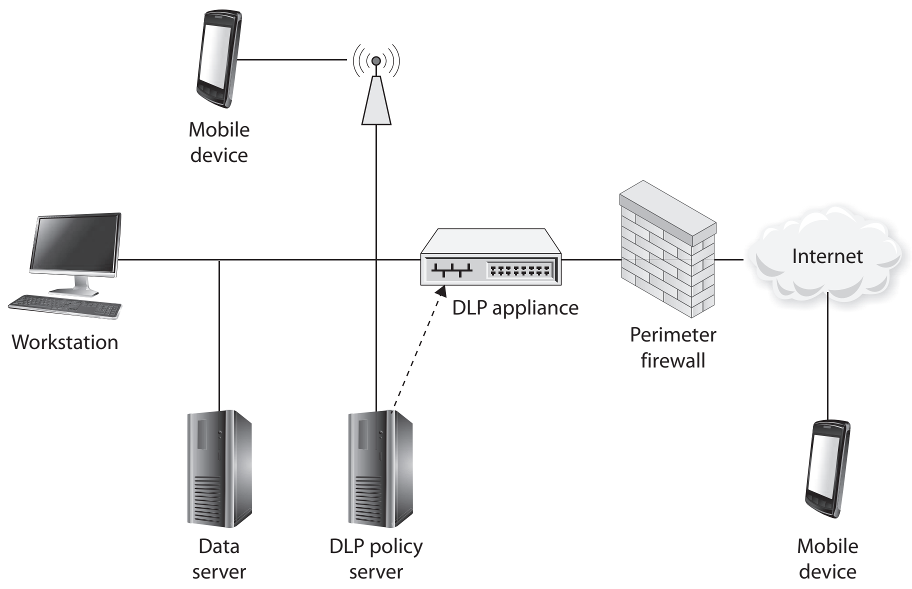


Figure 6-4 Network DLP

Network DLP

Network DLP (NDLP) applies data protection policies to data in motion. NDLP products are normally implemented as appliances that are deployed at the perimeter of an organization's networks. They can also be deployed at the boundaries of internal subnetworks and could be deployed as modules within a modular security appliance. Figure 6-4 shows how an NDLP solution might be deployed with a single appliance at the edge of the network and communicating with a DLP policy server.

DLP Resiliency

Resiliency is the ability to deal with challenges, damage, and crises and bounce back to normal or near-normal condition in short order. It is an important element of security in general and of DLP in particular.

Assume your organization's information systems have been compromised (and it wasn't detected): What does the adversary do next, and how can you detect and deal with *that*? It is a sad reality that virtually all organizations have been attacked and that most have been breached. A key differentiator between those who withstand attacks relatively unscathed and those who suffer tremendous damage is their attitude toward operating in contested environments. If an organization's entire security strategy hinges on keeping adversaries off its networks, then it will likely fail catastrophically when an adversary manages to break in. If, on the other hand, the strategy builds on the concept of resiliency and accounts for the continuation of critical processes even with adversaries operating inside the perimeter, then the failures will likely be less destructive and restoration may be much quicker.

From a practical perspective, the high cost of NDLP devices leads most organizations to deploy them at traffic choke points rather than throughout the network. Consequently, NDLP devices likely will not detect leaks that don't traverse the network segment on which the devices are installed. For example, suppose that an attacker is able to connect to a wireless access point and gain unauthorized access to a subnet that is not protected by an NDLP tool. This can be visualized in Figure 6-4 by supposing that the attacker is using the device connected to the WAP. Though this might seem like an obvious mistake, many organizations fail to consider their wireless subnets when planning for DLP. Alternatively, malicious insiders could connect their workstations directly to a mobile or external storage device, copy sensitive data, and remove it from the premises completely undetected.

The principal drawback of an NDLP solution is that it will not protect data on devices that are not on the organizational network. Mobile device users will be most at risk, since they will be vulnerable whenever they leave the premises. Since we expect the ranks of our mobile users to continue to increase into the future, this will be an enduring challenge for NDLP.

Endpoint DLP

Endpoint DLP (EDLP) applies protection policies to data at rest and data in use. EDLP is implemented in software running on each protected endpoint. This software, usually called a *DLP agent*, communicates with the DLP policy server to update policies and report events. Figure 6-5 illustrates an EDLP implementation.

EDLP allows a degree of protection that is normally not possible with NDLP. The reason is that the data is observable at the point of creation. When a user enters PII on

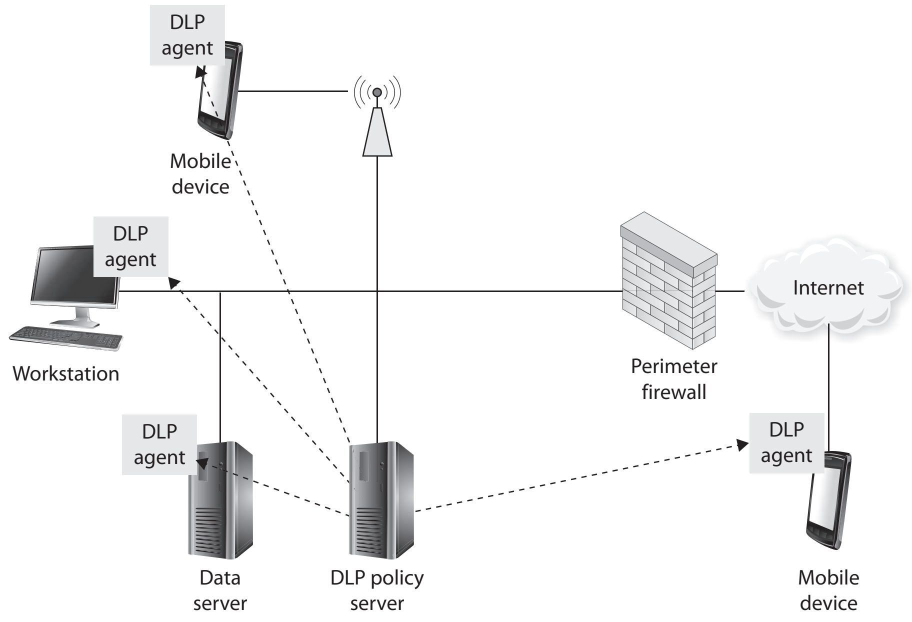


Figure 6-5 Endpoint DLP

the device during an interview with a client, the EDLP agent detects the new sensitive data and immediately applies the pertinent protection policies to it. Even if the data is encrypted on the device when it is at rest, it will have to be decrypted whenever it is in use, which allows for EDLP inspection and monitoring. Finally, if the user attempts to copy the data to a non-networked device such as a thumb drive, or if it is improperly deleted, EDLP will pick up on these possible policy violations. None of these examples would be possible using NDLP.

The main drawback of EDLP is complexity. Compared to NDLP, these solutions require a lot more presence points in the organization, and each of these points may have unique configuration, execution, or authentication challenges. Additionally, since the agents must be deployed to every device that could possibly handle sensitive data, the cost could be much higher than that of an NDLP solution. Another challenge is ensuring that all the agents are updated regularly, both for software patches and policy changes. Finally, since a pure EDLP solution is unaware of data-in-motion protection violations, it would be possible for attackers to circumvent the protections (e.g., by disabling the agent through malware) and leave the organization blind to the ongoing leakages. It is typically harder to disable NDLP, because it is normally implemented in an appliance that is difficult for attackers to exploit.

Hybrid DLP

Another approach to DLP is to deploy both NDLP and EDLP across the enterprise. Obviously, this approach is the costliest and most complex. For organizations that can afford it, however, it offers the best coverage. Figure 6-6 shows how a hybrid NDLP/EDLP deployment might look.

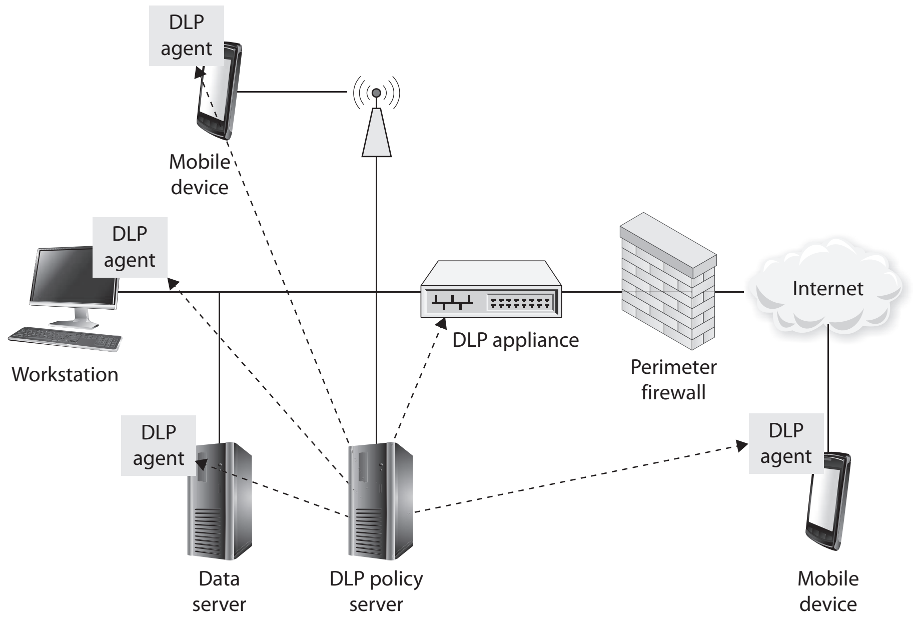


Figure 6-6 Hybrid NDLP/EDLP

Cloud Access Security Broker

The DLP approaches described so far work best (or perhaps only) in traditional network environments that have a clearly defined perimeter. But what about organizations that use cloud services, especially services that employees can access from their own devices? Whatever happens in the cloud is usually not visible (or controllable) by the organization. A *cloud access security broker (CASB)* is a system that provides visibility and security controls for cloud services. A CASB monitors what users do in the cloud and applies whatever policies and controls are applicable to that activity.

For example, suppose a nurse at a healthcare organization uses Microsoft 365 to take notes when interviewing a new patient. That document is created and exists only in the cloud and clearly contains sensitive healthcare information that must be protected under HIPAA. Without a CASB solution, the organization would depend solely on the nurse doing the right things, including ensuring the data is encrypted and not shared with any unauthorized parties. A CASB could automatically update the inventory of sensitive data, apply any labels in the document's metadata for tracking it, encrypt it, and ensure it is only shared with specific authorized entities.

Most CASBs do their work by leveraging one of two techniques: proxies or application programming interfaces (APIs). The proxy technique places the CASB in the data path between the endpoint and the cloud service provider, as shown on the left in Figure 6-7. For example, you could have an appliance in your network that automatically detects user connection requests to a cloud service, intercepts that user connection, and creates a tunnel to the service provider. In this way, all traffic to the cloud is routed through the CASB so that it can inspect it and apply the appropriate controls.

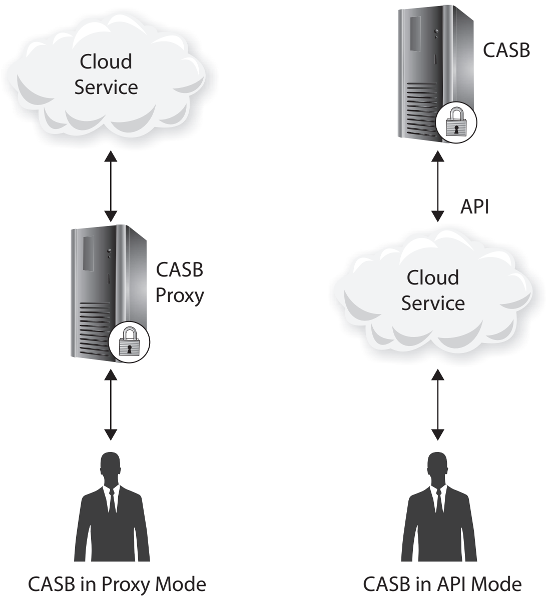


Figure 6-7 Two common approaches to implementing CASBs: proxy and API

But what if you have remote users who are not connected to your organization through a VPN? What about staff members trying to access the cloud services through a personal device (assuming that is allowed)? In those situations, you can set up a reverse proxy. The way this works is that the users log into the cloud service, which is configured to immediately route them back to the CASB, which then completes the connection back to the cloud.

There are a number of challenges with using proxies for CASBs. For starters, they need to intercept the users' encrypted traffic, which will generate browser alerts unless the browsers are configured to trust the proxy. While this works on organizational computers, it is a bit trickier to do on personally owned devices. Another challenge is that, depending on how much traffic goes to cloud service providers, the CASB can become a choke point that slows down the user experience. It also represents a single point of failure unless you deploy redundant systems. Perhaps the biggest challenge, however, has to do with the fast pace of innovation and updates to cloud services. As new features are added and others changed or removed, the CASB needs to be updated accordingly. The problem is not only that the CASB will miss something important but that it may actually break a feature by not knowing how to deal with it properly. For this reason, some vendors such as Google and Microsoft advise against using CASBs in proxy mode.

The other way to implement CASBs is by leveraging the APIs exposed by the service providers themselves, as you can see on the right side of Figure 6-7. An API is a way to have one software system directly access functionality in another one. For example, a properly authenticated CASB could ask Exchange Online (a cloud e-mail solution) for all the activities in the last 24 hours. Most cloud services include APIs to support CASB and, better yet, these APIs are updated by the vendors themselves. This ensures the CASB won't break anything as new features come up.

Chapter Review

Protecting data assets is a much more dynamic and difficult prospect than is protecting most other asset types. The main reason for this is that data is so fluid. It can be stored in unanticipated places, flow in multiple directions (and to multiple recipients) simultaneously, and end up being used in unexpected ways. Our data protection strategies must account for the various states in which our data may be found. For each state, there are multiple unique threats that our security controls must mitigate.

Still, regardless of our best efforts, data may end up in the wrong hands. We want to implement protection methods that minimize the risk of this happening, alert us as quickly as possible if it does, and allow us to track and, if possible, recover the data effectively. We devoted particular attention to three methods of protecting data that you should remember for the exam and for your job: Digital Rights Management (DRM), data loss/leak prevention (DLP), and cloud access security brokers (CASBs).

Quick Review

- Data at rest refers to data that resides in external or auxiliary storage devices, such as hard drives or optical discs.
- Every major operating system supports whole-disk encryption, which is a good way to protect data at rest.

- Data in motion is data that is moving between computing nodes over a data network such as the Internet.
- TLS, IPSec, and VPNs are typical ways to use cryptography to protect data in motion.
- Data in use is the term for data residing in primary storage devices, such as volatile memory (e.g., RAM), memory caches, or CPU registers.
- Scoping is taking a broader standard and trimming out the irrelevant or otherwise unwanted parts.
- Tailoring is making changes to specific provisions in a standard so they better address your requirements.
- A digital asset is anything that exists in digital form, has intrinsic value to the organization, and to which access should be restricted in some way.
- Digital asset management is the process by which organizations ensure their digital assets are properly stored, protected, and easily available to authorized users.
- Steganography is a method of hiding data in another media type so the very existence of the data is concealed.
- Digital Rights Management (DRM) refers to a set of technologies that is applied to controlling access to copyrighted data.
- Data leakage is the flow of sensitive information to unauthorized external parties.
- Data loss prevention (DLP) comprises the actions that organizations take to prevent unauthorized external parties from gaining access to sensitive data.
- Network DLP (NDLP) applies data protection policies to data in motion.
- Endpoint DLP (EDLP) applies data protection policies to data at rest and data in use.
- Cloud access security brokers (CASBs) provide visibility and control over user activities on cloud services.

Questions

Please remember that these questions are formatted and asked in a certain way for a reason. Keep in mind that the CISSP exam is asking questions at a conceptual level. Questions may not always have the perfect answer, and the candidate is advised against always looking for the perfect answer. Instead, the candidate should look for the best answer in the list.

1. Data at rest is commonly

A. Using a RESTful protocol for transmission

B. Stored in registers

C. Being transmitted across the network

D. Stored in external storage devices

2. Data in motion is commonly
   - **A.** Using a RESTful protocol for transmission
   - **B.** Stored in registers
   - **C.** Being transmitted across the network
   - **D.** Stored in external storage devices
3. Data in use is commonly
   - **A.** Using a RESTful protocol for transmission
   - **B.** Stored in registers
   - **C.** Being transmitted across the network
   - **D.** Stored in external storage devices
4. Which of the following best describes an application of cryptography to protect data at rest?
   - **A.** VPN
   - **B.** Degaussing
   - **C.** Whole-disk encryption
   - **D.** Up-to-date antivirus software
5. Which of the following best describes an application of cryptography to protect data in motion?
   - **A.** Testing software against side-channel attacks
   - **B.** TLS
   - **C.** Whole-disk encryption
   - **D.** EDLP
6. Which of the following is not a digital asset management task?
   - **A.** Tracking the number and location of backup versions
   - **B.** Deciding the classification of data assets
   - **C.** Documenting the history of changes
   - **D.** Carrying out secure disposal activities
7. Which data protection method would best allow you to detect a malicious insider trying to access a data asset within your corporate infrastructure?
   - **A.** Digital Rights Management (DRM)
   - **B.** Steganography
   - **C.** Cloud access security broker (CASB)
   - **D.** Data loss prevention (DLP)

8. What term best describes the flow of data assets to an unauthorized external party?
   - **A.** Data leakage
   - **B.** Data in motion
   - **C.** Data flow
   - **D.** Steganography

Answers

1. **D.** Data at rest is characterized by residing in secondary storage devices such as disk drives, DVDs, or magnetic tapes. Registers are temporary storage within the CPU and are used for data storage only when the data is being used.
2. **C.** Data in motion is characterized by network or off-host transmission. The RESTful protocol, while pertaining to a subset of data on a network, is not as good an answer as option C.
3. **B.** Registers are used only while data is being used by the CPU, so when data is resident in registers, it is, by definition, in use.
4. **C.** Data at rest is best protected using whole-disk encryption on the user workstations or mobile computers. None of the other options apply to data at rest.
5. **B.** Data in motion is best protected by network encryption solutions such as TLS, VPN, or IPSec. None of the other options apply to data in motion.
6. **B.** The classification of a data asset is determined by the asset owner before it starts being managed. Otherwise, how would the manager know how to handle it? All other answers are typically part of digital asset management.
7. **C.** Cloud access security brokers (CASBs) provide visibility and control over user activities on cloud services. Provided the asset in question is in the cloud, this would be your best option. Data loss prevention (DLP) systems are primarily concerned with preventing unauthorized external parties from gaining access to sensitive data.
8. **A.** Data leakage is the flow of sensitive information to unauthorized external parties.

This page intentionally left blank

PART III

Security Architecture and Engineering

This page intentionally left blank

CHAPTER 7

System Architectures

This chapter presents the following:

- General system architectures
- Industrial control systems
- Virtualized systems
- Cloud-based systems
- Pervasive systems
- Distributed systems

Computer system analysis is like child-rearing; you can do grievous damage, but you cannot ensure success.

—Tom DeMarco

As we have seen in previous chapters, most systems leverage other systems in some way, whether by sharing data with each other or by sharing services with each other. While each system has its own set of vulnerabilities, the interdependencies between them create a new class of vulnerabilities that we must address. In this chapter, we look at ways to assess and mitigate the vulnerabilities of security architectures, designs, and solution elements. We'll do this by looking at some of the most common system architectures. For each, we classify components based on their roles and the manner in which they interact with others. Along the way, we'll look at potential vulnerabilities in each architecture and also at the manner in which these vulnerabilities might affect other connected components.

General System Architectures

A *system* is a set of things working together in order to do something. An *architecture* describes the designed structure of something. A *system architecture*, then, is a description of how specific components are deliberately put together to perform some actions. Recall from the Chapter 4 discussion of TOGAF and the Zachman Framework that there are different perspectives or levels of abstraction at which a system architecture can be presented depending on the audience. In this chapter, we present what TOGAF would call application architectures. In other words, we describe how applications running in one or more computing devices interact with each other and with users.

Client-Based Systems

Let's start with the simplest computing system architecture, the one that ruled the early days of personal computing. *Client-based systems* are embodied in applications that execute entirely on one user device (such as a workstation or smartphone). The software is installed on a specific computer, and we can use it with no network connectivity. To be clear, the application may still reach out for software patches and updates or to save and retrieve files, but none of its core features require any processing on a remote device. Examples of these are the text and graphic applications that ship with almost every operating system. You could save documents on remote servers, but even with no networking the app is fully functional.

One of the main vulnerabilities of client-based systems is that they tend to have weak authentication mechanisms (if they have them at all). This means an adversary who gains access to the application would be able to also access its data on local or even remote data stores. Furthermore, this data is usually stored in plaintext (unless the underlying operating system encrypts it), which means that even without using the application, the adversary could read its data with ease.

Server-Based Systems

Unlike client-based systems, *server-based systems* (also called *client/server systems*) require that two (or more) separate applications interact with each other across a network connection in order for users to benefit from them. One application (the client) requests services over a network connection that the other application (the server) fulfills. Perhaps the most common example of a server-based application is your web browser, which is designed to connect to a web server. Sure, you could just use your browser to read local documents, but that's not really the way it's meant to be used. Most of us use our browsers to connect two tiers, a client and a server, which is why we call it a two-tier architecture.

Generally, server-based systems are known as *n*-tier architectures, where *n* is a numerical variable that can assume any value. The reason for this is that most of the time only the development team would know the number of tiers in the architecture (which could change over time) even if to the user it looks like just two. Consider the example of browsing the Web, which is probably a two-tier architecture if you are reading a static web page on a small web server. If, on the other hand, you are browsing a typical commercial site, you will probably be going through many more tiers. For example, your client (tier 1) could be connecting to a web server (tier 2) that provides the static HTML, CSS, and some images. The dynamic content, however, is pulled by the web server from an application server (tier 3) that in turn gets the necessary data from a backend database (tier 4). Figure 7-1 shows what this four-tier architecture would look like.

As you can imagine by looking at Figure 7-1, there are multiple potential security issues to address in a server-based architecture. For starters, access to each tier needs to be deliberately and strictly controlled. Having users authenticate from their clients makes perfect sense, but we must not forget that each of the tiers needs to establish and maintain trust with the others. A common way to ensure this is by developing access control lists (ACLs) that determine which connections are allowed. For example, the database management system in Figure 7-1 might be listening on port 5432 (the default

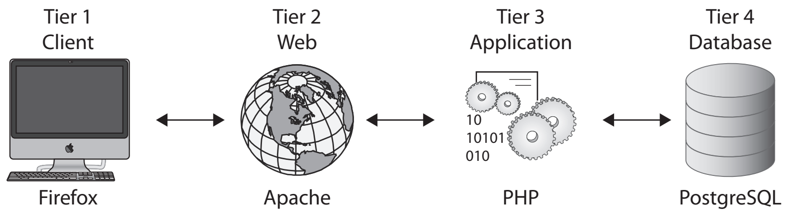


Figure 7-1 A typical four-tier server-based system

port for PostgreSQL, a popular open-source database server), so it makes perfect sense for the application server on tier 3 to connect to that port on the database server. However, it probably shouldn't be allowed to connect on port 3389 and establish a Remote Desktop Protocol (RDP) session because servers don't normally communicate this way.

The following are some other guidelines in securing server-based systems. Keep in mind, however, that this list is by no means comprehensive; it's just meant to give you food for thought.

- Block traffic by default between any components and allow only the specific set of connections that are absolutely necessary.
- Ensure all software is patched and updated as soon as possible.
- Maintain backups (ideally offline) of all servers.
- Use strong authentication for both clients and servers.
- Encrypt all network communications, even between the various servers.
- Encrypt all sensitive data stored anywhere in the system
- Enable logging of all relevant system events, ideally to a remote server.

Database Systems

Most interactive (as opposed to static) web content, such as that in the example four-tier architecture we just looked at, requires a web application to interact with some sort of data source. You may be looking at a catalog of products on an e-commerce site, updating customer data on a customer relationship management (CRM) system, or just reading a blog online. In any case, you need a system to manage your product, or customer, or blog data. This is where database systems come in.

A database management system (DBMS) is a software system that allows you to efficiently create, read, update, and delete (CRUD) any given set of data. Of course, you can always keep all the data in a text file, but that makes it *really* hard to organize, search, maintain, and share among multiple users. A DBMS makes this all easy. It is optimized for efficient storage of data, which means that, unlike flat files, it gives you ways to optimize the storage of all your information. A DBMS also provides the capability to speed up searches, for example, through the use of indexes. Another key feature of a DBMS is that it can provide mechanisms to prevent the accidental corruption of data while it is being manipulated. We typically call changes to a database *transactions*, which is a term to describe the sequence of actions required to change the state of the database.

A foundational principle in database transactions is referred to as their ACID properties, which stands for atomicity, consistency, isolation, and durability. *Atomicity* means that either the entire transactions succeeds or the DBMS rolls it back to its previous state (in other words, clicks the "undo" button). Suppose you are transferring funds between two bank accounts. This transaction consists of two distinct operations: first, you withdraw the funds from the first account, and then you deposit the same amount of funds into the second account. What would happen if there's a massive power outage right after the withdrawal is complete but before the deposit happens? In that case, the money could just disappear. If this was an atomic transaction, the system would detect the failure and put the funds back into the source account.

Consistency means that the transaction strictly follows all applicable rules (e.g., you can't withdraw funds that don't exist) on any and all data affected. *Isolation* means that if transactions are allowed to happen in parallel (which most of them are), then they will be isolated from each other so that the effects of one don't corrupt another. In other words, isolated transactions have the same effect whether they happen in parallel or one after the other. Finally, *durability* is the property that ensures that a completed transaction is permanently stored (for instance, in nonvolatile memory) so that it cannot be wiped by a power outage or other such failure.

Securing database systems mainly requires the same steps we listed for securing server-based systems. However, databases introduce two unique security issues you need to consider: aggregation and inference. *Aggregation* happens when a user does not have the clearance or permission to access specific information but she does have the permission to access components of this information. She can then figure out the rest and obtain restricted information. She can learn of information from different sources and combine it to learn something she does not have the clearance to know.

The following is a silly conceptual example. Let's say a database administrator does not want anyone in the Users group to be able to figure out a specific sentence, so he segregates the sentence into components and restricts the Users group from accessing it, as represented in Figure 7-2. However, Emily can access components A, C, and F. Because she is particularly bright, she figures out the sentence and now knows the restricted secret.

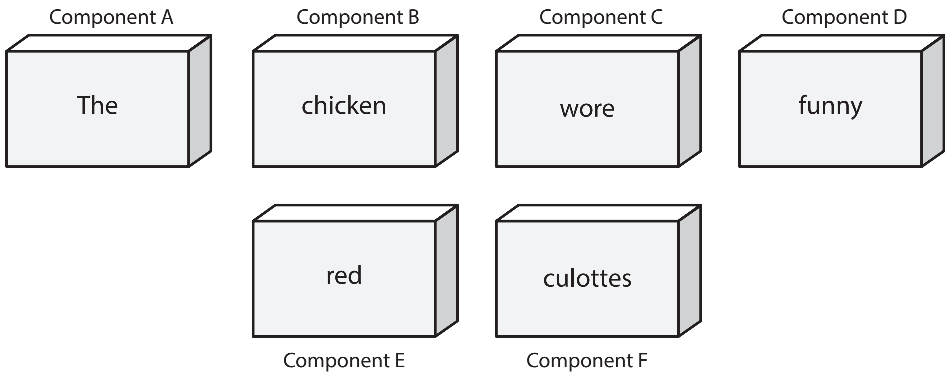


Figure 7-2 Because Emily has access to components A, C, and F, she can figure out the secret sentence through aggregation.

To prevent aggregation, the subject, and any application or process acting on the subject's behalf, needs to be prevented from gaining access to the whole collection, including the independent components. The objects can be placed into containers, which are classified at a higher level to prevent access from subjects with lower-level permissions or clearances. A subject's queries can also be tracked, and context-dependent access control can be enforced. This would keep a history of the objects that a subject has accessed and restrict an access attempt if there is an indication that an aggregation attack is underway.

EXAM TIP *Aggregation* is the act of combining information from separate sources. The combination of the data forms new information, which the subject does not have the necessary rights to access. The combined information has a sensitivity that is greater than that of the individual parts.

The other security issue is *inference*, which is the intended result of aggregation. The inference problem happens when a subject deduces the full story from the pieces he learned of through aggregation. This is seen when data at a lower security level indirectly portrays data at a higher level.

EXAM TIP *Inference* is the ability to derive information not explicitly available.

For example, if a clerk were restricted from knowing the planned movements of troops based in a specific country but did have access to food shipment requirement forms and tent allocation documents, he could figure out that the troops were moving to a specific place because that is where the food and tents are being shipped. The food shipment and tent allocation documents were classified as confidential, and the troop movement was classified as top secret. Because of the varying classifications, the clerk could access and ascertain top-secret information he was not supposed to know.

The trick is to prevent the subject, or any application or process acting on behalf of that subject, from indirectly gaining access to the inferable information. This problem is usually dealt with in the development of the database by implementing content- and context-dependent access control rules. *Content-dependent access control* is based on the sensitivity of the data. The more sensitive the data, the smaller the subset of individuals who can gain access to the data.

Context-dependent access control means that the software "understands" what actions should be allowed based upon the state and sequence of the request. So what does that mean? It means the software must keep track of previous access attempts by the user and understand what sequences of access steps are allowed. Content-dependent access control can go like this: "Does Julio have access to File A?" The system reviews the ACL on File A and returns with a response of "Yes, Julio can access the file, but can only read it." In a context-dependent access control situation, it would be more like this: "Does Julio have access to File A?" The system then reviews several pieces of data: What other access attempts has Julio made? Is this request out of sequence of how a safe series of requests takes place? Does this request fall within the allowed time period of system access (8 A.M. to 5 P.M.)? If the answers to all of these questions are within a set of preconfigured parameters, Julio can access the file. If not, he can't.

If context-dependent access control is being used to protect against inference attacks, the database software would need to keep track of what the user is requesting. So Julio makes a request to see field 1, then field 5, then field 20, which the system allows, but once he asks to see field 15, the database does not allow this access attempt. The software must be preprogrammed (usually through a rule-based engine) as to what sequence and how much data Julio is allowed to view. If he is allowed to view more information, he may have enough data to infer something we don't want him to know.

Obviously, content-dependent access control is not as complex as context-dependent access control because of the number of items that need to be processed by the system.

Some other common attempts to prevent inference attacks are cell suppression, partitioning the database, and noise and perturbation. *Cell suppression* is a technique used to hide specific cells that contain information that could be used in inference attacks. *Partitioning* the database involves dividing the database into different parts, which makes it much harder for an unauthorized individual to find connecting pieces of data that can be brought together and other information that can be deduced or uncovered. *Noise and perturbation* is a technique of inserting bogus information in the hopes of misdirecting an attacker or confusing the matter enough that the actual attack will not be fruitful.

Often, security is not integrated into the planning and development of a database. Security is an afterthought, and a trusted front end is developed to be used with the database instead. This approach is limited in the granularity of security and in the types of security functions that can take place.

A common theme in security is a balance between effective security and functionality. In many cases, the more you secure something, the less functionality you have. Although this could be the desired result, it is important not to excessively impede user productivity when security is being introduced.

High-Performance Computing Systems

All the architectures we've discussed so far in this chapter support significant amounts of computing. From high-end workstations used for high-resolution video processing to massive worldwide e-commerce sites supporting hundreds of millions of transactions per day, the power available to these systems today is very impressive indeed. As we will see shortly, the use of highly scalable cloud services can help turbo-charge these architectures, too. But what happens when even that is not enough? That's when we have to abandon these architectures and go for something altogether different.

High-performance computing (HPC) is the aggregation of computing power in ways that exceed the capabilities of general-purpose computers for the specific purpose of solving large problems. You may have already encountered this architecture if you've read about supercomputers. These are devices whose performance is so optimized that, even with electrons traveling at close to the speed of light down their wires, engineers spend significant design effort to make those wires even a few inches shorter. This is partially achieved by dividing the thousands (or tens of thousands) of processors in a typical system into tightly packed clusters, each with its own high-speed storage devices. Large problems can be broken down into individual jobs and assigned to the different clusters by a central scheduler. Once these smaller jobs are completed, they are progressively put together with other jobs (which, in turn, would be a job) until the final answer is computed.

While it may seem that most of us will seldom (if ever) work with HPC, the move toward big data analytics will probably drive us there sooner rather than later. For this reason, we need to be at least aware of some of the biggest security challenges with HPC. The first one is, quite simply, the very purpose of HPC's existence: efficiency. Large organizations spend millions of dollars building these custom systems for the purpose of crunching numbers really fast. Security tends to slow down (at least a little) just about everything, so we're already fighting an uphill battle. Fortunately, the very fact that HPC systems are so expensive and esoteric can help us justify the first rule for securing them, which is to put them in their own isolated enclave. Complete isolation is probably infeasible in many cases because raw data must flow in and solutions must flow out at some point. The goal would be to identify exactly how those flows should happen and then force them through a few gateways that can restrict who can communicate with the HPC system and under what conditions.

Another way in which HPC systems actually help us secure them is by following some very specific patterns of behavior during normal operations: jobs come in to the schedulers, which then assign them to specific clusters, which then return results in a specific format. Apart from some housekeeping functions, that's pretty much all that happens in an HPC system. It just happens *a lot*! These predictable patterns mean that anomaly detection is much easier than in a typical IT environment with thousands of users each doing their own thing.

Finally, since performance is so critical to HPC, most attacks are likely to affect it in noticeable ways. For this reason, simply monitoring the performance of the system will probably reveal nefarious activities. This noticeable impact on performance, as we will see shortly, affects other, less-esoteric systems, like those that control our factories, refineries, and electric grids.

Industrial Control Systems

Industrial control systems (ICS) consist of information technology that is specifically designed to control physical devices in industrial processes. ICS exist on factory floors to control conveyor belts and industrial robots. They exist in the power and water infrastructures to control the flows of these utilities. Because, unlike the majority of other IT systems, ICS control things that can directly cause physical harm to humans, safety must be paramount in operating and securing them. Another important consideration is that, due to the roles these systems typically fulfill in manufacturing and infrastructure, maintaining their "uptime" or availability is critical. For these two reasons (safety and availability), securing ICS requires a slightly different approach than that used to secure traditional IT systems.

EXAM TIP Safety is the paramount concern in operating and securing industrial control systems.

The term industrial control system actually is an umbrella term covering a number of somewhat different technologies that were developed independently to solve different problems. The term encompasses programmable logic controllers (PLCs) that open or close valves, remote terminal units (RTUs) that relay readings and execute commands, and specialized databases called data historians that capture all process data for analysis. ICS, with all its technologies, protocols, and devices, can generally be divided into two solution spaces:

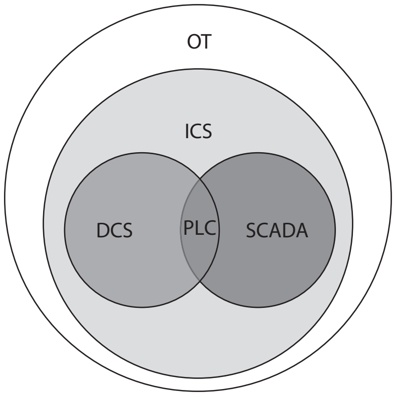
- Controlling physical processes that take place in a (more or less) local area. This involves what are called distributed control systems (DCS).
- Controlling processes that take place at multiple sites separated by significant distances. This is addressed through supervisory control and data acquisition (SCADA).

We'll delve into both of these solution spaces shortly.

NOTE A good resource for ensuring ICS safety, security, and availability is NIST Special Publication 800-82, Revision 2, *Guide to Industrial Control Systems (ICS) Security*, discussed further later in this section.

Another umbrella term you may see is *operational technology (OT)*, which includes both ICS and some traditional IT systems that are needed to make sure all the ICS devices can talk to each other. Figure 7-3 shows the relationship between these terms. Note that there is overlap between DCS and SCADA, in this case shown by the PLC, which supports both types of systems. Before we discuss each of the two major categories of ICS, let's take a quick look at some of the devices, like PLCs, that are needed to make these systems work.

Figure 7-3 Relationship between OT terms



Devices

There are a lot of different types of devices in use in OT systems. Increasingly, the lines between these types are blurred as different features converge in newer devices. However, most OT environments will have PLCs, a human-machine interface (HMI), and a data historian, which we describe in the following sections. Please note that you don't need to memorize what any of the following devices do in order to pass the CISSP exam. However, being familiar with them will help you understand the security implications of ICS and how OT and IT systems intertwine in the real world.

Programmable Logic Controller

When automation (the physical kind, not the computing kind to which we're accustomed) first showed up on factory floors, it was bulky, brittle, and difficult to maintain. If, for instance, you wanted an automatic hammer to drive nails into boxes moving through a conveyor belt, you would arrange a series of electrical relays such that they would sequentially actuate the hammer, retrieve it, and then wait for the next box. Whenever you wanted to change your process or repurpose the hammer, you would have to suffer through a complex and error-prone reconfiguration process.

Programmable logic controllers (PLCs) are computers designed to control electromechanical processes such as assembly lines, elevators, roller coasters, and nuclear centrifuges. The idea is that a PLC can be used in one application today and then easily reprogrammed to control something else tomorrow. PLCs normally connect to the devices they control over a standard serial interface such as RS-232, and to the devices that control them over Ethernet cables. The communications protocols themselves, however, are not always standard. The dominant protocols are Modbus and EtherNet/IP, but this is not universal. While this lack of universality in communications protocols creates additional challenges to securing PLCs, we are seeing a trend toward standardization of these serial connection protocols. This is particularly important because, while early PLCs had limited or no network connectivity, it is now rare to see a PLC that is not network-enabled.

PLCs can present some tough security challenges. Unlike the IT devices with which many of us are more familiar, these OT devices tend to have very long lifetimes. It's not unusual for production systems to include PLCs that are ten years old or older. Depending on how the ICS was architected, it may be difficult to update or patch the PLCs. When you couple this difficulty with the risk of causing downtime to a critical industrial process, you may understand why some PLCs can go years without getting patched. To make things worse, we've seen plenty of PLCs using factory default passwords that are well documented. While modern PLCs come with better security features, odds are that an OT environment will have some legacy controllers hiding somewhere. The best thing to do is to ensure that all PLC network segments are strictly isolated from all nonessential devices and are monitored closely for anomalous traffic.

Human-Machine Interface

A *human-machine interface (HMI)* is usually a regular workstation running a proprietary supervisory system that allows operators to monitor and control an ICS. An HMI normally has a dashboard that shows a diagram or schematic of the system being controlled,

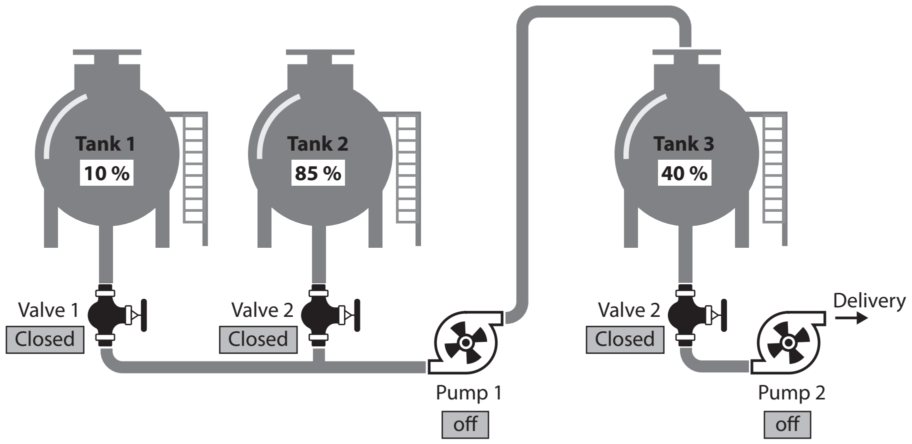


Figure 7-4 A simplified HMI screen

the readings from whatever sensors the system has in place, and buttons with which to control your actuators. Figure 7-4 shows a simplified HMI screen for a small fuel distribution system. Each of the three tanks shows how much fuel it contains. Three valves control the flow of fuel between the tanks, and all three are closed. If the operator wanted to move fuel around, she would simply click the CLOSED button, it would change to OPEN, and the fuel would be free to move. Similarly, clicking the OFF button on the pumps would turn them on to actually move the fuel around.

Another feature of HMIs is alarm monitoring. Each sensor (like those monitoring tank levels in our example) can be configured to alarm if certain values are reached. This is particularly important when it comes to the pressure in a pipeline, the temperature in a tank, or the load on a power line. HMIs usually include automation features that can automatically instruct PLCs to take certain actions when alarm conditions are met, such as tripping breakers when loads are too high.

HMIs simplify the myriad of details that make the ICS work so that the operators are not overwhelmed. In the simple example in Figure 7-4, Pump 1 would typically have a safety feature that would prevent it from being open unless Valve 1 and/or Valve 2 were open and the capacity in Tank 3 was not 100 percent. These features are manually programmed by the plant staff when the system is installed and are periodically audited for safety. Keep in mind that safety is of even more importance than security in OT environments.

Technically, securing an HMI is mostly the same as securing any IT system. Keep in mind that this is normally just a regular workstation that just happens to be running this proprietary piece of software. The challenge is that, because HMIs are part of mission-critical industrial systems where safety and efficiency are paramount, there can be significant resistance from OT staff to making any changes or taking any actions that can compromise either of these imperatives. These actions, of course, could include the typical security measures such as installing endpoint detection and response (EDR) systems, scanning them for vulnerabilities, conducting penetration tests, or even mandating unique credentials for each user with strong authentication. (Imagine what could happen if the HMI is locked, there is an emergency, and the logged-in user is on a break.)

Data Historian

As the name suggests, a *data historian* is a data repository that keeps a history of everything seen in the ICS. This includes all sensor values, alarms, and commands issued, all of which are timestamped. A data historian can communicate directly with other ICS devices, such as PLCs and HMIs. Sometimes, a data historian is embedded with (or at least running on the same workstation as) the HMI. Most OT environments, however, have a dedicated data historian (apart from the HMI) in a different network segment. The main reason for this is that this device usually communicates with enterprise IT systems for planning and accounting purposes. For example, the data historian in our fuel system example would provide data on how much fuel was delivered out of Tank 3.

One of the key challenges in securing the data historian stems from the fact that it frequently has to talk to both PLCs (and similar devices) and enterprise IT systems (e.g., for accounting purposes). A best practice when this is required is to put the data historian in a specially hardened network segment like a demilitarized zone (DMZ) and implement restrictive ACLs to ensure unidirectional traffic from the PLCs to the historian and from the historian to the enterprise IT systems. This can be done using a traditional firewall (or even a router), but some organizations instead use specialized devices called *data diodes*, which are security hardened and permit traffic to flow only in one direction.

Distributed Control System

A *distributed control system (DCS)* is a network of control devices within fairly close proximity that are part of one or more industrial processes. DCS usage is very common in manufacturing plants, oil refineries, and power plants, and is characterized by decisions being made in a concerted manner, but by different nodes within the system.

You can think of a DCS as a hierarchy of devices. At the bottom level, you will find the physical devices that are being controlled or that provide inputs to the system. One level up, you will find the microcontrollers and PLCs that directly interact with the physical devices but also communicate with higher-level controllers. Above the PLCs are the supervisory computers that control, for example, a given production line. You can also have a higher level that deals with plant-wide controls, which would require some coordination among different production lines.

As you can see, the concept of a DCS was born from the need to control fairly localized physical processes. Because of this, the communications protocols in use are not optimized for wide-area communications or for security. Another byproduct of this localized approach is that DCS users felt for many years that all they needed to do to secure their systems was to provide physical security. If the bad guys can't get into the plant, it was thought, then they can't break our systems. This is because, typically, a DCS consists of devices within the same plant. However, technological advances and converging technologies are blurring the line between a DCS and a SCADA system.

Supervisory Control and Data Acquisition

While DCS technology is well suited for local processes such as those in a manufacturing plant, it was never intended to operate across great distances. The *supervisory control and data acquisition (SCADA)* systems were developed to control large-scale physical processes involving nodes separated by significant distances. The main conceptual differences between DCS and SCADA are size and distances. So, while the control of a power plant is perfectly suited for a traditional DCS, the distribution of the generated power across a power grid would require a SCADA system.

SCADA systems typically involve three kinds of devices: endpoints, backends, and user stations. A *remote terminal unit (RTU)* is an endpoint that connects directly to sensors and/or actuators. Though there are still plenty of RTUs in use, many RTUs have been replaced with PLCs. The *data acquisition servers (DAS)* are backends that receive all data from the endpoints through a telemetry system and perform whatever correlation or analysis may be necessary. Finally, the users in charge of controlling the system interact with it through the use of the previously introduced *human-machine interface (HMI)*, the user station that displays the data from the endpoints and allows the users to issue commands to the actuators (e.g., to close a valve or open a switch).

One of the main challenges with operating at great distances is effective communications, particularly when parts of the process occur in areas with limited, spotty, or nonexistent telecommunications infrastructures. SCADA systems commonly use dedicated cables and radio links to cover these large expanses. Many legacy SCADA implementations rely on older proprietary communications protocols and devices. For many years, this led this community to feel secure because only someone with detailed knowledge of an obscure protocol and access to specialized communications gear could compromise the system. In part, this assumption is one of the causes of the lack of effective security controls on legacy SCADA communications. While this thinking may have been arguable in the past, today's convergence on IP-based protocols makes it clear that this is not a secure way of doing business.

ICS Security

The single greatest vulnerability in ICS is their increasing connectivity to traditional IT networks. This has two notable side effects: it accelerates convergence toward standard protocols, and it exposes once-private systems to anyone with an Internet connection. NIST SP 800-82 Rev. 2 has a variety of recommendations for ICS security, but we highlight some of the most important ones here:

- Apply a risk management process to ICS.
- Segment the network to place IDS/IPS at the subnet boundaries.
- Disable unneeded ports and services on all ICS devices.
- Implement least privilege through the ICS.
- Use encryption wherever feasible.
- Ensure there is a process for patch management.
- Monitor audit trails regularly.

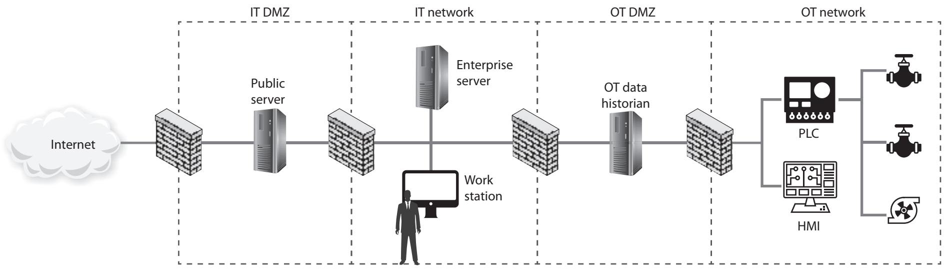


Figure 7-5 A simplified IT/OT environment

Let's look at a concrete (if seriously simplified) example in Figure 7-5. We're only showing a handful of IT and OT devices, but the zones are representative of a real environment. Starting from the right, you see the valves and pumps that are controlled by the PLC in the OT network. The PLC is directly connected to the HMI so that the PLC can be monitored and controlled by the operator. Both the PLC and the HMI are also connected (through a firewall) to the OT data historian in the OT DMZ. This is so that everything that happens in the OT network can be logged and analyzed. The OT data historian can also communicate with the enterprise server in the IT network to pass whatever data is required for planning, accounting, auditing, and reporting. If a user, say, in the accounting department, wants any of this data, he would get it from the enterprise server and would not be able to connect directly to the OT data historian. If a customer wanted to check via the Internet how much fuel they've been dispensed, they would log into their portal on the public server and that device would query the enterprise server for the relevant data.

Note that each segment is protected by a firewall (or data diode) that allows only specific devices in the next zone to connect in very restrictive ways to get only specific data. No device should ever be able to connect any further than one segment to the left or right.

Network segmentation also helps mitigate one of the common risks in many OT environments: unpatched devices. It is not rare to find devices that have been operating unpatched for several years. There are many reasons for this. First, ICS devices have very long shelf lives. They can remain in use for a decade or longer and may no longer receive updates from the manufacturer. They can also be very expensive, which means organizations may be unwilling or unable to set up a separate laboratory in which to test patches to ensure they don't cause unanticipated effects on the production systems. While this is a pretty standard practice in IT environments, it is pretty rare in the OT world. Without prior testing, patches could cause outages or safety issues and, as we know, maintaining availability and ensuring safety are the two imperatives of the OT world.

So, it is not all that strange for us to have to live with unpatched devices. The solution is to isolate them as best as we can. At a very minimum, it should be impossible for ICS devices to be reachable from the Internet. Better yet, we control access strictly from one zone to the next, as discussed previously. But for unpatched control devices, we have to be extremely paranoid and surround them with protective barriers that are monitored continuously.

EXAM TIP The most important principle in defending OT systems is to isolate them from the public Internet, either logically or physically.

Virtualized Systems

If you have been into computers for a while, you might remember computer games that did not have the complex, lifelike graphics of today's games. *Pong* and *Asteroids* were what we had to play with when we were younger. In those simpler times, the games were 16-bit and were written to work in a 16-bit MS-DOS environment. When our Windows operating systems moved from 16-bit to 32-bit, the 32-bit operating systems were written to be backward compatible, so someone could still load and play a 16-bit game in an environment that the game did not understand. The continuation of this little life pleasure was available to users because the OSs created virtual environments for the games to run in. Backward compatibility was also introduced with 64-bit OSs.

When a 32-bit application needs to interact with a 64-bit OS, it has been developed to make system calls and interact with the computer's memory in a way that would only work within a 32-bit OS—not a 64-bit system. So, the virtual environment simulates a 32-bit OS, and when the application makes a request, the OS converts the 32-bit request into a 64-bit request (this is called *thunking*) and reacts to the request appropriately. When the system sends a reply to this request, it changes the 64-bit reply into a 32-bit reply so the application understands it.

Today, virtual environments are much more advanced. *Virtualized systems* are those that exist in software-simulated environments. In our previous example of *Pong*, the 16-bit game "thinks" it is running on a 16-bit computer when in fact this is an illusion created by a layer of virtualizing software. In this case, the virtualized system was developed to provide backward compatibility. In many other cases, virtualization allows us to run multiple services or even full computers simultaneously on the same hardware, greatly enhancing resource (e.g., memory, processor) utilization, reducing operating costs, and even providing improved security, among other benefits.

Virtual Machines

Virtual machines (VMs) are entire computer systems that reside inside a virtualized environment. This means that you could have a legitimate Windows workstation running within a Linux server, complete with automatic updates from Microsoft, licensed apps from any vendor, and performance that is virtually indistinguishable (pun intended) from a similar Windows system running on "bare metal." This VM is commonly referred to as a *guest* that is executed in the *host* environment, which, in our example, would be the Linux server.

Virtualization allows a single host environment to execute multiple guests at once, with multiple VMs dynamically pooling resources from a common physical system. Computer resources such as RAM, processors, and storage are emulated through the host environment. The VMs do not directly access these resources; instead, they communicate with a *hypervisor* within the host environment, which is responsible for managing system resources. The hypervisor is the central program that controls the execution of the various guest operating

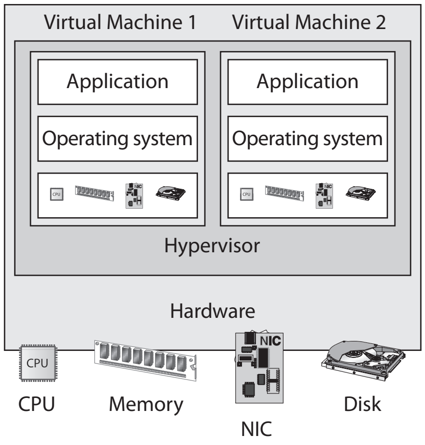


Figure 7-6 The hypervisor controls virtual machine instances.

systems and provides the abstraction level between the guest and host environments, as shown in Figure 7-6.

There are two types of hypervisors. A *type 1 hypervisor* runs directly on hardware or "bare metal" and manages access to it by its VMs. This is the sort of setup we use in server rooms and cloud environments. Examples of type 1 hypervisors are Citrix/Xen Server and VMware ESXi. A *type 2 hypervisor*, on the other hand, runs as an application on an OS. This allows users, for example, to host a Windows VM in their macOS computer. Type 2 hypervisors are commonly used by developers and security researchers to test their work in a controlled environment or use applications that are not available for the host OS. Examples of type 2 hypervisors are Oracle VM VirtualBox and VMware Workstation.

Hypervisors allow you to have one computer running several different operating systems at one time. For example, you can run a system with Windows 10, Linux, and Windows 2016 on one computer. Think of a house that has different rooms. Each OS gets its own room, but each shares the same resources that the house provides—a foundation, electricity, water, roof, and so on. An OS that is "living" in a specific room does not need to know about or interact with another OS in another room to take advantage of the resources provided by the house. The same concept happens in a computer: Each OS shares the resources provided by the physical system (memory, processor, buses, and so on). The OSs "live" and work in their own "rooms," which are the guest VMs. The physical computer itself is the host.

Why would we want to virtualize our machines? One reason is that it is cheaper than having a full physical system for each and every operating system. If they can all live on one system and share the same physical resources, your costs are reduced immensely. This is the same reason people get roommates. The rent can be split among different people, and all can share the same house and resources. Another reason to use virtualization is security. Providing to each OS its own "clean" environment to work within reduces the possibility of the various OSs negatively interacting with each other.

Furthermore, since every aspect of the virtual machine, including the contents of its disk drives and even its memory, is stored as files within the host, restoring a backup is a snap.

All you have to do is drop the set of backed-up files onto a new hypervisor and you will instantly restore a VM to whatever state it was in when the backup was made. Contrast this with having to rebuild a physical computer from backups, which can take a lot longer.

On the flip side of security, any vulnerability in the hypervisor would give an attacker unparalleled and virtually undetectable (pun not intended) power to compromise the confidentiality, integrity, or availability of VMs running on it. This is not a hypothetical scenario, as both VirtualBox and VMware have reported (and patched) such vulnerabilities in recent years. The takeaway from these discoveries is that we should assume that any component of an information system could be compromised and ask ourselves the questions "how would I detect it?" and "how can I mitigate it?"

Containerization

As virtualization matured, a new branch called *containerization* emerged. A container is an application that runs in its own isolated user space. Whereas virtual machines have their own complete operating systems running on top of hypervisors and share the resources provided by the bare metal, containers sit on top of OSs and share the resources provided by the host OS. Instead of abstracting the hardware for guest OSs, container software abstracts the kernel of the OS for the applications running above it. This allows for low overhead in running many applications and improved speed in deploying instances, because a whole VM doesn't have to be started for every application. Rather, the application, services, processes, libraries, and any other dependencies can be wrapped up into one unit.

Additionally, each container operates in a sandbox, with the only means to interact being through the user interface or application programming interface (API) calls. The big names to know in this space are Docker on the commercial side and Kubernetes as the open-source alternative. Containers have enabled rapid development operations because developers can test their code more quickly, changing only the components necessary in the container and then redeploying.

Securing containers requires a different approach than we'd take with full-sized VMs. Obviously, we want to harden the host OS. But we also need to pay attention to each container and the manner in which it interacts with clients and other containers. Keep in mind that containers are frequently used in rapid development. This means that, unless you build secure development right into the development team, you will likely end up with insecure code. We'll address the integration of development, security, and operations staff when we discuss DevSecOps in Chapters 24 and 25, but for now remember that it's really difficult to secure containers that have been developed insecurely.

NIST offers some excellent specific guidance on securing containers in NIST SP 800-190, *Application Container Security Guide*. Among the most important recommendations in that publication are the following:

- Use container-specific host OSs instead of general-purpose ones to reduce attack surfaces.
- Only group containers with the same purpose, sensitivity, and threat posture on a single host OS kernel to allow for additional defense in depth.

- Adopt container-specific vulnerability management tools and processes for images to prevent compromises.
- Use container-aware runtime defense tools such as intrusion prevention systems.

Microservices

A common use of containers is to host *microservices*, which is a way of developing software where, rather than building one large enterprise application, the functionality is divided into multiple smaller components that, working together in a distributed manner, implement all the needed features. Think of it as a software development version of the old "divide and conquer" approach. Microservices are considered an architectural style rather than a standard, but there is broad consensus that they consist of small, decentralized, individually deployable services built around business capabilities. They also tend to be *loosely coupled*, which means there aren't a lot of dependencies between the individual services. As a result, microservices are quick to develop, test, and deploy and can be exchanged without breaking the larger system. For many business applications, microservices are also more efficient and scalable than monolithic server-based architectures.

NOTE Containers and microservices don't have to be used together. It's just very common to do so.

The decentralization of microservices can present a security challenge. How can you track adversarial behaviors through a system of microservices, where each service does one discrete task? The answer is *log aggregation*. Whereas microservices are decentralized, we want to log them in a centralized fashion so we can look for patterns that span multiple services and can point to malicious intent. Admittedly, you will need automation and perhaps data analytics or artificial intelligence to detect these malicious events, but you won't have a chance at spotting them unless you aggregate the logs.

Serverless

If we gain efficiency and scalability by breaking up a big service into a bunch of microservices, can we gain even more by breaking up the microservices further? The answer, in many cases, is yes, because hosting a service (even a micro one) means that you have to provision, manage, update, and run the thing. So, if we're going to go further down this road of dividing and conquering, the next level of granularity is individual functions.

Hosting a service usually means setting up hardware, provisioning and managing servers, defining load management mechanisms, setting up requirements, and running the service. In a *serverless* architecture, the services offered to end users, such as compute, storage, or messaging, along with their required configuration and management, can be performed without a requirement from the user to set up any server infrastructure. The focus is strictly at the individual function level. These serverless models are designed primarily for massive scaling and high availability. Additionally, from a cost perspective, they are attractive, because billing occurs based on what cycles are actually used versus what is provisioned in advance.

Integrating security mechanisms into serverless models is not as simple as ensuring that the underlying technologies are hardened. Because visibility into host infrastructure operations is limited, implementing countermeasures for remote code execution or modifying access control lists isn't as straightforward as it would be with traditional server design. In the serverless model, security analysts are usually restricted to applying controls at the application or function level and then keeping a close eye on network traffic.

As you probably know by now, serverless architectures rely on the capability to automatically and securely provision, run, and then deprovision computing resources on demand. This capability undergirds their economic promise: you only pay for exactly the computing you need to perform just the functions that are required, and not a penny more. It is also essential to meet the arbitrary scalability of serverless systems. This capability is characteristic of cloud computing.

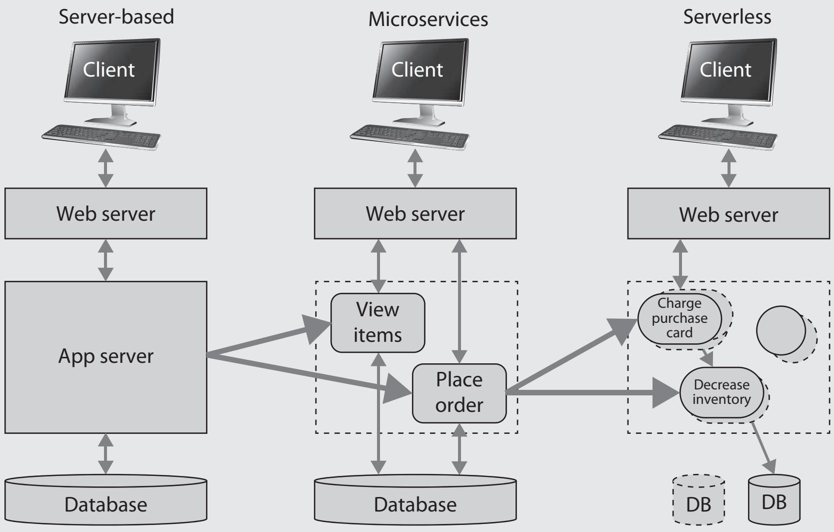
Comparing Server-Based, Microservice, and Serverless Architectures

A typical service houses a bunch of functions within it. Think of a very simple e-commerce web application server. It allows customers to log in, view the items that are for sale, and place orders. When placing an order, the server invokes a multitude of functions. For instance, it may have to charge the payment card, decrease inventory, schedule a shipment, and send a confirmation message. Here's how each of these three architectures handle this.

Server-based implementations provide all services (and their component functions) in the same physical or virtual server that houses the monolithic web application. The server must always be available (meaning powered on and connected to the Internet). If there's a sudden spike in orders, you better hope you have enough bandwidth, memory, and processing power to handle it. If you don't, you get to build a new server from scratch and either replace the original server with a beefier one or load-balance between the two. Either way, you now have more infrastructure to keep up and running.

Microservices can be created for each of the major features in the web application: view items and place orders. Each microservice lives in its own container and gets called as needed. If you see that spike in orders, you deploy a new container (in seconds), perhaps in a different host, and can destroy it when you no longer need it. Sure, you'll need some supervisory process to figure out when and how to spin up new containers, but at least you can dynamically respond to increased demands.

Serverless approaches would decompose each service into its fundamental functions and then dynamically provision those functions as needed. In other words, there is never a big web application server (like in the server-based approach) or even a microservice for order processing that is up and running. Instead, the charge_payment_card function is invoked in whatever infrastructure is available whenever a card needs to be processed. If that function is successful, it invokes the decrease_inventory function, again, in whatever infrastructure is available, and so on. After each function terminates, it simply evaporates so that no more resources are consumed than are absolutely needed. If there's a sudden spike in demand, the orchestrator spins up whatever additional resources are needed to run as many functions as are required.



Cloud-Based Systems

If you were asked to install a brand-new server room for your organization, you would probably have to clear your calendar for weeks (or longer) to address the many tasks that would be involved. From power and environmental controls to hardware acquisition, installation, and configuration to software builds, the list is long and full of headaches. Now, imagine that you can provision all the needed servers in minutes using a simple graphical interface or a short script and that you can get rid of them just as quickly when you no longer need them. This is one of the benefits of cloud computing.

Cloud computing is the use of shared, remote computing devices for the purpose of providing improved efficiencies, performance, reliability, scalability, and security. These devices are usually based on virtual machines running on shared infrastructure and can be outsourced to a third-party cloud service provider (CSP) on a *public cloud* or provided in-house on a *private cloud*. If you don't feel comfortable sharing infrastructure with random strangers (though this is done securely), there is also a *virtual private cloud (VPC)* model in which you get your own walled garden inside an otherwise public cloud.

Generally speaking, there are three models for cloud computing services:

- **Software as a Service (SaaS)** The user of SaaS is allowed to use a specific application that executes on the CSP's environment. Examples of SaaS are Microsoft 365 and Google Apps, which you use via a web interface but someone else provisions and maintains everything for you.
- **Platform as a Service (PaaS)** In this model, the user gets access to a computing platform that is typically built on a server operating system. An example of this would be spawning an instance of Windows Server 2019 to provide a web server. The CSP is normally responsible for configuring and securing the platform, however, so the user normally doesn't get administrative privileges over the entire platform.
- **Infrastructure as a Service (IaaS)** If you want full, unfettered access to (and responsibility for securing) a cloud-based VM, you would want to use the IaaS model. Following up on the previous example, this would allow you to manage the patching of the Windows Server 2019 instance. The catch is that the CSP has no responsibility for security; it's all on you.

If you are a user of IaaS, you probably won't do things too differently than you already do to secure your systems. The only exception is that you wouldn't have physical access to the computers if a CSP hosts them. If, on the other hand, you use SaaS or PaaS, the security of your systems will almost always rely on the policies and contracts that you put into place. The policies will dictate how your users interact with the cloud services. This would include the information classification levels that would be allowed on those services, terms of use, and other policies. The contracts will specify the quality of service and what the CSP will do with or for you in responding to security events.

CAUTION It is imperative that you carefully review the terms of service when evaluating a potential contract for cloud services and consider them in the context of your organization's security. Though the industry is getting better all the time, security provisions are oftentimes lacking in these contracts at this time.

Software as a Service

SaaS is pervasively used by most enterprises. According to some estimates, the average company uses nearly 2,000 unique cloud services for everything from writing memos to managing their sales pipeline. The whole idea is that, apart from a fairly small amount of allowed customization, you just pay for the licenses and the vendor takes care of making sure all your users have access to the software, regardless of where they are.

Given the popularity of SaaS solutions, cloud service providers such as Microsoft, Amazon, Cisco, and Google often dedicate large teams to securing all aspects of their service infrastructure. Increasingly, however, most security incidents involving SaaS occur at the data-handling level, where these infrastructure companies do not have the responsibility or visibility required to take action. For example, how could the CSP be held liable when one of your employees shares a confidential file with an unauthorized third party?

So, visibility is one of our main concerns as security professionals when it comes to SaaS. Do you know what assets you have and how they are being used? The “McAfee 2019 Cloud Adoption and Risk Report” describes the disconnect between the number of cloud services that organizations believe are being accessed by their users and the number of cloud services that are actually being accessed. The discrepancy, according to the report, can be several orders of magnitude. As we have mentioned before, you can’t protect what you don’t know you have. This is where solutions like cloud access security brokers (CASBs) and data loss prevention (DLP) systems can come in very handy.

NOTE We already covered CASBs and DLP systems in Chapter 6.

Platform as a Service

What if, instead of licensing someone else’s application, you have developed your own and need a place to host it for your users? You’d want to have a fair amount of flexibility in terms of configuring the hosting environment, but you probably could use some help in terms of provisioning and securing it. You can secure the app, for sure, but would like someone else to take care of things like hardening the host, patching the underlying OS, and maybe even monitoring access to the VM. This is where PaaS comes in.

PaaS has a similar set of functionalities as SaaS and provides many of the same benefits in that the CSP manages the foundational technologies of the stack in a manner transparent to the end user. You simply tell your provider, “I’d like a Windows Server 2019 with 64 gigabytes of RAM and eight cores,” and, voilà, there it is. You get direct access to a development or deployment environment that enables you to build and host your own solutions on a cloud-based infrastructure without having to build your own infrastructure. PaaS solutions, therefore, are optimized to provide value focused on software development. PaaS, by its very nature, is designed to provide an organization with tools that interact directly with what may be its most important asset: its source code.

At the physical infrastructure, in PaaS, service providers assume the responsibility of maintenance and protection and employ a number of methods to deter successful exploits at this level. This often means PaaS providers require trusted sources for hardware, use strong physical security for its data centers, and monitor access to the physical servers and connections to and from them. Additionally, PaaS providers often enhance their protection against distributed denial-of-service (DDoS) attacks using network-based technologies that require no additional configuration from the user.

While the PaaS model makes a lot of provisioning, maintenance, and security problems go away for you, it is worth noting that it does nothing to protect the software systems you host there. If you build and deploy insecure code, there is very little your CSP will be able to do to keep it protected. PaaS providers focus on the infrastructure on which the service runs, but you still have to ensure that the software is secure and the appropriate controls are in place. We'll dive into how to build secure code in Chapters 24 and 25.

Infrastructure as a Service

Sometimes, you just have to roll up your sleeves, get your hands dirty, and build your own servers from the ground up. Maybe the applications and services you have developed require your IT and security teams to install and configure components at the OS level that would not be accessible to you in the PaaS model. You don't need someone to make platforms that they manage available to you; you need to build platforms from the ground up yourself. IaaS gives you just that. You upload an image to the CSP's environment and build your own hosts however you need them.

As a method of efficiently assigning hardware through a process of constant assignment and reclamation, IaaS offers an effective and affordable way for organizations to get all of the benefits of managing their own hardware without incurring the massive overhead costs associated with acquisition, physical storage, and disposal of the hardware. In this service model, the vendor provides the hardware, network, and storage resources necessary for the user to install and maintain any operating system, dependencies, and applications they want. The vendor deals with all hardware issues for you, leaving you to focus on the virtual hosts.

In the IaaS model, the majority of the security controls (apart from physical ones) are your responsibility. Obviously, you want to have a robust security team to manage these. Still, there are some risks that are beyond your control and for which you rely on your vendor, such as any vulnerabilities that could allow an attacker to exploit flaws in hard disks, RAM, CPU caches, and GPUs. One attack scenario affecting IaaS cloud providers could enable a malicious actor to implant persistent back doors for data theft into bare-metal cloud servers. A vulnerability either in the hypervisor supporting the visualization of various tenant systems or in the firmware of the hardware in use could introduce a vector for this attack. This attack would be difficult for the customer to detect because it would be possible for all services to appear unaffected at a higher level of the technology stack.

Though the likelihood of a successful exploit of this kind of vulnerability is quite low, defects and errors at this level may still incur significant costs unrelated to an actual exploit. Take, for example, the 2014 hypervisor update performed by Amazon Web Services (AWS), which essentially forced a complete restart of a major cloud offering, the Elastic Compute Cloud (EC2). In response to the discovery of a critical security flaw in the open-source hypervisor Xen, Amazon forced EC2 instances globally to restart to ensure the patch would take correctly and that customers remained unaffected. In most cases, though, as with many other cloud services, attacks against IaaS environments are possible because of misconfiguration on the customer side.

Everything as a Service

It's worth reviewing the basic premise of cloud service offerings: you save money by only paying for exactly the resources you actually use, while having the capacity to scale those up as much as you need to at a moment's notice. If you think about it, this model can apply to things other than applications and computers. *Everything as a Service (XaaS)* captures the trend to apply the cloud model to a large range of offerings, from entertainment (e.g., television shows and feature-length movies), to cybersecurity (e.g., Security as a Service), to serverless computing environments (e.g., Function as a Service). Get ready for the inevitable barrage of <fill-in-the-blank> as a Service offerings coming your way.

Cloud Deployment Models

By now you may be a big believer in the promise of cloud computing but may be wondering, "Where, exactly, *is* the cloud?" The answer, as in so many questions in our field, is "It depends." There are four common models for deploying cloud computing resources, each with its own features and limitations:

- A *public cloud* is the most prevalent model, in which a vendor like AWS owns all the resources and provides them as a service to all its customers. Importantly, the resources are shared among all customers, albeit in a transparent and secure manner. Public cloud vendors typically also offer a virtual private cloud (VPC) as an option, in which increased isolation between users provides added security.
- A *private cloud* is owned and operated by the organization that uses its services. Here, you own, operate, and maintain the servers, storage, and networking needed to provide the services, which means you don't share resources with anyone. This approach can provide the best security, but the tradeoff might be higher costs and a cap on scalability.
- A *community cloud* is a private cloud that is co-owned (or at least shared) by a specific set of partner organizations. This approach is commonly implemented in large conglomerates where multiple firms report to the same higher-tier headquarters.
- A *hybrid cloud* combines on-premises infrastructure with a public cloud, with a significant effort placed in the management of how data and applications leverage each solution to achieve organizational goals. Organizations that use a hybrid model often derive benefits offered by both public and private models.

Pervasive Systems

Cloud computing is all about the concentration of computing power so that it may be dynamically reallocated among customers. Going in the opposite conceptual direction, *pervasive computing* (also called *ubiquitous computing* or *ubicomp*) is the concept that small (even tiny) amounts of computing power are spread out everywhere and computing is embedded into everyday objects that communicate with each other, often with little or no user interaction, to do very specific things for particular customers. In this model, computers are everywhere and communicate on their own with each other, bringing really cool new features but also really thorny new security challenges.

Embedded Systems

An *embedded system* is a self-contained computer system (that is, it has its own processor, memory, and input/output devices) designed for a very specific purpose. An embedded device is part of (or embedded into) some other mechanical or electrical device or system. Embedded systems typically are cheap, rugged, and small, and they use very little power. They are usually built around *microcontrollers*, which are specialized devices that consist of a CPU, memory, and peripheral control interfaces. Microcontrollers have a very basic operating system, if they have one at all. A digital thermometer is an example of a very simple embedded system; other examples of embedded systems include traffic lights and factory assembly line controllers. As you can see from these examples, embedded systems are frequently used to sense and/or act on a physical environment. For this reason, they are sometimes called *cyber-physical systems*.

The main challenge in securing embedded systems is that of ensuring the security of the software that drives them. Many vendors build their embedded systems around commercially available microprocessors, but they use their own proprietary code that is difficult, if not impossible, for a customer to audit. Depending on the risk tolerance of your organization, this may be acceptable as long as the embedded systems are standalone. The problem, however, is that these systems are increasingly shipping with some sort of network connectivity. For example, some organizations have discovered that some of their embedded devices have "phone home" features that are not documented. In some cases, this has resulted in potentially sensitive information being transmitted to the manufacturer. If a full audit of the embedded device security is not possible, at a very minimum, you should ensure that you see what data flows in and out of it across any network.

Another security issue presented by many embedded systems concerns the ability to update and patch them securely. Many embedded devices are deployed in environments where they have no Internet connectivity. Even if this is not the case and the devices can check for updates, establishing secure communications or verifying digitally signed code, both of which require processor-intensive cryptography, may not be possible on a cheap device.

Internet of Things

The *Internet of Things (IoT)* is the global network of connected embedded systems. What distinguishes the IoT is that each node is connected to the Internet and is uniquely addressable. By some accounts, this network is expected to reach 31 billion devices by 2025, which makes this a booming sector of the global economy. Perhaps the most visible aspect of this explosion is in the area of smart homes in which lights, furnaces, and even refrigerators collaborate to create the best environment for the residents.

With this level of connectivity and access to physical devices, the IoT poses many security challenges. Among the issues to address by anyone considering adoption of IoT devices are the following:

- **Authentication** Embedded devices are not known for incorporating strong authentication support, which is the reason why most IoT devices have very poor (if any) authentication.

- **Encryption** Cryptography is typically expensive in terms of processing power and memory requirements, both of which are very limited in IoT devices. The fallout of this is that data at rest and data in transit can be vulnerable in many parts of the IoT.
- **Updates** Though IoT devices are networked, many vendors in this fast-moving sector do not provide functionality to automatically update their software and firmware when patches are available.

Perhaps the most dramatic illustration to date of what can happen when millions of insecure IoT devices are exploited by an attacker is the Mirai botnet. Mirai is a malware strain that infects IoT devices and was behind one of the largest and most effective botnets in recent history. The Mirai botnet took down major websites via massive DDoS attacks against several sites and service providers using hundreds of thousands of compromised IoT devices. In October 2016, a Mirai attack targeted the popular DNS provider Dyn, which provided name resolution to many popular websites such as Airbnb, Amazon, GitHub, HBO, Netflix, PayPal, Reddit, and Twitter. After taking down Dyn, Mirai left millions of users unable to access these sites for hours.

Distributed Systems

A distributed system is one in which multiple computers work together to do something. The earlier section "Server-Based Systems" already covered a specific example of a four-tier distributed system. It is this collaboration that more generally defines a distributed system. A server-based system is a specific kind of distributed system in which devices in one group (or tier) act as clients for devices in an adjacent group. A tier-1 client cannot work directly with the tier-4 database, as shown earlier in Figure 7-1. We could then say that a *distributed system* is any system in which multiple computing nodes, interconnected by a network, exchange information for the accomplishment of collective tasks.

Not all distributed systems are hierarchical like the example in Figure 7-1. Another approach to distributed computing is found in *peer-to-peer systems*, which are systems in which each node is considered an equal (as opposed to a client or a server) to all others. There is no overarching structure, and nodes are free to request services from any other node. The result is an extremely resilient structure that fares well even when large numbers of nodes become disconnected or otherwise unavailable. If you had a typical client/server model and you lost your server, you'd be down for the count. In a peer-to-peer system, you could lose multiple nodes and still be able to accomplish whatever task you needed to. Clearly, not every application lends itself to this model, because some tasks are inherently hierarchical or centralized. Popular examples of peer-to-peer systems are file sharing systems like BitTorrent, anonymizing networks like The Onion Router (TOR), and cryptocurrencies like bitcoin.

One of the most important issues in securing distributed systems is network communications, which are essential to these systems. While the obvious approach would be to encrypt all traffic, it can be challenging to ensure all nodes are using cryptography that is robust enough to mitigate attacks. This is particularly true when the system includes IoT or OT components that may not have the same crypto capabilities as traditional computers.

Even if you encrypt all traffic (and you really should) in a distributed system, there's still the issue of trust. How do we ensure that every user and every node is trustworthy? How could you tell if part of the system was compromised? Identity and access management is another key area to address, as is the ability to isolate users or nodes from the system should they become compromised.

NOTE We will discuss identity and access management (IAM) in Chapter 16.

Edge Computing Systems

An interesting challenge brought about by the proliferation of IoT devices is how to service them in a responsive, scalable, and cost-effective manner. To understand the problem, let's first consider a server-based example. Suppose you enjoy playing a massively multiplayer online game (MMOG) on your web browser. The game company would probably host the backend servers in the cloud to allow massive scalability, so the processing power is not an issue. Now suppose all these servers were provisioned in the eastern United States. Gamers in New York would have no problem enjoying the game, but those in Japan would probably have noticeable network latency issues because every one of their commands would have to be sent literally around the world to be processed by the U.S. servers, and then the resulting graphics sent back around the world to the player in Japan. That player would probably lose interest in the game really quickly. Now, suppose that the company kept its main servers in the United States but provisioned regional servers, with one of them in, say, Singapore. Most of the commands are processed in the regional server, which means that the user experience of players in Japan is a lot better, while the global leaderboard is maintained centrally in the United States. This is an example of edge computing.

Edge computing is an evolution of content distribution networks (CDNs), which were designed to bring web content closer to its clients. CDNs helped with internationalization of websites but were also very good for mitigating the effects of DDoS attacks. *Edge computing* is a distributed system in which some computational and data storage assets are deployed close to where they are needed in order to reduce latency and network traffic. As shown in Figure 7-7, an edge computing architecture typically has three layers: end devices, edge devices, and cloud infrastructure. The end devices can be anything from smart thermometers to self-driving cars. They have a requirement for processing data in real time, which means there are fairly precise time constraints. Think of a thermal sensor in one of your data centers and how you would need to have an alarm within minutes (at most) of it detecting rising or excessive heat.

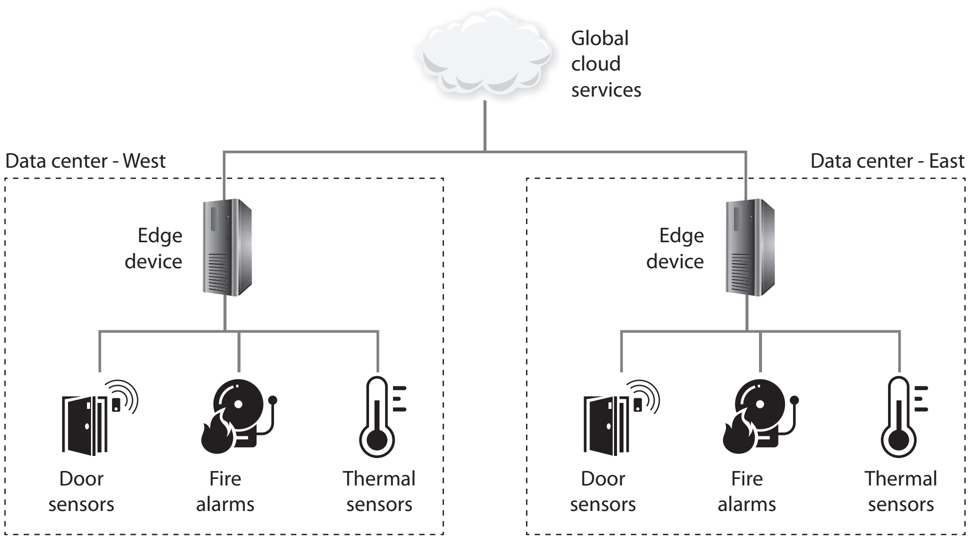


Figure 7-7 A sample edge computing architecture for facility management

To reduce the turnaround time for these computing requirements, we deploy edge devices that are closer to, and in some cases embedded within, the end devices. Returning to the thermometer example, suppose you have several of these devices in each of your two data centers. You also have a multitude of other sensors such as fire alarms and door sensors. Rather than configuring an alarm to sound whenever the data center gets too hot, you integrate all these sensors to develop an understanding of what is going in the facility. For example, maybe the temperature is rising because someone left the back door open on a hot summer day. If it keeps going up, you want to sound a door alarm, not necessarily a temperature alarm, and do it while there is still time for the cooling system to keep the ambient temperature within tolerance. The sensors (including the thermometer) would send their data to the edge device, which is located near or in the same facility. This reduces the time needed to compute solutions and also provides a degree of protection against network outages. The determination to sound the door alarm (and when) is made there, locally, at the edge device. All (or maybe some of) the data from all the sensors at both data centers is also sent to the global cloud services infrastructure. There, we can take our time and run data analytics to discover useful patterns that could tell us how to be more efficient in how we use our resources around the world.

NOTE As increased computing power finds its way into IoT devices, these too are becoming edge devices in some cases.

Chapter Review

Central to securing our systems is understanding their components and how they interact with each other—in other words, their architectures. While it may seem that architectural terminology overlaps quite a bit, in reality each approach brings some unique challenges and some not-so-unique challenges. As security professionals, we need to understand where architectures are similar and where they differ. We can mix and match, of course, but must also do so with a clear understanding of the underlying issues. In this chapter, we've classified the more common system architectures and discussed what makes them unique and what specific security challenges they pose. Odds are that you will encounter devices and systems in most, if not all, of the architectures we've covered here.

Quick Review

- Client-based systems execute all their core functions on the user's device and don't require network connectivity.
- Server-based systems require that a client make requests from a server across a network connection.
- Transactions are sequences of actions required to properly change the state of a database.
- Database transactions must be atomic, consistent, isolated, and durable (ACID).
- Aggregation is the act of combining information from separate sources and is a security problem when it allows unauthorized individuals to piece together sensitive information.
- Inference is deducing a whole set of information from a subset of its aggregated components. This is a security problem when it allows unauthorized individuals to infer sensitive information.
- High-performance computing (HPC) is the aggregation of computing power in ways that exceed the capabilities of general-purpose computers for the specific purpose of solving large problems.
- Industrial control systems (ICS) consist of information technology that is specifically designed to control physical devices in industrial processes.
- Any system in which computers and physical devices collaborate via the exchange of inputs and outputs to accomplish a task or objective is an embedded or cyber-physical system.
- The two main types of ICS are distributed control systems (DCS) and supervisory control and data acquisition (SCADA) systems. The main difference between them is that a DCS controls local processes while SCADA is used to control things remotely.
- ICS should always be logically or physically isolated from public networks.
- Virtualized systems are those that exist in software-simulated environments.
- Virtual machines (VMs) are systems in which the computing hardware has been virtualized for the operating systems running in them.

- Containers are systems in which the operating systems have been virtualized for the applications running in them.
- Microservices are software architectures in which features are divided into multiple separate components that work together in a distributed manner across a network.
- Containers and microservices don't have to be used together but it's very common to do so.
- In a serverless architecture, the services offered to end users can be performed without a requirement to set up any dedicated server infrastructure.
- Cloud computing is the use of shared, remote computing devices for the purpose of providing improved efficiencies, performance, reliability, scalability, and security.
- Software as a Service (SaaS) is a cloud computing model that provides users access to a specific application that executes in the service provider's environment.
- Platform as a Service (PaaS) is a cloud computing model that provides users access to a computing platform but not to the operating system or to the virtual machine on which it runs.
- Infrastructure as a Service (IaaS) is a cloud computing model that provides users unfettered access to a cloud device, such as an instance of a server, which includes both the operating system and the virtual machine on which it runs.
- An embedded system is a self-contained, typically ruggedized, computer system with its own processor, memory, and input/output devices that is designed for a very specific purpose.
- The Internet of Things (IoT) is the global network of connected embedded systems.
- A distributed system is a system in which multiple computing nodes, interconnected by a network, exchange information for the accomplishment of collective tasks.
- Edge computing is a distributed system in which some computational and data storage assets are deployed close to where they are needed in order to reduce latency and network traffic.

Questions

Please remember that these questions are formatted and asked in a certain way for a reason. Keep in mind that the CISSP exam is asking questions at a conceptual level. Questions may not always have the perfect answer, and the candidate is advised against always looking for the perfect answer. Instead, the candidate should look for the best answer in the list.

1. Which of the following lists two foundational properties of database transactions?
   - **A.** Aggregation and inference
   - **B.** Scalability and durability
   - **C.** Consistency and performance
   - **D.** Atomicity and isolation

2. Which of the following is *not* true about containers?
   - **A.** They are embedded systems.
   - **B.** They are virtualized systems.
   - **C.** They commonly house microservices.
   - **D.** They operate in a sandbox.
3. What is the term that describes a database attack in which an unauthorized user is able to combine information from separate sources to learn sensitive information to which the user should not have access?
   - **A.** Aggregation
   - **B.** Containerization
   - **C.** Serialization
   - **D.** Collection
4. What is the main difference between a distributed control system (DCS) and supervisory control and data acquisition (SCADA)?
   - **A.** SCADA is a type of industrial control system (ICS), while a DCS is a type of bus.
   - **B.** SCADA controls systems in close proximity, while a DCS controls physically distant ones.
   - **C.** A DCS controls systems in close proximity, while SCADA controls physically distant ones.
   - **D.** A DCS uses programmable logic controllers (PLCs), while SCADA uses remote terminal units (RTUs).
5. What is the main purpose of a hypervisor?
   - **A.** Virtualize hardware resources and manage virtual machines
   - **B.** Virtualize the operating system and manage containers
   - **C.** Provide visibility into virtual machines for access control and logging
   - **D.** Provide visibility into containers for access control and logging
6. Which cloud service model provides customers direct access to hardware, the network, and storage?
   - **A.** SaaS
   - **B.** PaaS
   - **C.** IaaS
   - **D.** FaaS
7. Which cloud service model do you recommend to enable access to developers to write custom code while also providing all employees access from remote offices?
   - **A.** PaaS
   - **B.** SaaS

C. FaaS

D. IaaS

8. Which of the following is not a major issue when securing embedded systems?

A. Use of proprietary code

B. Devices that "phone home"

C. Lack of microcontrollers

D. Ability to update and patch them securely

9. Which of the following is true about edge computing?

A. Uses no centralized computing resources, pushing all computation to the edge

B. Pushes computation to the edge while retaining centralized data management

C. Typically consists of two layers: end devices and cloud infrastructure

D. Is an evolution of content distribution networks

Use the following scenario to answer Questions 10–12. You were just hired as director of cybersecurity for an electric power company with facilities around your country. Carmen is the director of operations and offers to give you a tour so you can see the security measures that are in place on the operational technology (OT).

10. What system would be used to control power generation, distribution, and delivery to all your customers?

A. Supervisory control and data acquisition (SCADA)

B. Distributed control system (DCS)

C. Programmable logic controller

D. Edge computing system

11. You see a new engineer being coached remotely by a more senior member of the staff in the use of the human-machine interface (HMI). Carmen tells you that senior engineers are allowed to access the HMI from their personal computers at home to facilitate this sort of impromptu training. She asks what you think of this policy. How should you respond?

A. Change the policy. They should not access the HMI with their personal computers, but they could do so using a company laptop, assuming they also use a virtual private network (VPN).

B. Change the policy. ICS devices should always be isolated from the Internet.

C. It is acceptable because the HMI is only used for administrative purposes and not operational functions.

D. It is acceptable because safety is the fundamental concern in ICS, so it is best to let the senior engineers be available to train other staff from home.

12. You notice that several ICS devices have never been patched. When you ask why, Carmen tells you that those are mission-critical devices, and her team has no way of testing the patches before patching these production systems. Fearing that patching them could cause unexpected outages or, worse, injure someone, she has authorized them to remain as they are. Carmen asks whether you agree. How could you respond?

  A. Yes. As long as we document the risk and ensure the devices are as isolated and as closely monitored as possible.

  B. Yes. Safety and availability trump all other concerns when it comes to ICS security.

  C. No. You should stand up a testing environment so you can safely test the patches and then deploy them to all devices.

  D. No. These are critical devices and should be patched as soon as possible.

Answers

1. **D.** The foundational properties of database transactions are atomicity, consistency, isolation, and durability (ACID).
2. **A.** Containers are virtualized systems that commonly (though not always) house microservices and run in sandboxes. It would be highly unusual to implement a container as an embedded system.
3. **A.** Aggregation happens when a user does not have the clearance or permission to access specific information, but she does have the permission to access components of this information. She can then figure out the rest and obtain restricted information.
4. **C.** The main difference is that a DCS controls devices within fairly close proximity, while SCADA controls large-scale physical processes involving nodes separated by significant distances. They both can (and frequently use) PLCs, but RTUs are almost always seen in SCADA systems.
5. **A.** Hypervisors are almost always used to virtualize the hardware on which virtual machines run. They can also provide visibility and logging, but these are secondary functions. Containers are the equivalents of hypervisors, but they work at a higher level by virtualizing the operating system.
6. **C.** Infrastructure as a Service (IaaS) offers an effective and affordable way for organizations to get all the benefits of managing their own hardware without the massive overhead costs associated with acquisition, physical storage, and disposal of the hardware.
7. **A.** Platform as a Service (PaaS) solutions are optimized to provide value focused on software development, offering direct access to a development environment to enable an organization to build its own solutions on the cloud infrastructure, rather than providing its own infrastructure.

8. **C.** Embedded systems are usually built around microcontrollers, which are specialized devices that consist of a CPU, memory, and peripheral control interfaces. All the other answers are major issues in securing embedded systems.
9. **D.** Edge computing is an evolution of content distribution networks, which were designed to bring web content closer to its clients. It is a distributed system in which some computational and data storage assets are deployed close to where they are needed in order to reduce latency and network traffic. Accordingly, some computing and data management is handled in each of three different layers: end devices, edge devices, and cloud infrastructure.
10. **A.** SCADA was designed to control large-scale physical processes involving nodes separated by significant distances, as is the case with electric power providers.
11. **B.** It is a best practice to completely isolate ICS devices from Internet access. Sometimes this is not possible for operational reasons, so remote access through a VPN could be allowed even though it is not ideal.
12. **A.** It is all too often the case that organizations can afford neither the risk of pushing untested patches to ICS devices nor the costs of standing up a testing environment. In these conditions, the best strategy is to isolate and monitor the devices as much as possible.

CHAPTER 8

Cryptology

This chapter presents the following:

- Principles of cryptology
- Symmetric cryptography
- Asymmetric cryptography
- Public key infrastructure
- Cryptanalytic attacks

Three can keep a secret, if two of them are dead.

—Benjamin Franklin

Now that you have a pretty good understanding of system architectures from Chapter 7, we turn to a topic that is central to protecting these architectures. *Cryptography* is the practice of storing and transmitting information in a form that only authorized parties can understand. Properly designed and implemented, cryptography is an effective way to protect sensitive data throughout its life cycle. However, with enough time, resources, and motivation, hackers can successfully attack most cryptosystems and reveal the information. So, a more realistic goal of cryptography is to make obtaining the information too work intensive or time consuming to be worthwhile to the attacker.

Cryptanalysis is the name collectively given to techniques that aim to weaken or defeat cryptography. This is what the adversary attempts to do to thwart the defender's use of cryptography. Together, cryptography and cryptanalysis comprise *cryptology*. In this chapter, we'll take a good look at both sides of this topic. This is an important chapter in the book, because we can't defend our information systems effectively without understanding applied cryptology.

The History of Cryptography

Cryptography has roots in antiquity. Around 600 B.C., Hebrews invented a cryptographic method called *atbash* that required the alphabet to be flipped so each letter in the original message was mapped to a different letter in the flipped, or shifted, message. An example of an encryption key used in the atbash encryption scheme is shown here:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ZYXWVUTSRQPONMLKJIHGFEDCBA
```

If you want to encrypt the word "security" you would instead use "hvxfirgb." Atbash is an example of a *substitution cipher* because each character is replaced with another character. This type of substitution cipher is referred to as a *monoalphabetic substitution cipher* because it uses only one alphabet, whereas a *polyalphabetic substitution cipher* uses multiple alphabets.

TIP Cipher is another term for algorithm.

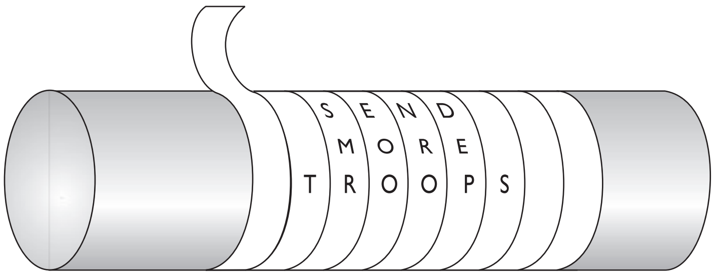
Around 400 B.C., the Spartans used a system of encrypting information in which they would write a message on a sheet of papyrus (a type of paper) that was wrapped around a staff (a stick or wooden rod), which was then delivered and wrapped around a different staff by the recipient. The message was only readable if it was wrapped around the correct size staff, which made the letters properly match up, as shown in Figure 8-1. When the papyrus was not wrapped around the staff, the writing appeared as just a bunch of random characters. This approach, known as the *scytale cipher*, is an example of a *transposition cipher* because it relies on changing the sequence of the characters to obscure their meaning. Only someone who knows how to rearrange them would be able to recover the original message.

Later, in Rome, Julius Caesar (100–44 B.C.) developed a simple method of shifting letters of the alphabet, similar to the atbash scheme. He simply shifted the alphabet by three positions. The following example shows a standard alphabet and a shifted alphabet. The alphabet serves as the algorithm, and the key is the number of locations it has been shifted during the encryption and decryption process.

- **Standard alphabet:**
  `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
- **Cryptographic alphabet:**
  `DEFGHIJKLMNOPQRSTUVWXYZABC`

As an example, suppose we need to encrypt the message "MISSION ACCOMPLISHED." We take the first letter of this message, *M*, and shift up three locations within the alphabet. The encrypted version of this first letter is *P*, so we write

Figure 8-1 The scytale was used by the Spartans to decipher encrypted messages.



that down. The next letter to be encrypted is *I*, which matches *L* when we shift three spaces. We continue this process for the whole message. Once the message is encrypted, a carrier takes the encrypted version to the destination, where the process is reversed.

- **Original message:**
  `MISSION ACCOMPLISHED`
- **Encrypted message:**
  `PLVVLRQ DFFRPSOLVKHG`

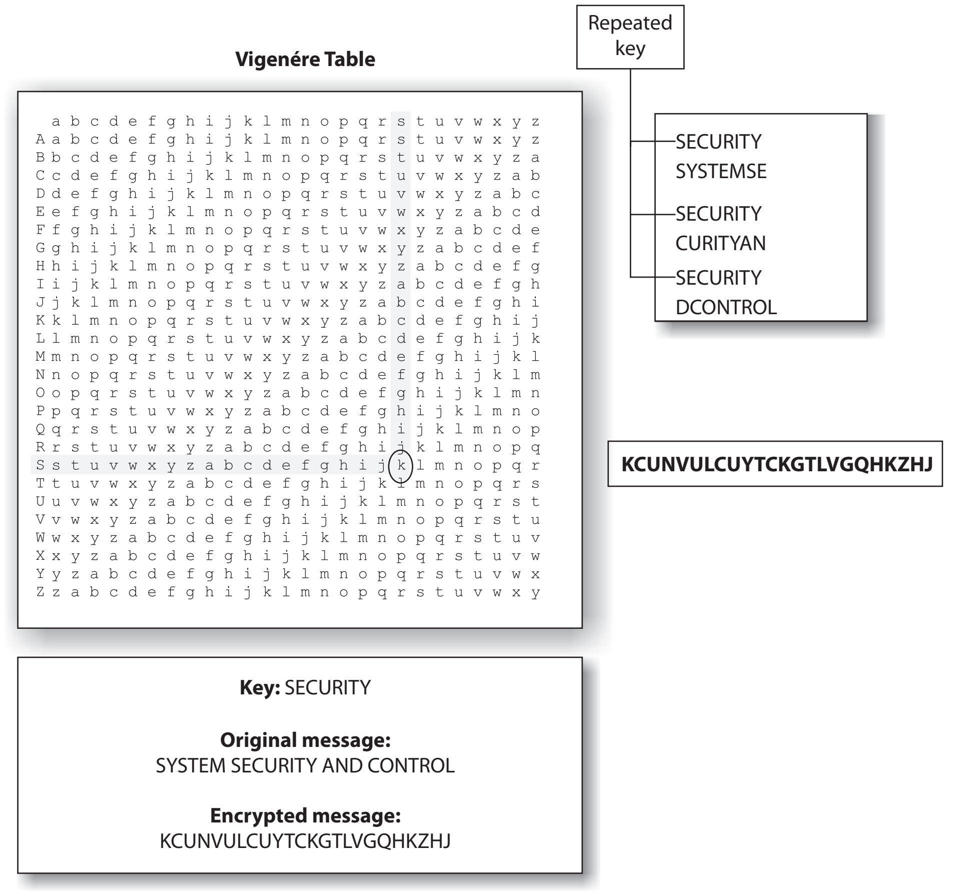
Today, this technique seems too simplistic to be effective, but in the time of Julius Caesar, not very many people could read in the first place, so it provided a high level of protection. The Caesar cipher, like the atbash cipher, is an example of a monoalphabetic cipher. Once more people could read and reverse-engineer this type of encryption process, the cryptographers of that day increased the complexity by creating polyalphabetic ciphers.

In the 16th century in France, Blaise de Vigenère developed a polyalphabetic substitution cipher for Henry III. This was based on the Caesar cipher, but it increased the difficulty of the encryption and decryption process. As shown in Figure 8-2, we have a message that needs to be encrypted, which is SYSTEM SECURITY AND CONTROL. We have a key with the value of SECURITY. We also have a Vigenère table, or algorithm, which is really the Caesar cipher on steroids. Whereas the Caesar cipher used a single shift alphabet (letters were shifted up three places), the Vigenère cipher has 27 shift alphabets and the letters are shifted up only one place.

So, looking at the example in Figure 8-2, we take the first value of the key, *S*, and starting with the first alphabet in our algorithm, trace over to the *S* column. Then we look at the first character of the original message that needs to be encrypted, which is *S*, and go down to the *S* row. We follow the column and row and see that they intersect on the value *K*. That is the first encrypted value of our message, so we write down *K*. Then we go to the next value in our key, which is *E*, and the next character in the original message, which is *Y*. We see that the *E* column and the *Y* row intersect at the cell with the value of *C*. This is our second encrypted value, so we write that down. We continue this process for the whole message (notice that the key repeats itself, since the message is longer than the key). The result is an encrypted message that is sent to the destination. The destination must have the same algorithm (Vigenère table) and the same key (SECURITY) to properly reverse the process to obtain a meaningful message.

The evolution of cryptography continued as countries refined it using new methods, tools, and practices with varying degrees of success. Mary, Queen of Scots, lost her life in the 16th century when an encrypted message she sent was intercepted. During the American Revolutionary War, Benedict Arnold used a codebook cipher to exchange information on troop movement and strategic military advancements. By the late 1800s, cryptography was commonly used in the methods of communication between military factions.

During World War II, encryption devices were used for tactical communication, which drastically improved with the mechanical and electromechanical technology that provided the world with telegraphic and radio communication. The rotor cipher machine, which is a device that substitutes letters using different rotors within the



	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
A	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
B	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a
C	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b
D	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
E	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d
F	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e
G	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f
H	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g
I	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h
J	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i
K	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j
L	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k
M	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l
N	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m
O	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n
P	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
Q	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
R	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
S	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
T	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
U	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
V	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
W	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
X	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
Y	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x
Z	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y

Key: SECURITY

Original message:
SYSTEM SECURITY AND CONTROL

Encrypted message:
KCUNVULCUYTCKGTLVGQHKZHJ

Figure 8-2 Polyalphabetic algorithms were developed to increase encryption complexity.

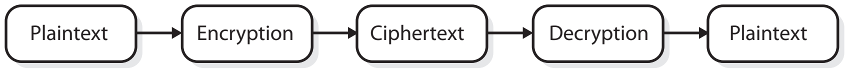
machine, was a huge breakthrough in military cryptography that provided complexity that proved difficult to break. This work gave way to the most famous cipher machine in history to date: Germany's *Enigma* machine. The Enigma machine had separate rotors, a plug board, and a reflecting rotor.

The originator of the message would configure the Enigma machine to its initial settings before starting the encryption process. The operator would type in the first letter of the message, and the machine would substitute the letter with a different letter and present it to the operator. This encryption was done by moving the rotors a predefined number of times. So, if the operator typed in a *T* as the first character, the Enigma machine might present an *M* as the substitution value. The operator would write down the letter *M* on his sheet. The operator would then advance the rotors and enter the next letter. Each time a new letter was to be encrypted, the operator would advance the rotors to a new setting. This process was followed until the whole message was encrypted. Then the encrypted text was transmitted over the airwaves, most likely to a German U-boat. The chosen substitution for each letter was dependent upon the rotor setting, so the crucial and secret part of this process (the key) was the initial setting and how the operators advanced the rotors when encrypting and decrypting a message. The operators at each end needed to know this sequence of increments to advance each rotor in order to enable the German military units to properly communicate.

When computers were invented, the possibilities for encryption methods and devices expanded exponentially and cryptography efforts increased dramatically. This era brought unprecedented opportunity for cryptographic designers to develop new encryption techniques. A well-known and successful project was *Lucifer*, which was developed at IBM. Lucifer introduced complex mathematical equations and functions that were later adopted and modified by the U.S. National Security Agency (NSA) to establish the U.S. Data Encryption Standard (DES) in 1976, a federal government standard. DES was used worldwide for financial and other transactions, and was embedded into numerous commercial applications. Though it was cracked in the late 1990s and is no longer considered secure, DES represented a significant advancement for cryptography. It was replaced a few years later by the Advanced Encryption Standard (AES), which continues to protect sensitive data to this day.

Cryptography Definitions and Concepts

Encryption is a method of transforming readable data, called *plaintext*, into a form that appears to be random and unreadable, which is called *ciphertext*. Plaintext is in a form that can be understood either by a person (a document) or by a computer (executable code). Once plaintext is transformed into ciphertext, neither human nor machine can properly process it until it is decrypted. This enables the transmission of confidential information over insecure channels without unauthorized disclosure. When sensitive data is stored on a computer, it is usually protected by logical and physical access controls. When this same sensitive information is sent over a network, it no longer has the advantage of these controls and is in a much more vulnerable state.



A system or product that provides encryption and decryption is referred to as a *cryptosystem* and can be created through hardware components or program code in an application. The cryptosystem uses an encryption algorithm (which determines how simple or complex the encryption process will be), keys, and the necessary software components and protocols. Most algorithms are complex mathematical formulas that are applied in a specific sequence to the plaintext. Most encryption methods use a secret value called a key (usually a long string of bits), which works with the algorithm to encrypt and decrypt the text.

The *algorithm*, the set of rules also known as the *cipher*, dictates how enciphering and deciphering take place. Many of the mathematical algorithms used in computer systems today are publicly known and are not the secret part of the encryption process. If the internal mechanisms of the algorithm are not a secret, then something must be: the key.

A common analogy used to illustrate this point is the use of locks you would purchase from your local hardware store. Let's say 20 people bought the same brand of lock. Just because these people share the same type and brand of lock does not mean they can now unlock each other's doors and gain access to their private possessions. Instead, each lock comes with its own key, and that one key can open only that one specific lock.

In encryption, the *key* (also known as *cryptovariable*) is a value that comprises a large sequence of random bits. Is it just any random number of bits crammed together? Not really. An algorithm contains a *keyspace*, which is a range of values that can be used to construct a key. When the algorithm needs to generate a new key, it uses random values from this keyspace. The larger the keyspace, the more available values that can be used to represent different keys—and the more random the keys are, the harder it is for intruders to figure them out. For example, if an algorithm allows a key length of 2 bits, the keyspace for that algorithm would be 4, which indicates the total number of different keys that would be possible. (Remember that we are working in binary and that 2^2 equals 4.) That would not be a very large keyspace, and certainly it would not take an attacker very long to find the correct key that was used.

A large keyspace allows for more possible keys. (Today, we are commonly using key sizes of 128, 256, 512, or even 1,024 bits and larger.) So a key size of 512 bits would provide 2^{512} possible combinations (the keyspace). The encryption algorithm should use the entire keyspace and choose the values to make up the keys as randomly as possible. If a smaller keyspace were used, there would be fewer values to choose from when generating a key, as shown in Figure 8-3. This would increase an attacker's chances of figuring out the key value and deciphering the protected information.

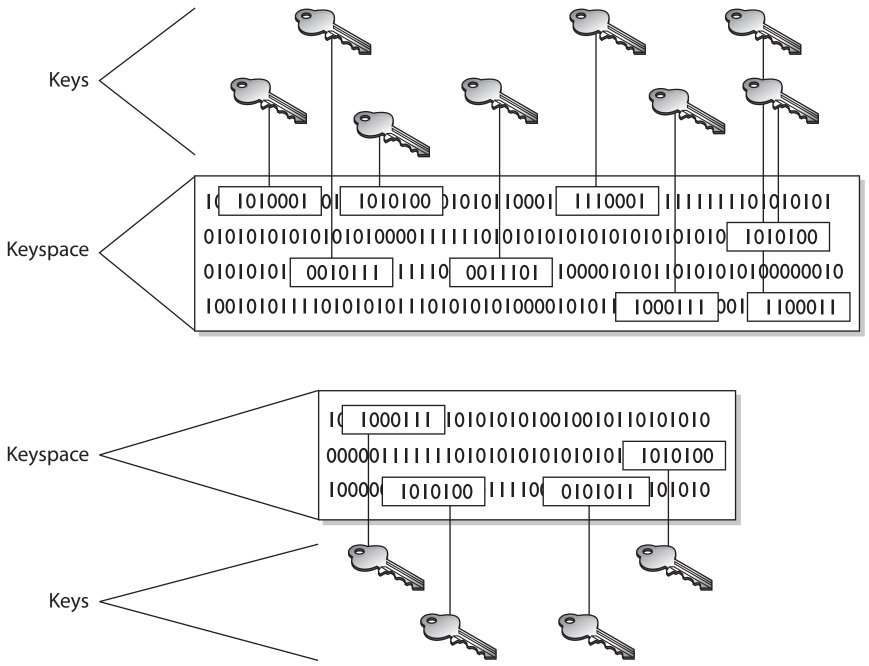


Figure 8-3 Larger keyspaces permit a greater number of possible key values.

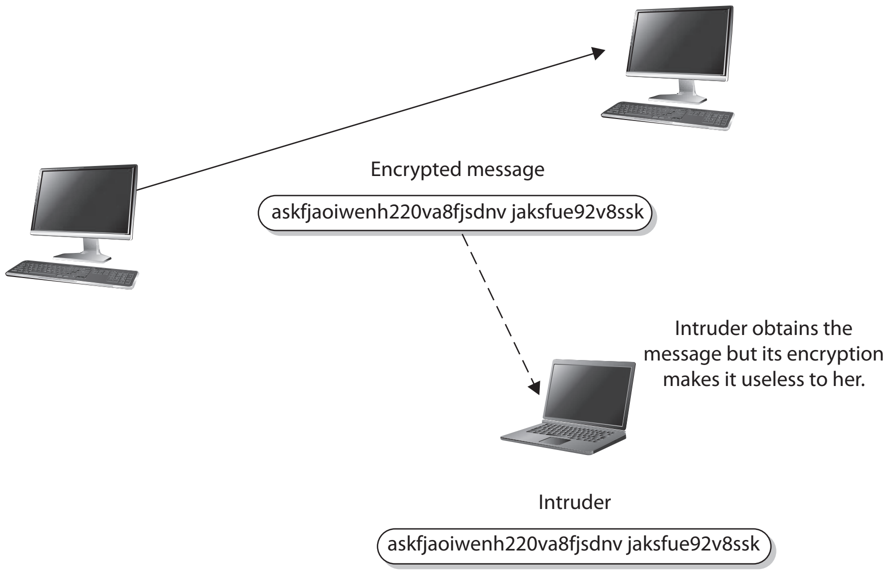

Encrypted message
askfjaoiwenh220va8fjsdnv jaksfue92v8ssk
Intruder obtains the message but its encryption makes it useless to her.
Intruder
askfjaoiwenh220va8fjsdnv jaksfue92v8ssk

Figure 8-4 Without the right key, the captured message is useless to an attacker.

If an eavesdropper captures a message as it passes between two people, she can view the message, but it appears in its encrypted form and is therefore unusable. Even if this attacker knows the algorithm that the two people are using to encrypt and decrypt their information, without the key, this information remains useless to the eavesdropper, as shown in Figure 8-4.

Cryptosystems

A *cryptosystem* encompasses all of the necessary components for encryption and decryption to take place. Pretty Good Privacy (PGP) is just one example of a cryptosystem. A cryptosystem is made up of at least the following:

- Software
- Protocols
- Algorithms
- Keys

Cryptosystems can provide the following services:

- **Confidentiality** Renders the information unintelligible except by authorized entities.
- **Integrity** Ensures that data has not been altered in an unauthorized manner since it was created, transmitted, or stored.
- **Authentication** Verifies the identity of the user or system that created the information.

- **Authorization** Provides access to some resource to the authenticated user or system.
- **Nonrepudiation** Ensures that the sender cannot deny sending the message.

As an example of how these services work, suppose your boss sends you an e-mail message stating that you will be receiving a raise that doubles your salary. The message is encrypted, so you can be sure it really came from your boss (authenticity), that someone did not alter it before it arrived at your computer (integrity), that no one else was able to read it as it traveled over the network (confidentiality), and that your boss cannot deny sending it later when he comes to his senses (nonrepudiation).

Different types of messages and transactions require higher or lower degrees of one or all of the services that cryptography methods can supply. Military and intelligence agencies are very concerned about keeping information confidential, so they would choose encryption mechanisms that provide a high degree of secrecy. Financial institutions care about confidentiality, but they also care about the integrity of the data being transmitted, so the encryption mechanism they would choose may differ from the military's encryption methods. If messages were accepted that had a misplaced decimal point or zero, the ramifications could be far reaching in the financial world. Legal agencies may care most about the authenticity of the messages they receive. If information received ever needed to be presented in a court of law, its authenticity would certainly be questioned; therefore, the encryption method used must ensure authenticity, which confirms who sent the information.

NOTE If David sends a message and then later claims he did not send it, this is an act of repudiation. When a cryptography mechanism provides nonrepudiation, the sender cannot later deny he sent the message (well, he can try to deny it, but the cryptosystem proves otherwise).

The types and uses of cryptography have increased over the years. At one time, cryptography was mainly used to keep secrets secret (confidentiality), but today we use cryptography to ensure the integrity of data, to authenticate messages, to confirm that a message was received, to provide access control, and much more.

Kerckhoffs' Principle

Auguste Kerckhoffs published a paper in 1883 stating that the only secrecy involved with a cryptography system should be the key. He claimed that the algorithm should be publicly known. He asserted that if security were based on too many secrets, there would be more vulnerabilities to possibly exploit.

So, why do we care what some guy said almost 140 years ago? Because this debate is still going on. Cryptographers in certain sectors agree with *Kerckhoffs' principle*, because making an algorithm publicly available means that many more people can view the source code, test it, and uncover any type of flaws or weaknesses. It is the attitude of "many heads are better than one." Once someone uncovers some type of flaw, the developer can fix the issue and provide society with a much stronger algorithm.

But not everyone agrees with this philosophy. Governments around the world create their own algorithms that are not released to the public. Their stance is that if a smaller number of people know how the algorithm actually works, then a smaller number of people will know how to possibly break it. Cryptographers in the private sector do not agree with this practice and do not commonly trust algorithms they cannot examine. It is basically the same as the open-source versus compiled software debate that is in full force today.

The Strength of the Cryptosystem

The *strength* of an encryption method comes from the algorithm, the secrecy of the key, the length of the key, and how they all work together within the cryptosystem. When strength is discussed in encryption, it refers to how hard it is to figure out the algorithm or key, whichever is not made public. Attempts to break a cryptosystem usually involve processing an amazing number of possible values in the hopes of finding the one value (key) that can be used to decrypt a specific message. The strength of an encryption method correlates to the amount of necessary processing power, resources, and time required to break the cryptosystem or to figure out the value of the key.

Breaking a cryptosystem can be accomplished by a *brute-force attack*, which means trying every possible key value until the resulting plaintext is meaningful. Depending on the algorithm and length of the key, this can be an easy task or one that is close to impossible. If a key can be broken with an Intel Core i5 processor in three hours, the cipher is not strong at all. If the key can only be broken with the use of a thousand multiprocessing systems over 1.2 million years, then it is pretty darned strong. The introduction of commodity cloud computing has really increased the threat of brute-force attacks.

The goal when designing an encryption method is to make compromising it too expensive or too time consuming. Another name for cryptography strength is *work factor*, which is an estimate of the effort and resources it would take an attacker to penetrate a cryptosystem.

Even if the algorithm is very complex and thorough, other issues within encryption can weaken encryption methods. Because the key is usually the secret value needed to actually encrypt and decrypt messages, improper protection of the key can weaken the encryption. Even if a user employs an algorithm that has all the requirements for strong encryption, including a large keyspace and a large and random key value, if she shares her key with others, the strength of the algorithm becomes almost irrelevant.

Important elements of encryption are to use an algorithm without flaws, use a large key size, use all possible values within the keyspace selected as randomly as possible, and protect the actual key. If one element is weak, it could be the link that dooms the whole process.

One-Time Pad

A *one-time pad* is a perfect encryption scheme because it is considered unbreakable if implemented properly. It was invented by Gilbert Vernam in 1917, so sometimes it is referred to as the Vernam cipher.

This cipher does not use shift alphabets, as do the Caesar and Vigenère ciphers discussed earlier, but instead uses a pad made up of random values, as shown in Figure 8-5. Our plaintext message that needs to be encrypted has been converted into bits, and our one-time

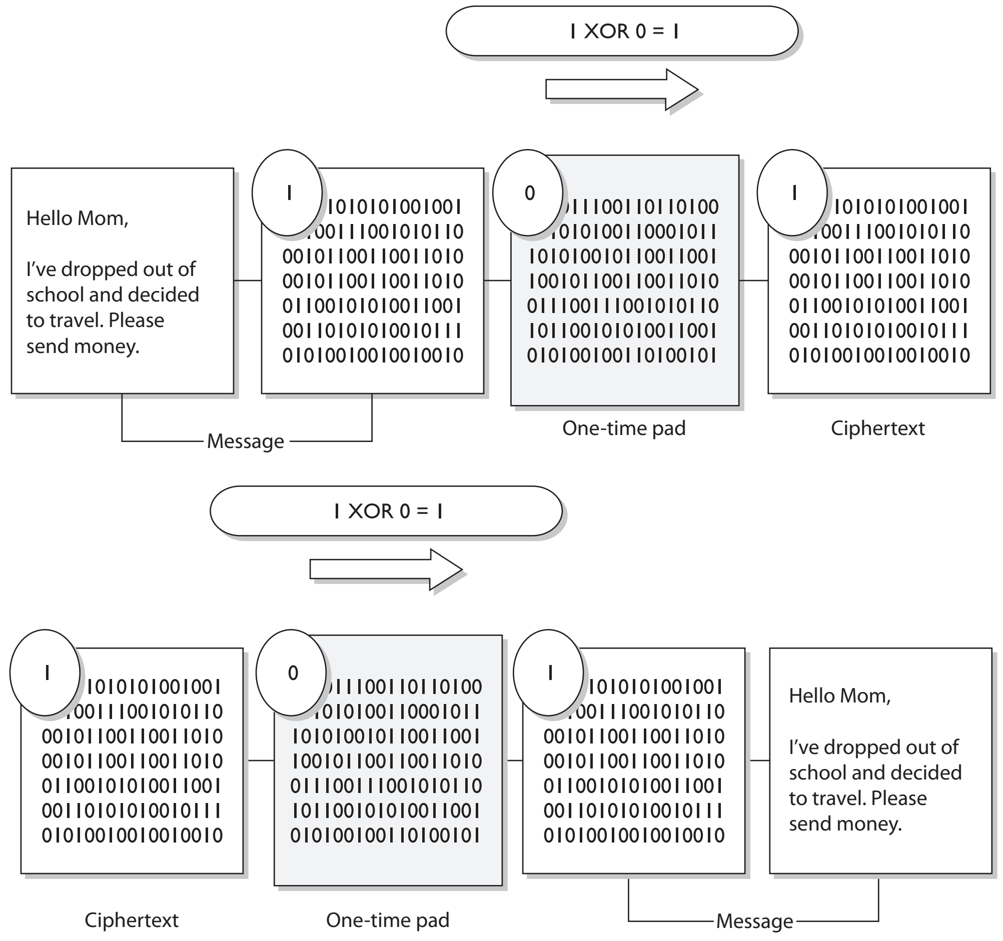

1 XOR 0 = 1
Hello Mom,
I've dropped out of school and decided to travel. Please send money.
1
0
1
Message
One-time pad
Ciphertext
1 XOR 0 = 1
1
0
1
Hello Mom,
I've dropped out of school and decided to travel. Please send money.
Ciphertext
One-time pad
Message

Figure 8-5 A one-time pad

pad is made up of random bits. This encryption process uses a binary mathematic function called exclusive-OR, usually abbreviated as XOR.

XOR is an operation that is applied to 2 bits and is a function commonly used in binary mathematics and encryption methods. When combining the bits, if both values are the same, the result is 0 (1 XOR 1 = 0). If the bits are different from each other, the result is 1 (1 XOR 0 = 1). For example:

Message stream:	1	0	0	1	0	1	0	1	1	1
Keystream:	0	0	1	1	1	0	1	0	1	0
Ciphertext stream:	1	0	1	0	1	1	1	1	0	1

So in our example, the first bit of the message is XORed to the first bit of the one-time pad, which results in the ciphertext value 1. The second bit of the message is XORed with the second bit of the pad, which results in the value 0. This process continues until the whole message is encrypted. The result is the encrypted message that is sent to the receiver.

In Figure 8-5, we also see that the receiver must have the same one-time pad to decrypt the message by reversing the process. The receiver takes the first bit of the encrypted message and XORs it with the first bit of the pad. This results in the plaintext value. The receiver continues this process for the whole encrypted message until the entire message is decrypted.

The one-time pad encryption scheme is deemed unbreakable only if the following things are true about the implementation process:

- *The pad must be used only one time.* If the pad is used more than one time, this might introduce patterns in the encryption process that will aid the eavesdropper in his goal of breaking the encryption.
- *The pad must be at least as long as the message.* If it is not as long as the message, the pad will need to be reused to cover the whole message. This would be the same thing as using a pad more than one time, which could introduce patterns.
- *The pad must be securely distributed and protected at its destination.* This is a very cumbersome process to accomplish, because the pads are usually just individual pieces of paper that need to be delivered by a secure courier and properly guarded at each destination.
- *The pad must be made up of truly random values.* This may not seem like a difficult task, but even our computer systems today do not have truly random number generators; rather, they have pseudorandom number generators.

NOTE Generating truly random numbers is very difficult. Most systems use an algorithmic *pseudorandom number generator (PRNG)* that takes as its input a seed value and creates a stream of pseudorandom values from it. Given the same seed, a PRNG generates the same sequence of values. Truly random numbers must be based on natural phenomena such as thermal noise and quantum mechanics.

Although the one-time pad approach to encryption can provide a very high degree of security, it is impractical in most situations because of all of its different requirements. Each possible pair of entities that might want to communicate in this fashion must receive, in a secure fashion, a pad that is as long as, or longer than, the actual message. This type of key management can be overwhelming and may require more overhead than it is worth. The distribution of the pad can be challenging, and the sender and receiver must be perfectly synchronized so each is using the same pad.

EXAM TIP The one-time pad, though impractical for most modern applications, is the only perfect cryptosystem.

One-Time Pad Requirements

For a one-time pad encryption scheme to be considered unbreakable, each pad in the scheme must be

- Made up of truly random values
- Used only one time
- Securely distributed to its destination
- Secured at sender's and receiver's sites
- At least as long as the message

Cryptographic Life Cycle

Since most of us will probably not be using one-time pads (the only "perfect" system) to defend our networks, we have to consider that cryptography, like a fine steak, has a limited shelf life. Given enough time and resources, any cryptosystem can be broken, either through analysis or brute force. The *cryptographic life cycle* is the ongoing process of identifying your cryptography needs, selecting the right algorithms, provisioning the needed capabilities and services, and managing keys. Eventually, you determine that your cryptosystem is approaching the end of its shelf life and you start the cycle all over again.

How can you tell when your algorithms (or choice of keyspaces) are about to go stale? You need to stay up to date with the cryptologic research community. They are the best source for early warning that things are going sour. Typically, research papers postulating weaknesses in an algorithm are followed by academic exercises in breaking the algorithm under controlled conditions, which are then followed by articles on how it is broken in general cases. When the first papers come out, it is time to start looking for replacements.

Cryptographic Methods

By far, the most commonly used cryptographic methods today are *symmetric key cryptography*, which uses symmetric keys (also called secret keys), and *asymmetric key cryptography*, which uses two different, or asymmetric, keys (also called public and private keys). Asymmetric key cryptography is also called *public key cryptography* because one of its keys can be made public. As we will see shortly, public key cryptography typically uses powers of prime numbers for encryption and decryption. A variant of this approach uses elliptic curves, which allows much smaller keys to be just as secure and is (unsurprisingly) called *elliptic curve cryptography (ECC)*. Though you may not know it, it is likely that you've used ECC at some point to communicate securely on the Web. (More on that later.) Though these three cryptographic methods are considered secure today (given that you use good keys), the application of quantum computing to cryptology could dramatically change this situation. The following sections explain the key points of these four methods of encryption.

Symmetric Key Cryptography

In a cryptosystem that uses symmetric key cryptography, the sender and receiver use two instances of the same key for encryption and decryption, as shown in Figure 8-6. So the key has dual functionality in that it can carry out both encryption and decryption processes. Symmetric keys are also called *secret* keys, because this type of encryption relies on each user to keep the key a secret and properly protected. If an intruder were to get this key, he could decrypt any intercepted message encrypted with it.

Each pair of users who want to exchange data using symmetric key encryption must have two instances of the same key. This means that if Dan and Iqqi want to communicate, both need to obtain a copy of the same key. If Dan also wants to communicate using symmetric encryption with Norm and Dave, he needs to have three separate keys, one for each friend. This might not sound like a big deal until Dan realizes that he may communicate with hundreds of people over a period of several months, and keeping track and using the correct key that corresponds to each specific receiver can become a daunting task. If 10 people needed to communicate securely with each other using symmetric keys, then 45 keys would need to be kept track of. If 100 people were going to communicate, then 4,950 keys would be involved. The equation used to calculate the number of symmetric keys needed is

$$N(N - 1)/2 = \text{number of keys}$$

The security of the symmetric encryption method is completely dependent on how well users protect their shared keys. This should raise red flags for you if you have ever had to depend on a whole staff of people to keep a secret. If a key is compromised, then all messages encrypted with that key can be decrypted and read by an intruder. This is complicated further by how symmetric keys are actually shared and updated when necessary. If Dan wants to communicate with Norm for the first time, Dan has to figure out how to get the right key to Norm securely. It is not safe to just send it in an e-mail

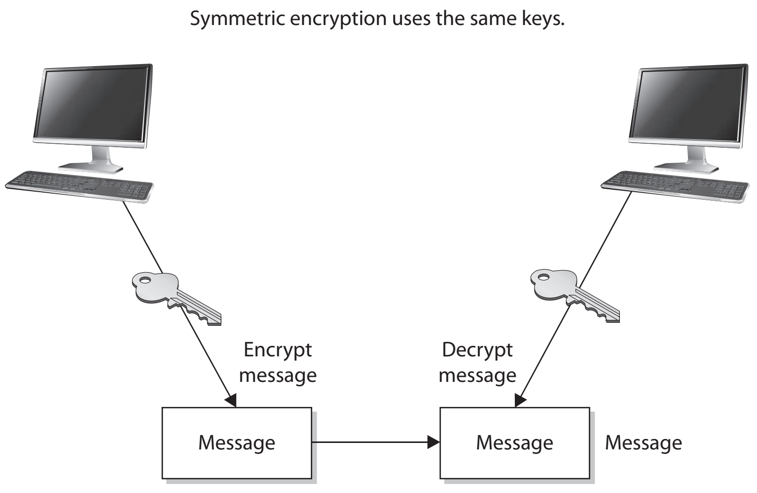


Figure 8-6 When using symmetric algorithms, the sender and receiver use the same key for encryption and decryption functions.

Symmetric Key Cryptosystems Summary

The following outlines the strengths and weaknesses of symmetric key algorithms.

Strengths:

- Much faster (less computationally intensive) than asymmetric systems.
- Hard to break if using a large key size.

Weaknesses:

- Requires a secure mechanism to deliver keys properly.
- Each pair of users needs a unique key, so as the number of individuals increases, so does the number of keys, possibly making key management overwhelming.
- Provides confidentiality but not authenticity or nonrepudiation.

Examples:

- Advanced Encryption Standard (AES)
- ChaCha20

message, because the key is not protected and can be easily intercepted and used by attackers. Thus, Dan must get the key to Norm through an *out-of-band method*. Dan can save the key on a thumb drive and walk over to Norm's desk, or have a secure courier deliver it to Norm. This is a huge hassle, and each method is very clumsy and insecure.

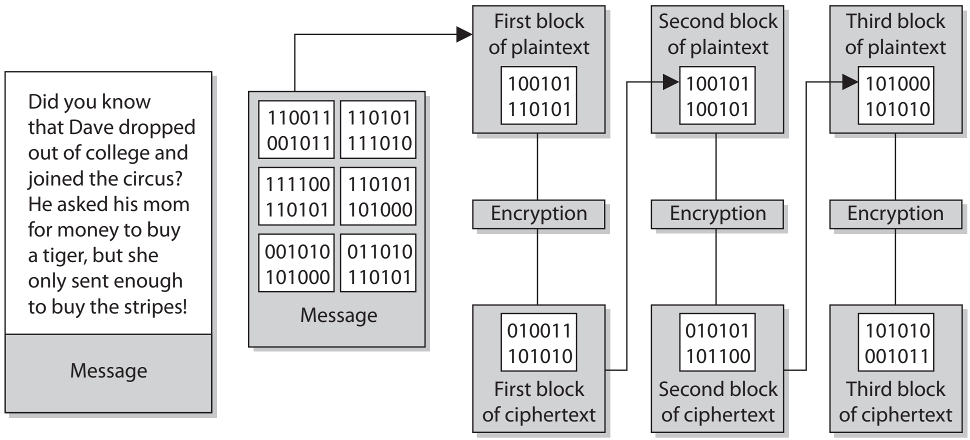
Because both users employ the same key to encrypt and decrypt messages, symmetric cryptosystems can provide confidentiality, but they cannot provide authentication or nonrepudiation. There is no way to prove through cryptography who actually sent a message if two people are using the same key.

If symmetric cryptosystems have so many problems and flaws, why use them at all? Because they are very fast and can be hard to break. Compared with asymmetric systems, symmetric algorithms scream in speed. They can encrypt and decrypt relatively quickly large amounts of data that would take an unacceptable amount of time to encrypt and decrypt with an asymmetric algorithm. It is also difficult to uncover data encrypted with a symmetric algorithm if a large key size is used. For many of our applications that require encryption, symmetric key cryptography is the only option.

The two main types of symmetric algorithms are block ciphers, which work on blocks of bits, and stream ciphers, which work on one bit at a time.

Block Ciphers

When a *block cipher* is used for encryption and decryption purposes, the message is divided into blocks of bits. These blocks are then put through mathematical functions, one block at a time. Suppose you need to encrypt a message you are sending to your mother and you are using a block cipher that uses 64 bits. Your message of 640 bits is chopped up into 10 individual blocks of 64 bits. Each block is put through a succession of mathematical formulas, and what you end up with is 10 blocks of encrypted text.


Did you know that Dave dropped out of college and joined the circus? He asked his mom for money to buy a tiger, but she only sent enough to buy the stripes!
Message
110011 001011
110101 111010
111100 110101
110101 101000
001010 101000
011010 110101
Message
First block of plaintext
100101 110101
Second block of plaintext
100101 100101
Third block of plaintext
101000 101010
Encryption
Encryption
Encryption
010011 101010
First block of ciphertext
010101 101100
Second block of ciphertext
101010 001011
Third block of ciphertext

You send this encrypted message to your mother. She has to have the same block cipher and key, and those 10 ciphertext blocks go back through the algorithm in the reverse sequence and end up in your plaintext message.

A strong cipher contains the right level of two main attributes: confusion and diffusion. *Confusion* is commonly carried out through substitution, while *diffusion* is carried out by using transposition. For a cipher to be considered strong, it must contain both of these attributes to ensure that reverse-engineering is basically impossible. The randomness of the key values and the complexity of the mathematical functions dictate the level of confusion and diffusion involved.

In algorithms, diffusion takes place as individual bits of a block are scrambled, or *diffused*, throughout that block. Confusion is provided by carrying out complex substitution functions so the eavesdropper cannot figure out how to substitute the right values and come up with the original plaintext. Suppose you have 500 wooden blocks with individual letters written on them. You line them all up to spell out a paragraph (plaintext). Then you substitute 300 of them with another set of 300 blocks (confusion through substitution). Then you scramble all of these blocks (diffusion through transposition) and leave them in a pile. For someone else to figure out your original message, they would have to substitute the correct blocks and then put them back in the right order. Good luck.

Confusion pertains to making the relationship between the key and resulting ciphertext as complex as possible so the key cannot be uncovered from the ciphertext. Each ciphertext value should depend upon several parts of the key, but this mapping between the key values and the ciphertext values should seem completely random to the observer.

Diffusion, on the other hand, means that a single plaintext bit has influence over several of the ciphertext bits. Changing a plaintext value should change many ciphertext values, not just one. In fact, in a strong block cipher, if one plaintext bit is changed, it will change every ciphertext bit with the probability of 50 percent. This means that if one plaintext bit changes, then about half of the ciphertext bits will change.

A very similar concept of diffusion is the *avalanche effect*. If an algorithm follows strict avalanche effect criteria, this means that if the input to an algorithm is slightly modified, then the output of the algorithm is changed significantly. So a small change to the key or the plaintext should cause drastic changes to the resulting ciphertext. The ideas of diffusion and avalanche effect are basically the same—they were just derived from different people. Horst Feistel came up with the avalanche term, while Claude Shannon came up with the diffusion term. If an algorithm does not exhibit the necessary degree of the avalanche effect, then the algorithm is using poor randomization. This can make it easier for an attacker to break the algorithm.

Block ciphers use diffusion and confusion in their methods. Figure 8-7 shows a conceptual example of a simplistic block cipher. It has four block inputs, and each block is made up of 4 bits. The block algorithm has two layers of 4-bit substitution boxes called

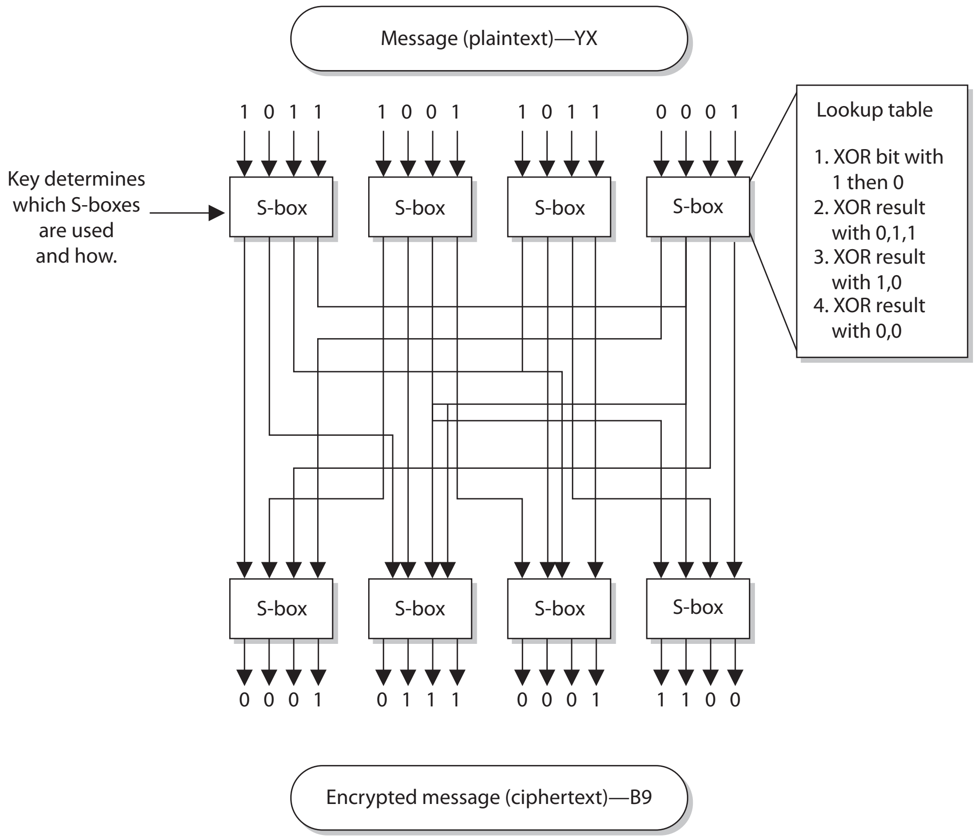


Figure 8-7 A message is divided into blocks of bits, and substitution and transposition functions are performed on those blocks.

S-boxes. Each S-box contains a lookup table used by the algorithm as instructions on how the bits should be encrypted.

Figure 8-7 shows that the key dictates what S-boxes are to be used when scrambling the original message from readable plaintext to encrypted nonreadable ciphertext. Each S-box contains the different substitution methods that can be performed on each block. This example is simplistic—most block ciphers work with blocks of 32, 64, or 128 bits in size, and many more S-boxes are usually involved.

Stream Ciphers

As stated earlier, a block cipher performs mathematical functions on blocks of bits. A stream cipher, on the other hand, does not divide a message into blocks. Instead, a *stream cipher* treats the message as a stream of bits and performs mathematical functions on each bit individually.

When using a stream cipher, a plaintext bit will be transformed into a different ciphertext bit each time it is encrypted. Stream ciphers use *keystream generators*, which produce a stream of bits that is XORed with the plaintext bits to produce ciphertext, as shown in Figure 8-8.

NOTE This process is very similar to the one-time pad explained earlier. The individual bits in the one-time pad are used to encrypt the individual bits of the message through the XOR function, and in a stream algorithm the individual bits created by the keystream generator are used to encrypt the bits of the message through XOR also.

In block ciphers, it is the key that determines what functions are applied to the plaintext and in what order. The key provides the randomness of the encryption process. As stated earlier, most encryption algorithms are public, so people know how they work. The secret to the secret sauce is the key. In stream ciphers, the key also provides randomness, so that the stream of bits that is XORed to the plaintext is as random as possible. This concept

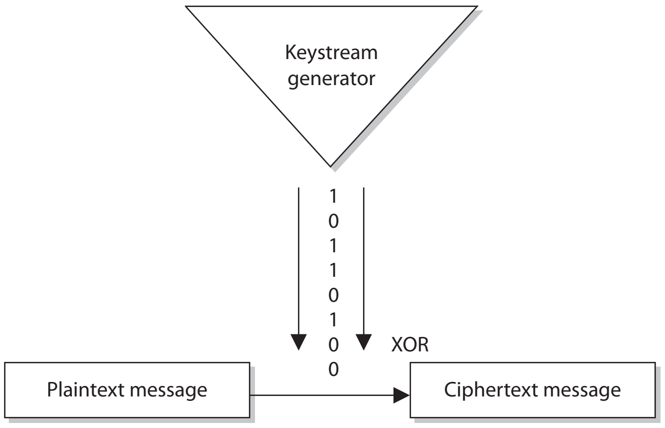


Figure 8-8 With stream ciphers, the bits generated by the keystream generator are XORed with the bits of the plaintext message.

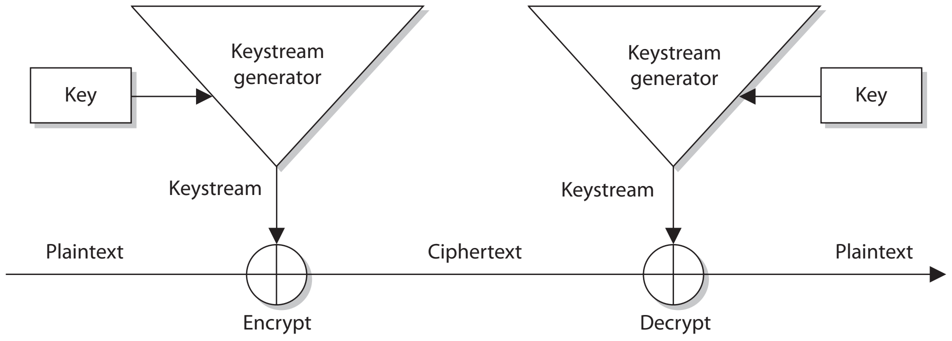


Figure 8-9 The sender and receiver must have the same key to generate the same keystream.

is shown in Figure 8-9. As you can see in this graphic, both the sending and receiving ends must have the same key to generate the same keystream for proper encryption and decryption purposes.

Initialization Vectors

Initialization vectors (IVs) are random values that are used with algorithms to ensure patterns are not created during the encryption process. They are used with keys and do not need to be encrypted when being sent to the destination. If IVs are not used, then two identical plaintext values that are encrypted with the same key will create the same ciphertext. Providing attackers with these types of patterns can make their job easier in breaking the encryption method and uncovering the key. For example, if we have the plaintext value of "See Spot run" two times within our message, we need to make sure that even though there is a pattern in the plaintext message, a pattern in the resulting ciphertext will not be created. So the IV and key are both used by the algorithm to provide more randomness to the encryption process.

A strong and effective stream cipher contains the following characteristics:

- **Easy to implement in hardware** Complexity in the hardware design makes it more difficult to verify the correctness of the implementation and can slow it down.
- **Long periods of no repeating patterns within keystream values** Bits generated by the keystream are not truly random in most cases, which will eventually lead to the emergence of patterns; we want these patterns to be rare.
- **A keystream not linearly related to the key** If someone figures out the keystream values, that does not mean she now knows the key value.
- **Statistically unbiased keystream (as many zeroes as ones)** There should be no dominance in the number of zeroes or ones in the keystream.

Stream ciphers require a lot of randomness and encrypt individual bits at a time. This requires more processing power than block ciphers require, which is why stream ciphers are better suited to be implemented at the hardware level. Because block ciphers do not require as much processing power, they can be easily implemented at the software level.

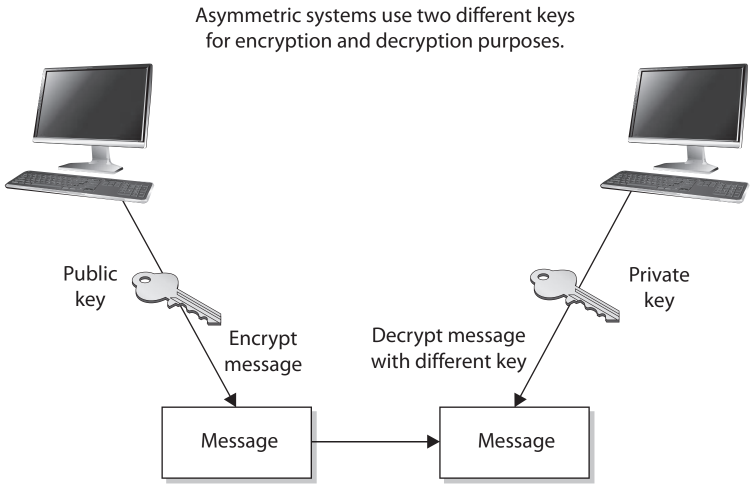
Asymmetric Key Cryptography

In symmetric key cryptography, a single secret key is used between entities, whereas in public key systems, each entity has different, *asymmetric keys*. The two different asymmetric keys are mathematically related. If a message is encrypted by one key, the other key is required in order to decrypt the message. One key is called public and the other one private. The *public key* can be known to everyone, and the *private key* must be known and used only by the owner. Many times, public keys are listed in directories and databases of e-mail addresses so they are available to anyone who wants to use these keys to encrypt or decrypt data when communicating with a particular person. Figure 8-10 illustrates the use of the different keys.

The public and private keys of an asymmetric cryptosystem are mathematically related, but if someone gets another person's public key, she should not be able to figure out the corresponding private key. This means that if an eavesdropper gets a copy of Bob's public key, she can't employ some mathematical magic and find out Bob's private key. But if someone gets Bob's private key, then there is big trouble—no one other than the owner should have access to a private key.

If Bob encrypts data with his private key, the receiver must have a copy of Bob's public key to decrypt it. The receiver can decrypt Bob's message and decide to reply to Bob in an encrypted form. All the receiver needs to do is encrypt her reply with Bob's public key, and then Bob can decrypt the message with his private key. It is not possible to encrypt and decrypt using the same key when using an asymmetric key encryption technology because, although mathematically related, the two keys are not the same key, as they are in symmetric cryptography. Bob can encrypt data with his private key, and the receiver can then decrypt it with Bob's public key. By decrypting the message with Bob's

Figure 8-10
An asymmetric cryptosystem


Asymmetric systems use two different keys for encryption and decryption purposes.
Public key
Encrypt message
Decrypt message with different key
Private key
Message
Message

public key, the receiver can be sure the message really came from Bob. A message can be decrypted with a public key only if the message was encrypted with the corresponding private key. This provides authentication, because Bob is the only one who is supposed to have his private key. However, it does not truly provide confidentiality because anyone with the public key (which is, after all, public) can decrypt it. If the receiver wants to make sure Bob is the only one who can read her reply, she will encrypt the response with his public key. Only Bob will be able to decrypt the message because he is the only one who has the necessary private key.

The receiver can also choose to encrypt data with her private key instead of using Bob's public key. Why would she do that? Authentication—she wants Bob to know that the message came from her and no one else. If she encrypted the data with Bob's public key, it does not provide authenticity because anyone can get Bob's public key. If she uses her private key to encrypt the data, then Bob can be sure the message came from her and no one else. Symmetric keys do not provide authenticity, because the same key is used on both ends. Using one of the secret keys does not ensure the message originated from a specific individual.

If confidentiality is the most important security service to a sender, she would encrypt the file with the receiver's public key. This is called a *secure message format* because it can only be decrypted by the person who has the corresponding private key.

If authentication is the most important security service to the sender, then she would encrypt the data with her private key. This provides assurance to the receiver that the only person who could have encrypted the data is the individual who has possession of that private key. If the sender encrypted the data with the receiver's public key, authentication is not provided because this public key is available to anyone.

Encrypting data with the sender's private key is called an *open message format* because anyone with a copy of the corresponding public key can decrypt the message. Confidentiality is not ensured.

Each key type can be used to encrypt and decrypt, so do not get confused and think the public key is only for encryption and the private key is only for decryption. They both have the capability to encrypt and decrypt data. However, if data is encrypted with a private key, it cannot be decrypted with a private key. If data is encrypted with a private key, it must be decrypted with the corresponding public key.

An asymmetric algorithm works much more slowly than a symmetric algorithm, because symmetric algorithms carry out relatively simplistic mathematical functions on the bits during the encryption and decryption processes. They substitute and scramble (transposition) bits, which is not overly difficult or processor intensive. The reason it is hard to break this type of encryption is that the symmetric algorithms carry out this type of functionality over and over again. So a set of bits will go through a long series of being substituted and scrambled.

Asymmetric algorithms are slower than symmetric algorithms because they use much more complex mathematics to carry out their functions, which requires more processing time. Although they are slower, asymmetric algorithms can provide authentication and nonrepudiation, depending on the type of algorithm being used. Asymmetric systems also provide for easier and more manageable key distribution than symmetric systems and do not have the scalability issues of symmetric systems. The reason for these differences

Asymmetric Key Cryptosystems Summary

The following outlines the strengths and weaknesses of asymmetric key algorithms.

Strengths:

- Better key distribution than symmetric systems.
- Better scalability than symmetric systems.
- Can provide authentication and nonrepudiation.

Weaknesses:

- Works much more slowly than symmetric systems.
- Mathematically intensive tasks.

Examples:

- Rivest-Shamir-Adleman (RSA)
- Elliptic curve cryptography (ECC)
- Digital Signature Algorithm (DSA)

is that, with asymmetric systems, you can send out your public key to all of the people you need to communicate with, instead of keeping track of a unique key for each one of them. The "Hybrid Encryption Methods" section later in this chapter shows how these two systems can be used together to get the best of both worlds.

TIP Public key cryptography is asymmetric cryptography. The terms can be used interchangeably.

Table 8-1 summarizes the differences between symmetric and asymmetric algorithms.

Diffie-Hellman Algorithm

The first group to address the shortfalls of symmetric key cryptography decided to attack the issue of secure distribution of the symmetric key. Whitfield Diffie and Martin Hellman worked on this problem and ended up developing the first asymmetric key agreement algorithm, called, naturally, Diffie-Hellman.

To understand how *Diffie-Hellman* works, consider an example. Let's say that Tanya and Erika would like to communicate over an encrypted channel by using Diffie-Hellman. They would both generate a private and public key pair and exchange public keys. Tanya's software would take her private key (which is just a numeric value) and Erika's public key (another numeric value) and put them through the Diffie-Hellman algorithm. Erika's software would take her private key and Tanya's public key and insert them into the Diffie-Hellman algorithm on her computer. Through this process, Tanya and Erika derive the same shared value, which is used to create instances of symmetric keys.

Attribute	Symmetric	Asymmetric
Keys	One key is shared between two or more entities.	One entity has a public key, and the other entity has the corresponding private key.
Key exchange	Out-of-band through secure mechanisms.	A public key is made available to everyone, and a private key is kept secret by the owner.
Speed	The algorithm is less complex and faster.	The algorithm is more complex and slower.
Use	Bulk encryption, which means encrypting files and communication paths.	Key distribution and digital signatures.
Security service provided	Confidentiality.	Confidentiality, authentication, and nonrepudiation.

Table 8-1 Differences Between Symmetric and Asymmetric Systems

So, Tanya and Erika exchanged information that did not need to be protected (their public keys) over an untrusted network, and in turn generated the exact same symmetric key on each system. They both can now use these symmetric keys to encrypt, transmit, and decrypt information as they communicate with each other.

NOTE The preceding example describes key *agreement*, which is different from key *exchange*, the functionality used by the other asymmetric algorithms that will be discussed in this chapter. With key exchange functionality, the sender encrypts the symmetric key with the receiver's public key before transmission.

The Diffie-Hellman algorithm enables two systems to generate a symmetric key securely without requiring a previous relationship or prior arrangements. The algorithm allows for key distribution, but does not provide encryption or digital signature functionality. The algorithm is based on the difficulty of calculating discrete logarithms in a finite field.

The original Diffie-Hellman algorithm is vulnerable to a man-in-the-middle attack, because no authentication occurs before public keys are exchanged. In our example, when Tanya sends her public key to Erika, how does Erika really know it is Tanya's public key? What if Lance spoofed his identity, told Erika he was Tanya, and sent over his key? Erika would accept this key, thinking it came from Tanya. Let's walk through the steps of how this type of attack would take place, as illustrated in Figure 8-11:

1. Tanya sends her public key to Erika, but Lance grabs the key during transmission so it never makes it to Erika.
2. Lance spoofs Tanya's identity and sends over his public key to Erika. Erika now thinks she has Tanya's public key.

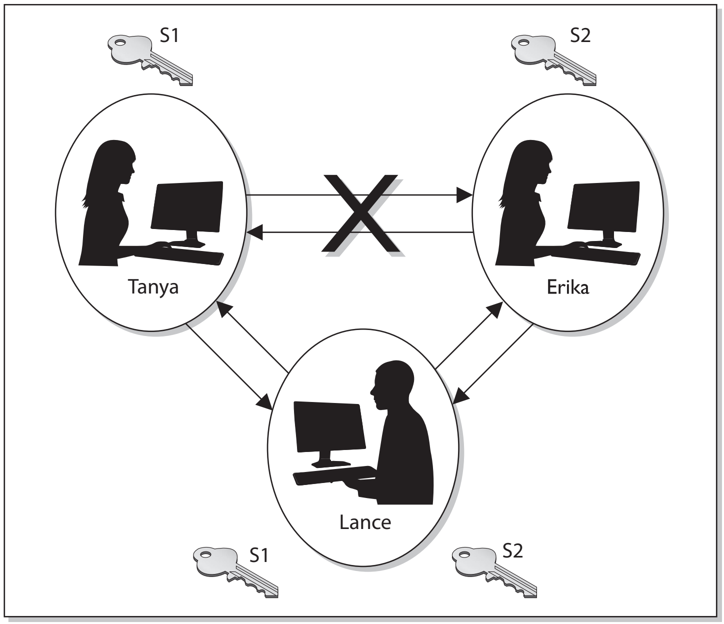


Figure 8-11 A man-in-the-middle attack against a Diffie-Hellman key agreement

3. Erika sends her public key to Tanya, but Lance grabs the key during transmission so it never makes it to Tanya.
4. Lance spoofs Erika's identity and sends over his public key to Tanya. Tanya now thinks she has Erika's public key.
5. Tanya combines her private key and Lance's public key and creates symmetric key S1.
6. Lance combines his private key and Tanya's public key and creates symmetric key S1.
7. Erika combines her private key and Lance's public key and creates symmetric key S2.
8. Lance combines his private key and Erika's public key and creates symmetric key S2.
9. Now Tanya and Lance share a symmetric key (S1) and Erika and Lance share a different symmetric key (S2). Tanya and Erika think they are sharing a key between themselves and do not realize Lance is involved.
10. Tanya writes a message to Erika, uses her symmetric key (S1) to encrypt the message, and sends it.
11. Lance grabs the message and decrypts it with symmetric key S1, reads or modifies the message and re-encrypts it with symmetric key S2, and then sends it to Erika.
12. Erika takes symmetric key S2 and uses it to decrypt and read the message.

The countermeasure to this type of attack is to have authentication take place before accepting someone's public key. The basic idea is that we use some sort of certificate to attest the identity of the party on the other side before trusting the data we receive from it. One of the most common ways to do this authentication is through the use of the RSA cryptosystem, which we describe next.

RSA

RSA, named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman, is a public key algorithm that is the most popular when it comes to asymmetric algorithms. RSA is a worldwide de facto standard and can be used for digital signatures, key exchange, and encryption. It was developed in 1978 at MIT and provides authentication as well as key encryption.

The security of this algorithm comes from the difficulty of factoring large numbers into their original prime numbers. The public and private keys are functions of a pair of large prime numbers, and the necessary activity required to decrypt a message from ciphertext to plaintext using a private key is comparable to factoring a product into two prime numbers.

NOTE A prime number is a positive whole number whose only factors (i.e., integer divisors) are 1 and the number itself.

One advantage of using RSA is that it can be used for encryption and digital signatures. Using its one-way function, RSA provides encryption and signature verification, and the inverse direction performs decryption and signature generation.

RSA has been implemented in applications; in operating systems; and at the hardware level in network interface cards, secure telephones, and smart cards. RSA can be used as a *key exchange protocol*, meaning it is used to encrypt the symmetric key to get it securely to its destination. RSA has been most commonly used with the symmetric algorithm AES. So, when RSA is used as a key exchange protocol, a cryptosystem generates a symmetric key to be used with the AES algorithm. Then the system encrypts the symmetric key with the receiver's public key and sends it to the receiver. The symmetric key is protected because only the individual with the corresponding private key can decrypt and extract the symmetric key.

Diving into Numbers Cryptography is really all about using mathematics to scramble bits into an undecipherable form and then using the same mathematics in reverse to put the bits back into a form that can be understood by computers and people. RSA's mathematics are based on the difficulty of factoring a large integer into its two prime factors. Put on your nerdy hat with the propeller and let's look at how this algorithm works.

The algorithm creates a public key and a private key from a function of large prime numbers. When data is encrypted with a public key, only the corresponding private key can decrypt the data. This act of decryption is basically the same as factoring the product of two prime numbers. So, let's say Ken has a secret (encrypted message), and for you to be able to uncover the secret, you have to take a specific large number and factor it and come up with the two numbers Ken has written down on a piece of paper. This may sound simplistic, but the number you must properly factor can be 2^{2048} in size. Not as easy as you may think.

The following sequence describes how the RSA algorithm comes up with the keys in the first place:

1. Choose two random large prime numbers, p and q.
2. Generate the product of these numbers: $n = pq$. n is used as the modulus.
3. Choose a random integer e (the public key) that is greater than 1 but less than $(p – 1)(q – 1)$. Make sure that e and $(p – 1)(q – 1)$ are relatively prime.
4. Compute the corresponding private key, d, such that $de – 1$ is a multiple of $(p – 1)(q – 1)$.
5. The public key = (n, e).
6. The private key = (n, d).
7. The original prime numbers p and q are discarded securely.

We now have our public and private keys, but how do they work together?

If someone needs to encrypt message m with your public key (e, n), the following formula results in ciphertext c:

$$c = m^e \bmod n$$

Then you need to decrypt the message with your private key (d), so the following formula is carried out:

$$m = c^d \bmod n$$

In essence, you encrypt a plaintext message by multiplying it by itself e times (taking the modulus, of course), and you decrypt it by multiplying the ciphertext by itself d times (again, taking the modulus). As long as e and d are large enough values, an attacker will have to spend an awfully long time trying to figure out through trial and error the value of d. (Recall that we publish the value of e for the whole world to know.)

You may be thinking, “Well, I don’t understand these formulas, but they look simple enough. Why couldn’t someone break these small formulas and uncover the encryption key?” Maybe someone will one day. As the human race advances in its understanding of mathematics and as processing power increases and cryptanalysis evolves, the RSA algorithm may be broken one day. If we were to figure out how to quickly and more easily factor large numbers into their original prime values, all of these cards would fall down, and this algorithm would no longer provide the security it does today. But we have not hit that bump in the road yet, so we are all happily using RSA in our computing activities.

One-Way Functions A *one-way function* is a mathematical function that is easier to compute in one direction than in the opposite direction. An analogy of this is when you drop a glass on the floor. Although dropping a glass on the floor is easy, putting all the pieces back together again to reconstruct the original glass is next to impossible. This concept is similar to how a one-way function is used in cryptography, which is what the RSA algorithm, and all other asymmetric algorithms, are based upon.

The easy direction of computation in the one-way function that is used in the RSA algorithm is the process of multiplying two large prime numbers. If I asked you to multiply two prime numbers, say 79 and 73, it would take you just a few seconds to punch that into a calculator and come up with the product (5,767). Easy. Now, suppose I asked you to find out which two numbers, when multiplied together, produce the value 5,767. This is called factoring and, when the factors involved are large prime numbers, it turns out to be a *really* hard problem. This difficulty in factoring the product of large prime numbers is what provides security for RSA key pairs.

As explained earlier in this chapter, *work factor* is the amount of time and resources it would take for someone to break an encryption method. In asymmetric algorithms, the work factor relates to the difference in time and effort that carrying out a one-way function in the easy direction takes compared to carrying out a one-way function in the hard direction. In most cases, the larger the key size, the longer it would take for the adversary to carry out the one-way function in the hard direction (decrypt a message).

The crux of this section is that all asymmetric algorithms provide security by using mathematical equations that are easy to perform in one direction and next to impossible to perform in the other direction. The "hard" direction is based on a "hard" mathematical problem. RSA's hard mathematical problem requires factoring large numbers into their original prime numbers.

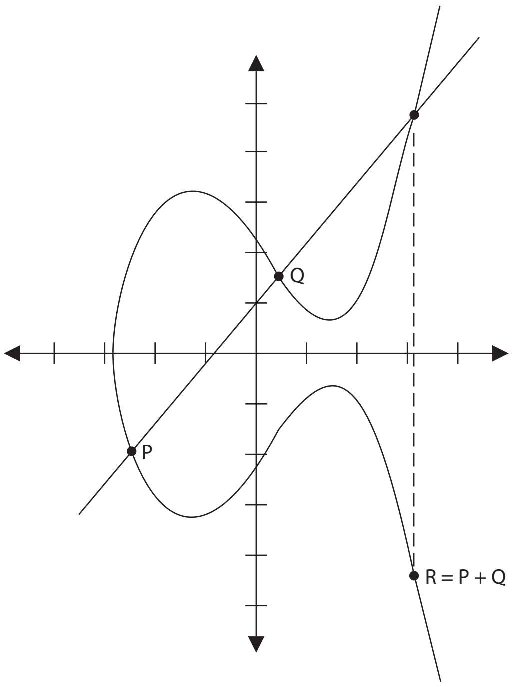
Elliptic Curve Cryptography

The one-way function in RSA has survived cryptanalysis for over four decades but eventually will be cracked simply because we keep building computers that are faster. Sooner or later, computers will be able to factor the products of ever-larger prime numbers in reasonable times, at which point we would need to either ditch RSA or figure out how to use larger keys. Anticipating this eventuality, cryptographers found an even better trapdoor in elliptic curves. An *elliptic curve*, such as the one shown in Figure 8-12, is the set of points that satisfies a specific mathematical equation such as this one:

$$y^2 = x^3 + ax + b$$

Elliptic curves have two properties that are useful for cryptography. The first is that they are symmetrical about the X axis. This means that the top and bottom parts of the curve are mirror images of each other. The second useful property is that a straight line will intersect them in no more than three points. With these properties in mind, you can define a "dot" function that, given two points on the curve, gives you a third point on the flip side of it. Figure 8-12 shows how P dot Q = R. You simply follow the line through P and Q to find its third point of intersection on the curve (which could be between the two), and then drop down to that point R on the mirror image (in this case) below the X axis. You can keep going from there, so R dot P gives you another point that is

Figure 8-12
Elliptic curve



somewhere to the left and up from Q on the curve. If you keep "dotting" the original point P with the result of the previous "dot" operation *n* times (for some reasonably large value of *n*), you end up with a point that is really hard for anyone to guess or brute-force if they don't know the value of *n*. If you do know that value, then computing the final point is pretty easy. That is what makes this a great one-way function.

An *elliptic curve cryptosystem (ECC)* is a public key cryptosystem that can be described by a prime number (the equivalent of the modulus value in RSA), a curve equation, and a public point on the curve. The private key is some number *d*, and the corresponding public key *e* is the public point on the elliptic curve "dotted" with itself *d* times. Computing the private key from the public key in this kind of cryptosystem (i.e., reversing the one-way function) requires calculating the elliptic curve discrete logarithm function, which turns out to be really, really hard.

ECC provides much of the same functionality RSA provides: digital signatures, secure key distribution, and encryption. One differing factor is ECC's efficiency. ECC is more efficient than RSA and any other asymmetric algorithm. To illustrate this, an ECC key of 256 bits offers the equivalent protection of an RSA key of 3,072 bits. This is particularly useful because some devices have limited processing capacity, storage, power supply, and bandwidth, such as wireless devices and mobile telephones. With these types of devices, efficiency of resource use is very important. ECC provides encryption functionality, requiring a smaller percentage of the resources compared to RSA and other algorithms, so it is used in these types of devices.

Quantum Cryptography

Both RSA and ECC rely on the difficulty of reversing one-way functions. But what if we were able to come up with a cryptosystem in which it was impossible (not just difficult) to do this? This is the promise of quantum cryptography, which, despite all the hype, is still very much in its infancy. *Quantum cryptography* is the field of scientific study that applies quantum mechanics to perform cryptographic functions. The most promising application of this field, and the one we may be able to use soonest, provides a solution to the key distribution problem associated with symmetric key cryptosystems.

Quantum key distribution (QKD) is a system that generates and securely distributes encryption keys of any length between two parties. Though we could, in principle, use anything that obeys the principles of quantum mechanics, photons (the tiny particles that make up light) are the most convenient particles to use for QKD. It turns out photons are polarized or spin in ways that can be described as vertical, horizontal, diagonal left (–45°), and diagonal right (45°). If we put a polarized filter in front of a detector, any photon that makes it to that detector will have the polarization of its filter. Two types of filters are commonly used in QKD. The first is rectilinear and allows vertically and horizontally polarized photons through. The other is a (you guessed it) diagonal filter, which allows both diagonally left and diagonally right polarized photons through. It is important to note that the only way to measure the polarization on a photon is to essentially destroy it: either it is blocked by the filter if the polarizations are different or it is absorbed by the sensor if it makes it through.

Let's suppose that Alice wants to securely send an encryption key to Bob using QKD. They would use the following process.

1. They agree beforehand that photons that have either vertical or diagonal-right polarization represent the number zero and those with horizontal or diagonal-left polarization represent the number one.
2. The polarization of each photon is then generated randomly but is known to Alice.
3. Since Bob doesn't know what the correct spins are, he'll pass them through filters, randomly detect the polarization for each photon, and record his results. Because he's just guessing the polarizations, on average, he'll get half of them wrong, as we can see in Figure 8-13. He will, however, know which filter he applied to each photon, whether he got it right or wrong.
4. Once Alice is done sending bits, Bob will send her a message over an insecure channel (they don't need encryption for this), telling her the sequence of polarizations he recorded.
5. Alice will compare Bob's sequence to the correct sequence and tell him which polarizations he got right and which ones he got wrong.
6. They both discard Bob's wrong guesses and keep the remaining sequence of bits. They now have a shared secret key through this process, which is known as *key distillation*.

But what if there's a third, malicious, party eavesdropping on the exchange? Suppose this is Eve and she wants to sniff the secret key so she can intercept whatever messages

Figure 8-13 Key distillation between Alice and Bob

Alice's bit	0	1	1	0	1	0	0	1
Alice's basis	+	+	×	+	×	×	×	+
Alice's polarization	↑	→	↖	↑	↖	↗	↗	→
Bob's filter	+	×	×	×	+	×	+	+
Bob's measurement	↑	↗	↖	↗	→	↗	→	→
Shared secret key	0		1			0		1

Alice and Bob encrypt with it. Since the quantum state of photons is destroyed when they are filtered or measured, she would have to follow the same process as Bob intends to and then generate a new photon stream to forward to Bob. The catch is that Eve (just like Bob) will get 50 percent of the measurements wrong, but (unlike Bob) now has to guess what random basis was used and send these guesses to Bob. When Alice and Bob compare polarizations, they'll note a much higher error rate than normal and be able to infer that someone was eavesdropping.

If you're still awake and paying attention, you may be wondering, "Why use the polarization filters in the first place? Why not just capture the photon and see how it's spinning?" The answer gets complicated in a hurry, but the short version is that polarization is a random quantum state until you pass the photon through the filter and force the photon to "decide" between the two polarizations. Eve cannot just re-create the photon's quantum state like she would do with conventional data. Keep in mind that quantum mechanics are pretty weird but lead to unconditional security of the shared key.

Now that we have a basic idea of how QKD works, let's think back to our discussion of the only perfect and unbreakable cryptosystem: the one-time pad. You may recall that it has five major requirements that largely make it impractical. We list these here and show how QKD addresses each of them rather nicely:

- **Made up of truly random values** Quantum mechanics deals with attributes of matter and energy that are truly random, unlike the pseudo-random numbers we can generate algorithmically on a traditional computer.
- **Used only one time** Since QKD solves the key distribution problem, it allows us to transmit as many unique keys as we want, reducing the temptation (or need) to reuse keys.
- **Securely distributed to its destination** If someone attempts to eavesdrop on the key exchange, they will have to do so actively in a way that, as we've seen, is pretty much guaranteed to produce evidence of their tampering.
- **Secured at sender's and receiver's sites** OK, this one is not really addressed by QKD directly, but anyone going through all this effort would presumably not mess this one up, right?
- **At least as long as the message** Since QKD can be used for arbitrarily long key streams, we can easily generate keys that are at least as long as the longest message we'd like to send.

Now, before you get all excited and try to buy a QKD system for your organization, keep in mind that this technology is not quite ready for prime time. To be clear, commercial QKD devices are available as a “plug and play” option. Some banks in Geneva, Switzerland, use QKD to secure bank-to-bank traffic, and the Canton of Geneva uses it to secure online voting. The biggest challenge to widespread adoption of QKD at this point is the limitation on the distance at which photons can be reliably transmitted. As we write these lines, the maximum range for QKD is just over 500 km over fiber-optic wires. While space-to-ground QKD has been demonstrated using satellites and ground stations, drastically increasing the reach of such systems, it remains extremely difficult due to atmospheric interference. Once this problem is solved, we should be able to leverage a global, satellite-based QKD network.

Hybrid Encryption Methods

Up to this point, we have figured out that symmetric algorithms are fast but have some drawbacks (lack of scalability, difficult key management, and provide only confidentiality). Asymmetric algorithms do not have these drawbacks but are very slow. We just can’t seem to win. So we turn to a hybrid system that uses symmetric and asymmetric encryption methods together.

Asymmetric and Symmetric Algorithms Used Together

Asymmetric and symmetric cryptosystems are used together very frequently. In this hybrid approach, the two technologies are used in a complementary manner, with each performing a different function. A symmetric algorithm creates keys used for encrypting bulk data, and an asymmetric algorithm creates keys used for automated key distribution. Each algorithm has its pros and cons, so using them together can be the best of both worlds.

When a symmetric key is used for bulk data encryption, this key is used to encrypt the message you want to send. When your friend gets the message you encrypted, you want him to be able to decrypt it, so you need to send him the necessary symmetric key to use to decrypt the message. You do not want this key to travel unprotected, because if the message were intercepted and the key were not protected, an eavesdropper could intercept the message that contains the necessary key to decrypt your message and read your information. If the symmetric key needed to decrypt your message is not protected, there is no use in encrypting the message in the first place. So you should use an asymmetric algorithm to encrypt the symmetric key, as depicted in Figure 8-14. Why use the symmetric key on the message and the asymmetric key on the symmetric key? As stated earlier, the asymmetric algorithm takes longer because the math is more complex. Because your message is most likely going to be longer than the length of the key, you use the faster algorithm (symmetric) on the message and the slower algorithm (asymmetric) on the key.

How does this actually work? Let’s say Bill is sending Paul a message that Bill wants only Paul to be able to read. Bill encrypts his message with a secret key, so now Bill has ciphertext and a symmetric key. The key needs to be protected, so Bill encrypts the symmetric key with an asymmetric key. Remember that asymmetric algorithms use private and public keys, so Bill will encrypt the symmetric key with Paul’s public key. Now Bill has ciphertext from the message and ciphertext from the symmetric key. Why did Bill encrypt the symmetric key with Paul’s public key instead of his own private key? Because if Bill

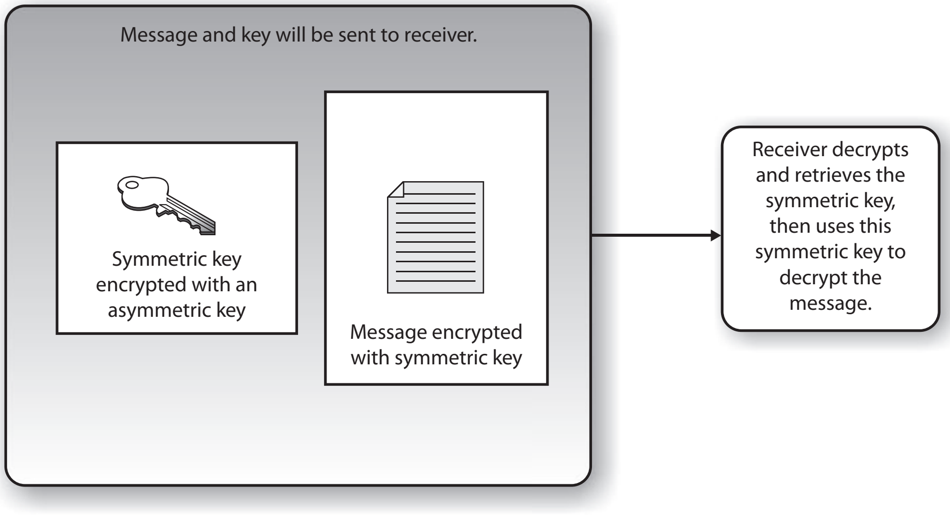
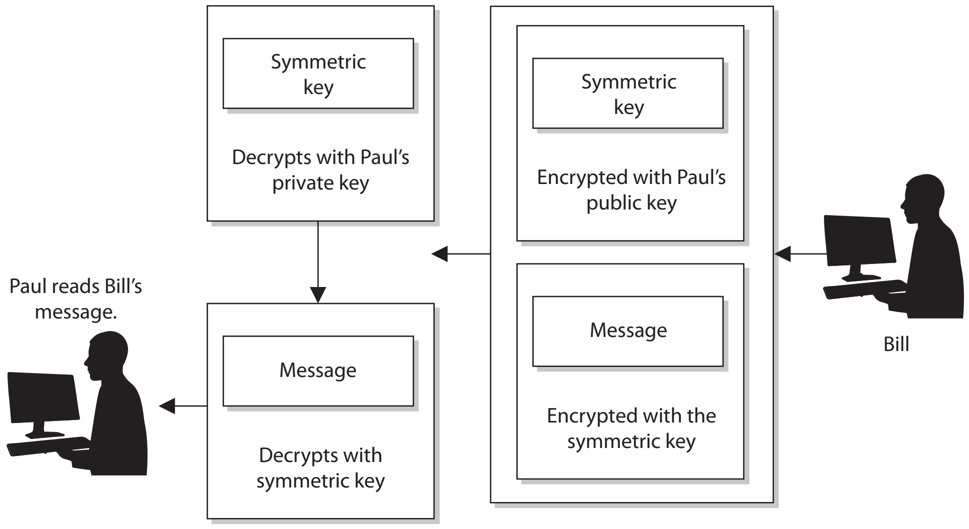

Message and key will be sent to receiver.
Symmetric key encrypted with an asymmetric key
Message encrypted with symmetric key
Receiver decrypts and retrieves the symmetric key, then uses this symmetric key to decrypt the message.

Figure 8-14 In a hybrid system, the asymmetric key is used to encrypt the symmetric key, and the symmetric key is used to encrypt the message

encrypted it with his own private key, then anyone with Bill's public key could decrypt it and retrieve the symmetric key. However, Bill does not want anyone who has his public key to read his message to Paul. Bill only wants Paul to be able to read it. So Bill encrypts the symmetric key with Paul's public key. If Paul has done a good job protecting his private key, he will be the only one who can read Bill's message.

Paul receives Bill's message, and Paul uses his private key to decrypt the symmetric key. Paul then uses the symmetric key to decrypt the message. Paul then reads Bill's very important and confidential message that asks Paul how his day is.


Symmetric key
Decrypts with Paul's private key
Symmetric key
Encrypted with Paul's public key
Message
Encrypted with the symmetric key
Bill
Paul reads Bill's message.
Message
Decrypts with symmetric key

Now, when we say that Bill is using this key to encrypt and that Paul is using that key to decrypt, those two individuals do not necessarily need to find the key on their hard drive and know how to properly apply it. We have software to do this for us—thank goodness.

If this is your first time with these issues and you are struggling, don't worry. Just remember the following points:

- An asymmetric algorithm performs encryption and decryption by using public and private keys that are related to each other mathematically.
- A symmetric algorithm performs encryption and decryption by using a shared secret key.
- A symmetric key is used to encrypt and/or decrypt the actual message.
- Public keys are used to encrypt the symmetric key for secure key exchange.
- A secret key is synonymous with a symmetric key.
- An asymmetric key refers to a public or private key.

So, that is how a hybrid system works. The symmetric algorithm uses a secret key that will be used to encrypt the bulk, or the message, and the asymmetric key encrypts the secret key for transmission.

To ensure that some of these concepts are driven home, ask these questions of yourself without reading the answers provided:

1. If a symmetric key is encrypted with a receiver's public key, what security service(s) is (are) provided?
2. If data is encrypted with the sender's private key, what security service(s) is (are) provided?
3. If the sender encrypts data with the receiver's private key, what security services(s) is (are) provided?
4. Why do we encrypt the message with the symmetric key?
5. Why don't we encrypt the symmetric key with another symmetric key?

Now check your answers:

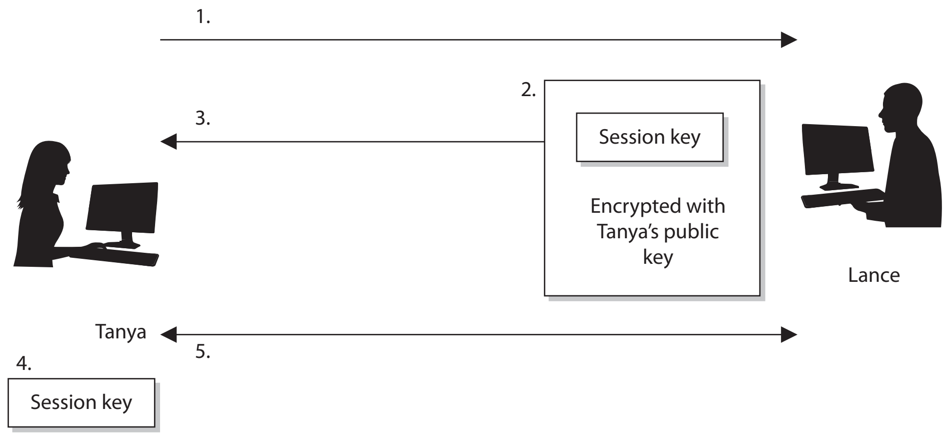
1. Confidentiality, because only the receiver's private key can be used to decrypt the symmetric key, and only the receiver should have access to this private key.
2. Authenticity of the sender and nonrepudiation. If the receiver can decrypt the encrypted data with the sender's public key, then she knows the data was encrypted with the sender's private key.

3. None, because no one but the owner of the private key should have access to it. Trick question.
4. Because the asymmetric key algorithm is too slow.
5. We need to get the necessary symmetric key to the destination securely, which can only be carried out through asymmetric cryptography via the use of public and private keys to provide a mechanism for secure transport of the symmetric key.

Session Keys

A *session key* is a single-use symmetric key that is used to encrypt messages between two users during a communication session. A session key is no different from the symmetric key described in the previous section, but it is only good for one communication session between users.

If Tanya has a symmetric key she uses to always encrypt messages between Lance and herself, then this symmetric key would not be regenerated or changed. They would use the same key every time they communicated using encryption. However, using the same key repeatedly increases the chances of the key being captured and the secure communication being compromised. If, on the other hand, a new symmetric key were generated each time Lance and Tanya wanted to communicate, as shown in Figure 8-15, it would be used only during their one dialogue and then destroyed. If they wanted to communicate an hour later, a new session key would be created and shared.



1) Tanya sends Lance her public key.
2) Lance generates a random session key and encrypts it using Tanya's public key.
3) Lance sends the session key, encrypted with Tanya's public key, to Tanya.
4) Tanya decrypts Lance's message with her private key and now has a copy of the session key.
5) Tanya and Lance use this session key to encrypt and decrypt messages to each other.

Figure 8-15 A session key is generated so all messages can be encrypted during one particular session between users.

A session key provides more protection than static symmetric keys because it is valid for only one session between two computers. If an attacker were able to capture the session key, she would have a very small window of time to use it to try to decrypt messages being passed back and forth.

In cryptography, almost all data encryption takes place through the use of session keys. When you write an e-mail and encrypt it before sending it over the wire, it is actually being encrypted with a session key. If you write another message to the same person one minute later, a brand-new session key is created to encrypt that new message. So if an eavesdropper happens to figure out one session key, that does not mean she has access to all other messages you write and send off.

When two computers want to communicate using encryption, they must first go through a handshaking process. The two computers agree on the encryption algorithms that will be used and exchange the session key that will be used for data encryption. In a sense, the two computers set up a virtual connection between each other and are said to be in session. When this session is done, each computer tears down any data structures it built to enable this communication to take place, releases the resources, and destroys the session key. These things are taken care of by operating systems and applications in the background, so a user would not necessarily need to be worried about using the wrong type of key for the wrong reason. The software will handle this, but it is important for security professionals to understand the difference between the key types and the issues that surround them.

CAUTION Private and symmetric keys should not be available in cleartext. This may seem obvious to you, but there have been several implementations over time that have allowed for this type of compromise to take place.

Unfortunately, we don't always seem to be able to call an apple an apple. In many types of technology, the exact same thing can have more than one name. For example, symmetric cryptography can be referred to as any of the following:

- Secret key cryptography
- Session key cryptography
- Shared key cryptography
- Private key cryptography

We know the difference between secret keys (static) and session keys (dynamic), but what is this "shared key" and "private key" mess? Well, using the term "shared key" makes sense, because the sender and receiver are sharing one single key. It's unfortunate that the term "private key" can be used to describe symmetric cryptography, because it only adds more confusion to the difference between symmetric cryptography (where one symmetric key is used) and asymmetric cryptography (where both a private and public key are used). You just need to remember this little quirk and still understand the difference between symmetric and asymmetric cryptography.

Integrity

Cryptography is mainly concerned with protecting the confidentiality of information. It can also, however, allow us to ensure its integrity. In other words, how can we be certain that a message we receive or a file we download has not been modified? For this type of protection, hash algorithms are required to successfully detect intentional and unintentional unauthorized modifications to data. However, as we will see shortly, it is possible for attackers to modify data, recompute the hash, and deceive the recipient. In some cases, we need a more robust approach to message integrity verification. Let's start off with hash algorithms and their characteristics.

Hashing Functions

A *one-way hash* is a function that takes a variable-length string (a message) and produces a fixed-length value called a *hash value*. For example, if Kevin wants to send a message to Maureen and he wants to ensure the message does not get altered in an unauthorized fashion while it is being transmitted, he would calculate a hash value for the message and append it to the message itself. When Maureen receives the message, she performs the same hashing function Kevin used and then compares her result with the hash value sent with the message. If the two values are the same, Maureen can be sure the message was not altered during transmission. If the two values are different, Maureen knows the message was altered, either intentionally or unintentionally, and she discards the message.

The hashing algorithm is not a secret—it is publicly known. The secrecy of the one-way hashing function is its "one-wayness." The function is run in only one direction, not the other direction. This is different from the one-way function used in public key cryptography, in which security is provided based on the fact that, without knowing a trapdoor, it is very hard to perform the one-way function backward on a message and come up with readable plaintext. However, one-way hash functions are never used in reverse; they create a hash value and call it a day. The receiver does not attempt to reverse the process at the other end, but instead runs the same hashing function one way and compares the two results.

EXAM TIP Keep in mind that hashing is not the same thing as encryption; you can't "decrypt" a hash. You can only run the same hashing algorithm against the same piece of text in an attempt to derive the same hash or fingerprint of the text.

Various Hashing Algorithms

As stated earlier, the goal of using a one-way hash function is to provide a fingerprint of the message. If two different messages produce the same hash value, it would be easier for an attacker to break that security mechanism because patterns would be revealed.

A strong one-hash function should not provide the same hash value for two or more different messages. If a hashing algorithm takes steps to ensure it does not create the same hash value for two or more messages, it is said to be *collision free*.

Algorithm	Description
Message Digest 5 (MD5) algorithm	Produces a 128-bit hash value. More complex than MD4.
Secure Hash Algorithm (SHA)	Produces a 160-bit hash value. Used with Digital Signature Algorithm (DSA).
SHA-1, SHA-256, SHA-384, SHA-512	Updated versions of SHA. SHA-1 produces a 160-bit hash value, SHA-256 creates a 256-bit value, and so on.

Table 8-2 Various Hashing Algorithms Available

Strong cryptographic hash functions have the following characteristics:

- The hash should be computed over the entire message.
- The hash should be a one-way function so messages are not disclosed by their values.
- Given a message and its hash value, computing another message with the same hash value should be impossible.
- The function should be resistant to birthday attacks (explained in the upcoming section "Attacks Against One-Way Hash Functions").

Table 8-2 and the following sections quickly describe some of the available hashing algorithms used in cryptography today.

MD5 *MD5* was created by Ron Rivest in 1991 as a better version of his previous message digest algorithm (MD4). It produces a 128-bit hash, but the algorithm is subject to collision attacks, and is therefore no longer suitable for applications like digital certificates and signatures that require collision attack resistance. It is still commonly used for file integrity checksums, such as those required by some intrusion detection systems, as well as for forensic evidence integrity.

SHA *SHA* was designed by the NSA and published by the National Institute of Standards and Technology (NIST) to be used with the Digital Signature Standard (DSS), which is discussed a bit later in more depth. SHA was designed to be used in digital signatures and was developed when a more secure hashing algorithm was required for U.S. government applications. It produces a 160-bit hash value, or message digest. This is then inputted into an asymmetric algorithm, which computes the signature for a message.

SHA is similar to MD5. It has some extra mathematical functions and produces a 160-bit hash instead of a 128-bit hash, which initially made it more resistant to collision attacks. Newer versions of this algorithm (collectively known as the SHA-2 and SHA-3 families) have been developed and released: SHA-256, SHA-384, and SHA-512. The SHA-2 and SHA-3 families are considered secure for all uses.

Attacks Against One-Way Hash Functions

A strong hashing algorithm does not produce the same hash value for two different messages. If the algorithm does produce the same value for two distinctly different messages, this is called a *collision*. An attacker can attempt to force a collision, which is referred to as a *birthday attack*. This attack is based on the mathematical birthday paradox that exists in standard statistics. Now hold on to your hat while we go through this—it is a bit tricky:

> How many people must be in the same room for the chance to be greater than even that another person has the same birthday as you?
> **Answer:** 253
>
> How many people must be in the same room for the chance to be greater than even that at least two people share the same birthday?
> **Answer:** 23

This seems a bit backward, but the difference is that in the first instance, you are looking for someone with a specific birthday date that matches yours. In the second instance, you are looking for any two people who share the same birthday. There is a higher probability of finding two people who share a birthday than of finding another person who shares your birthday. Or, stated another way, it is easier to find two matching values in a sea of values than to find a match for just one specific value.

Why do we care? The birthday paradox can apply to cryptography as well. Since any random set of 23 people most likely (at least a 50 percent chance) includes two people who share a birthday, by extension, if a hashing algorithm generates a message digest of 60 bits, there is a high likelihood that an adversary can find a collision using only 2^{30} inputs.

The main way an attacker can find the corresponding hashing value that matches a specific message is through a brute-force attack. If he finds a message with a specific hash value, it is equivalent to finding someone with a specific birthday. If he finds two messages with the same hash values, it is equivalent to finding two people with the same birthday.

The output of a hashing algorithm is *n*, and to find a message through a brute-force attack that results in a specific hash value would require hashing 2*n* random messages. To take this one step further, finding two messages that hash to the same value would require review of only 2*n/2* messages.

How Would a Birthday Attack Take Place?

Sue and Joe are going to get married, but before they do, they have a prenuptial contract drawn up that states if they get divorced, then Sue takes her original belongings and Joe takes his original belongings. To ensure this contract is not modified, it is hashed and a message digest value is created.

One month after Sue and Joe get married, Sue carries out some devious activity behind Joe's back. She makes a copy of the message digest value without anyone knowing. Then she makes a new contract that states that if Joe and Sue get a divorce, Sue owns both her own original belongings and Joe's original belongings. Sue hashes this new contract and compares the new message digest value with the message digest value that correlates with the contract. They don't match. So Sue tweaks her contract ever so slightly and creates another message digest value and compares them. She continues to tweak her contract until she forces a collision, meaning her contract creates the same message digest value as the original contract. Sue then changes out the original contract with her new contract and quickly divorces Joe. When Sue goes to collect Joe's belongings and he objects, she shows him that no modification could have taken place on the original document because it still hashes out to the same message digest. Sue then moves to an island.

Hash algorithms usually use message digest sizes (the value of *n*) that are large enough to make collisions difficult to accomplish, but they are still possible. An algorithm that has 256-bit output, like SHA-256, may require approximately 2^{128} computations to break. This means there is a less than 1 in 2^{128} chance that someone could carry out a successful birthday attack.

The main point of discussing this paradox is to show how important longer hashing values truly are. A hashing algorithm that has a larger bit output is less vulnerable to brute-force attacks such as a birthday attack. This is the primary reason why the new versions of SHA have such large message digest values.

Message Integrity Verification

Whether messages are encrypted or not, we frequently want to ensure that they arrive at their destination with no alterations, accidental or deliberate. We can use the principles we've discussed in this chapter to ensure the integrity of our traffic to various degrees of security. Let's look at three increasingly more powerful ways to do this, starting with a simple message digest.

Message Digest

A one-way hashing function takes place without the use of any keys. This means, for example, that if Cheryl writes a message, calculates a message digest, appends the digest to the message, and sends it on to Scott, Bruce can intercept this message, alter Cheryl's message, recalculate another message digest, append it to the message, and send it on to Scott. When Scott receives it, he verifies the message digest, but never knows the message was actually altered by Bruce. Scott thinks the message came straight from Cheryl and was never modified because the two message digest values are the same. This process is depicted in Figure 8-16 and consists of the following steps:

1. The sender writes a message.
2. The sender puts the message through a hashing function, generating a message digest.
3. The sender appends the message digest to the message and sends it to the receiver.
4. The receiver puts the message through a hashing function and generates his own message digest.
5. The receiver compares the two message digest values. If they are the same, the message has not been altered.

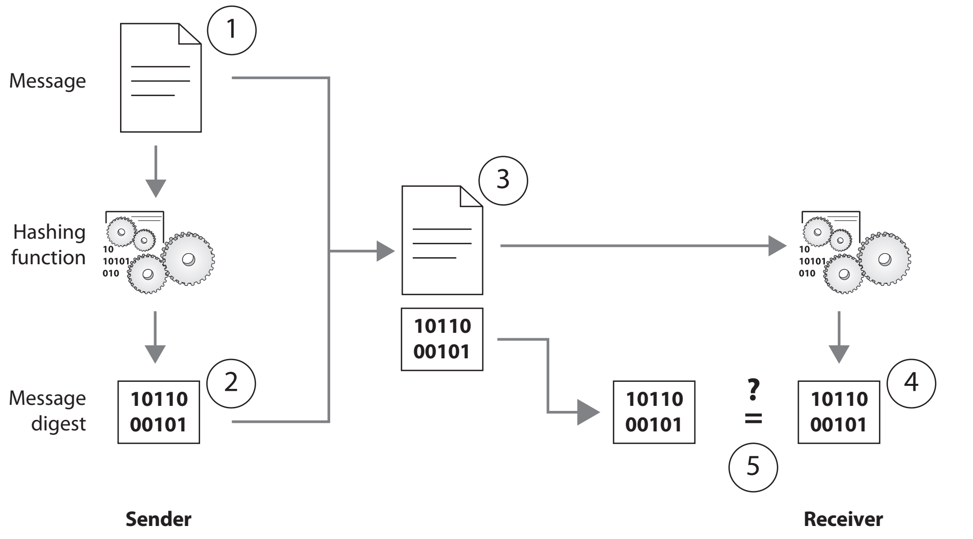


Figure 8-16 Verifying message integrity with a message digest

Message Authentication Code

If Cheryl wanted more protection than just described, she would need to use a *message authentication code (MAC)*, an authentication scheme derived by applying a secret key to a message in some form. This does not mean the symmetric key is used to encrypt the message, though. A good example of a MAC leverages hashing functions and is called a *hash MAC (HMAC)*.

In the previous example, if Cheryl were to use an HMAC function instead of just a plain hashing algorithm, a symmetric key would be concatenated with her message. The result of this process would be put through a hashing algorithm, and the result would be a MAC value. This MAC value would then be appended to her message and sent to Scott. If Bruce were to intercept this message and modify it, he would not have the necessary symmetric key to create the MAC value that Scott will attempt to generate. Figure 8-17 shows the following steps to use an HMAC:

1. The sender writes a message.
2. The sender concatenates a shared secret key with the message and puts them through a hashing function, generating a MAC.
3. The sender appends the MAC value to the message and sends it to the receiver. (Just the message with the attached MAC value. The sender does not send the symmetric key with the message.)
4. The receiver concatenates his copy of the shared secret key with the message and puts the results through a hashing algorithm to generate his own MAC.
5. The receiver compares the two MAC values. If they are the same, the message has not been modified.