Chapter 1

Introduction

Contents

1.1 Fundamental Concepts

In this chapter, we introduce several fundamental concepts in computer security. Topics range from theoretical cryptographic primitives, such as digital signatures, to practical usability issues, such as social engineering. This chapter provides an informal and intuitive description of a variety of topics that will be covered in more detail in the rest of the book.

Existing computer systems may contain legacy features of earlier versions dating back to bygone eras, such as when the Internet was the sole domain of academic researchers and military labs. For instance, assumptions of trust and lack of malicious behavior among network-connected machines, which may have been justifiable in the early eighties, are surprisingly still present in the way the Internet operates today. Such assumptions have led to the growth of Internet-based crime.

An important aspect of computer security is the identification of ***vulnerabilities*** in computer systems, which can, for instance, allow a malicious user to gain access to private data and even assume full control of a machine. Vulnerabilities enable a variety of ***attacks***. Analysis of these attacks can determine the severity of damage that can be inflicted and the likelihood that the attack can be further replicated. Actions that need to be taken to defend against attacks include identifying compromised machines, removing the malicious code, and patching systems to eliminate the vulnerability.

In order to have a secure computer system, sound ***models*** are a first step. In particular, it is important to define the ***security properties*** that must be assured, anticipate the types of ***attacks*** that could be launched, and develop specific defenses. The ***design*** should also take into account usability issues. Indeed, security measures that are difficult to understand and inconvenient to follow will likely lead to failure of adoption. Next, the hardware and software ***implementation*** of a system needs to be rigorously ***tested*** to detect programming errors that introduce vulnerabilities. Once the system is deployed, procedures should be put in place to ***monitor*** the behavior of the system, detect security breaches, and react to them. Finally, security-related ***patches*** to the system must be applied as soon as they become available.

Computer security concepts often are better understood by looking at issues in a broader context. For this reason, this book also includes discussions of the security of various physical and real-world systems, including locks, ATM machines, and passenger screening at airports.

1.1.1 Confidentiality, Integrity, and Availability

Computers and networks are being misused at a growing rate. Spam, phishing, and computer viruses are becoming multibillion-dollar problems, as is identity theft, which poses a serious threat to the personal finances and credit ratings of users, and creates liabilities for corporations. Thus, there is a growing need for broader knowledge of computer security in society as well as increased expertise among information technology professionals. Society needs more security-educated computer professionals, who can successfully defend against and prevent computer attacks, as well as security-educated computer users, who can safely manage their own information and the systems they use.

One of the first things we need to do in a book on computer security is to define our concepts and terms. Classically, information security has been defined in terms of the acronym ***C.I.A.***, which in this case stands for ***confidentiality***, ***integrity***, and ***availability***. (See Figure 1.1.)

Figure 1.1: The C.I.A. concepts: confidentiality, integrity, and availability.

Confidentiality

In the context of computer security, ***confidentiality*** is the avoidance of the unauthorized disclosure of information. That is, confidentiality involves the protection of data, providing access for those who are allowed to see it while disallowing others from learning anything about its content.

Keeping information secret is often at the heart of information security, and this concept, in fact, predates computers. For example, in the first recorded use of cryptography, Julius Caesar communicated commands to his generals using a simple cipher. In his cipher, Caesar took each letter in his message and substituted D for A, E for B, and so on. This cipher can be easily broken, making it an inappropriate tool for achieving confidentiality today. But in its time, the Caesar cipher was probably fairly secure, since most of Caesar's enemies couldn't read Latin anyway.

Nowadays, achieving confidentiality is more of a challenge. Computers are everywhere, and each one is capable of performing operations that could compromise confidentiality. With all of these threats to the confidentiality of information, computer security researchers and system designers have come up with a number of tools for protecting sensitive information. These tools incorporate the following concepts:

- ***Encryption***: the transformation of information using a secret, called an encryption key, so that the transformed information can only be read using another secret, called the decryption key (which may, in some cases, be the same as the encryption key). To be secure, an encryption scheme should make it extremely difficult for someone to determine the original information without use of the decryption key.
- ***Access control***: rules and policies that limit access to confidential information to those people and/or systems with a "need to know." This need to know may be determined by identity, such as a person's name or a computer's serial number, or by a role that a person has, such as being a manager or a computer security specialist.
- ***Authentication***: the determination of the identity or role that someone has. This determination can be done in a number of different ways, but it is usually based on a combination of something the person has (like a smart card or a radio key fob storing secret keys), something the person knows (like a password), and something the person is (like a human with a fingerprint). The concept of authentication is schematically illustrated in Figure 1.2.
- ***Authorization***: the determination if a person or system is allowed access to resources, based on an access control policy. Such authorizations should prevent an attacker from tricking the system into letting him have access to protected resources.

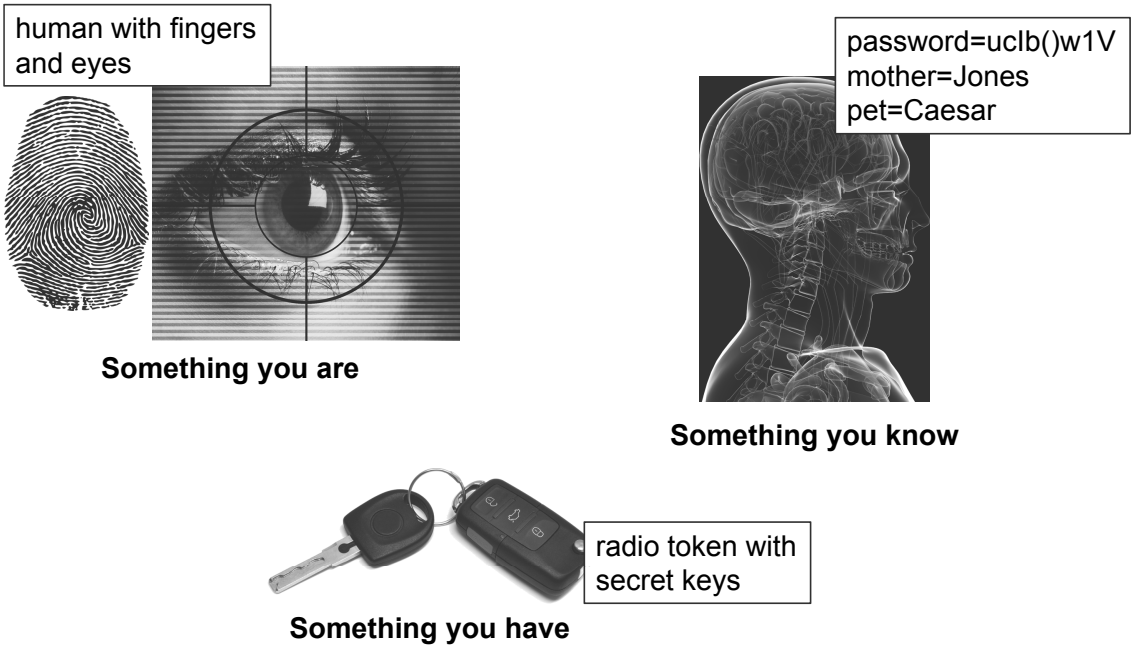


Figure 1.2: Three foundations for authentication.

- ***Physical security***: the establishment of physical barriers to limit access to protected computational resources. Such barriers include locks on cabinets and doors, the placement of computers in windowless rooms, the use of sound dampening materials, and even the construction of buildings or rooms with walls incorporating copper meshes (called ***Faraday cages***) so that electromagnetic signals cannot enter or exit the enclosure.

When we visit a web page that asks for our credit card number and our Internet browser shows a little lock icon in the corner, there is a lot that has gone on in the background to help ensure the confidentiality of our credit card number. In fact, a number of tools have probably been brought to bear here. Our browser begins the process by performing an authentication procedure to verify that the web site we are connecting to is indeed who it says it is. While this is going on, the web site might itself be checking that our browser is authentic and that we have the appropriate authorizations to access this web page according to its access control policy. Our browser then asks the web site for an encryption key to encrypt our credit card, which it then uses so that it only sends our credit card information in encrypted form. Finally, once our credit card number reaches the server that is providing this web site, the data center where

the server is located should have appropriate levels of physical security, access policies, and authorization and authentication mechanisms to keep our credit card number safe. We discuss these topics in some detail in this book.

For instance, in Section 2.4.2, we study a number of real demonstrated risks to physical eavesdropping. For example, researchers have shown that one can determine what someone is typing just by listening to a recording of their key strokes. Likewise, experiments show that it is possible to reconstruct the image of a computer screen either by monitoring its electromagnetic radiation or even from a video of a blank wall that the screen is shining on. Thus, physical security is an information security concept that should not be taken for granted.

Integrity

Another important aspect of information security is ***integrity***, which is the property that information has not be altered in an unauthorized way.

The importance of integrity is often demonstrated to school children in the ***Telephone game***. In this game, a group of children sit in a circle and the person who is "it" whispers a message in the ear of his or her neighbor on the right. Each child in the circle then waits to listen to the message from his or her neighbor on the left. Once a child has received the message, he or she then whispers this same message to their neighbor on the right. This message passing process continues until the message goes full circle and returns to the person who is "it." At that point, the last person to hear the message says the message out loud so that everyone can hear it. Typically, the message has been so mangled by this point that it is a great joke to all the children, and the game is repeated with a new person being "it." And, with each repeat play, the game reinforces that this whispering process rarely ever preserves data integrity. Indeed, could this be one of the reasons we often refer to rumors as being "whispered"?

There are a number of ways that data integrity can be compromised in computer systems and networks, and these compromises can be benign or malicious. For example, a benign compromise might come from a storage device being hit with a stray cosmic ray that flips a bit in an important file, or a disk drive might simply crash, completely destroying some of its files. A malicious compromise might come from a computer virus that infects our system and deliberately changes some the files of our operating system, so that our computer then works to replicate the virus and send it to other computers. Thus, it is important that computer systems provide tools to support data integrity.

The previously mentioned tools for protecting the confidentiality of information, denying access to data to users without appropriate access rights, also help prevent data from being modified in the first place. In addition, there are several tools specifically designed to support integrity, including the following:

- ***Backups***: the periodic archiving of data. This archiving is done so that data files can be restored should they ever be altered in an unauthorized or unintended way.

- ***Checksums***: the computation of a function that maps the contents of a file to a numerical value. A checksum function depends on the entire contents of a file and is designed in a way that even a small change to the input file (such as flipping a single bit) is highly likely to result in a different output value. Checksums are like trip-wires—they are used to detect when a breach to data integrity has occurred.

- ***Data correcting codes***: methods for storing data in such a way that small changes can be easily detected and automatically corrected. These codes are typically applied to small units of storage (e.g., at the byte level or memory word level), but there are also data-correcting codes that can be applied to entire files as well.

These tools for achieving data integrity all possess a common trait—they use ***redundancy***. That is, they involve the replication of some information content or functions of the data so that we can detect and sometimes even correct breaches in data integrity.

In addition, we should stress that it is not just the content of a data file that needs to be maintained with respect to integrity. We also need to protect the ***metadata*** for each data file, which are attributes of the file or information about access to the file that are not strictly a part of its content. Examples of metadata include the user who is the owner of the file, the last user who has modified the file, the last user who has read the file, the dates and times when the file was created and last modified and accessed, the name and location of the file in the file system, and the list of users or groups who can read or write the file. Thus, changing any metadata of a file should be considered a violation of its integrity.

For example, a computer intruder might not actually modify the content of any user files in a system he has infiltrated, but he may nevertheless be modifying metadata, such as access time stamps, by looking at our files (and thereby compromising their confidentiality if they are not encrypted). Indeed, if our system has integrity checks in place for this type of metadata, it may be able to detect an intrusion that would have otherwise gone unnoticed.

Availability

Besides confidentiality and integrity, another important property of information security is ***availability***, which is the property that information is accessible and modifiable in a timely fashion by those authorized to do so.

Information that is locked in a cast-iron safe high on a Tibetan mountain and guarded round the clock by a devoted army of ninjas may be considered safe, but it is not practically secure from an information security perspective if it takes us weeks or months to reach it. Indeed, the quality of some information is directly associated with how available it is.

For example, stock quotes are most useful when they are fresh. Also, imagine the damage that could be caused if someone stole our credit card and it took weeks before our credit card company could notify anyone, because its list of stolen numbers was unavailable to merchants. Thus, as with confidentiality and integrity, computer security researchers and system designers have developed a number of tools for providing availability, including the following:

- ***Physical protections***: infrastructure meant to keep information available even in the event of physical challenges. Such protections can include buildings housing critical computer systems to be constructed to withstand storms, earthquakes, and bomb blasts, and outfitted with generators and other electronic equipment to be able to cope with power outages and surges.

- ***Computational redundancies***: computers and storage devices that serve as fallbacks in the case of failures. For example, ***redundant arrays of inexpensive disks*** (***RAID***) use storage redundancies to keep data available to their clients. Also, web servers are often organized in multiples called "farms" so that the failure of any single computer can be dealt with without degrading the availability of the web site.

Because availability is so important, an attacker who otherwise doesn't care about the confidentiality or integrity of data may choose to attack its availability. For instance, a thief who steals lots of credit cards might wish to attack the availability of the list of stolen credit cards that is maintained and broadcast by a major credit card company. Thus, availability forms the third leg of support for the vital C.I.A. triad of information security.

1.1.2 Assurance, Authenticity, and Anonymity

In addition to the classic C.I.A. concepts of confidentiality, integrity, and availability, discussed in the previous section, there are a number of additional concepts that are also important in modern computer security applications. These concepts can likewise be characterized by a three-letter acronym, ***A.A.A.***, which in this context refers to ***assurance***, ***authenticity***, and ***anonymity***. (See Figure 1.3.)

Figure 1.3: The A.A.A. concepts: assurance, authenticity, and anonymity. Note that unlike the C.I.A. concepts, the A.A.A. concepts are independent of each other.

Assurance

Assurance, in the context of computer security, refers to how trust is provided and managed in computer systems. Admittedly, trust itself is difficult to quantify, but we know it involves the degree to which we have confidence that people or systems are behaving in the way we expect.

Furthermore, trust involves the interplay of the following:

- ***Policies*** specify behavioral expectations that people or systems have for themselves and others. For example, the designers of an online music system may specify policies that describe how users can access and copy songs.
- ***Permissions*** describe the behaviors that are allowed by the agents that interact with a person or system. For instance, an online music store may provide permissions for limited access and copying to people who have purchased certain songs.
- ***Protections*** describe mechanisms put in place to enforce permissions and polices. Using our running example of an online music store, we could imagine that such a system would build in protections to prevent people from unauthorized access and copying of its songs.

Assurance doesn't just go from systems to users, however. A user providing her credit card number to an online music system may expect the system to abide by its published policies regarding the use of credit card numbers, she might grant permission to the system to make small charges to her card for music purchases, and she may also have a protection system in place with her credit card company so that she would not be liable for any fraudulent charges on her card. Thus, with respect to computer systems, assurance involves the management of trust in two directions—from users to systems and from systems to users.

The designers of computer systems want to protect more than just the confidentiality, integrity, and availability of information. They also want to protect and manage the resources of these systems and they want to make sure users don't misuse these resources. Put in negative terms, they want, for example, to keep unauthorized people from using their CPUs, memory, and networks, even if no information is compromised in terms of the C.I.A. framework. Thus, designers want assurance that the people using the resources of their systems are doing so in line with their policies.

Likewise, managing information in a computer system can also go beyond the C.I.A. framework, in that we may wish to manage the way that information is used. For instance, if a user of an online movie rental system has rented an electronic copy of a movie, we might want to allow that user to watch it only a fixed number of times or we might want to insist that he watch it within the next 30 days. Designers of music playing devices and applications may likewise wish to allow users to make a few backup copies of their music for personal use, but restrict copying so that they cannot make hundreds of pirate CDs from their music files.

Thus, ***trust management*** deals with the design of effective, enforceable policies, methods for granting permissions to trusted users, and the components that can enforce those policies and permissions for protecting and managing the resources in the system. The policies can be complicated, like the contracts used in license agreements for movies, or they can be fairly simple, like a policy that says that only the owner of a computer is allowed to use its CPU. So it is best if a system designer comes up with policies that are easy to enforce and permissions that are easy to comply with.

Another important part of system assurance involves ***software engineering***. The designers of a system need to know that the software that implements their system is coded so that it conforms to their design. There are, in fact, plenty of examples of systems that were designed correctly "on paper," but which worked incorrectly because those designs were not implemented correctly.

A classic example of such an incorrect implementation involves the use of pseudo-random number generators in security designs. A ***pseudo-random number generator*** (***PRNG***) is a program that returns a sequence of numbers that are statistically random, given a starting number, called the ***seed***, which is assumed to be random. The designer of a system might specify that a PRNG be used in a certain context, like encryption, so that each encryption will be different. But if the person actually writing the program makes the mistake of always using the same seed for this pseudo-random number generator, then the sequences of so-called pseudo-random numbers will always be the same. Thus, the designers of secure systems should not only have good designs, they should also have good ***specifications*** and ***implementations***.

Placing trust in a system is more problematic. Users typically don't have the same computational power as the servers employed by such systems. So the trust that users place in a system has to come from the limited amount of computing that they can do, as well as the legal and reputational damage that the user can do to the company that owns the system if it fails to live up to the user's trust.

As mentioned above, when an Internet browser "locks the lock" to indicate that communication with a web site is now secure, it is performing a number of computational services on behalf of the user. It is encrypting the session so that no outsiders can eavesdrop on the communication and, if it is configured correctly, the browser has done some rudimentary checks to make sure the web site is being run by the company that it claims is its owner. So long as such knowledge can be enforced, then the user at least has some recourse should she be cheated by the web site—she can take evidence of this bad behavior to court or to a reputation opinion web site.

Authenticity

With so many online services providing content, resources, and even computational services, there is a need for these systems to be able to enforce their policies. Legally, this requires that we have an electronic way of enforcing contracts. That is, when someone says that they are going to buy a song from an online music store, there should be some way to enforce this commitment. Likewise, when an online movie store commits to allowing a user to rent a movie and watch it sometime in the following 30 days, there should be some enforceable way for that user to know that the movie will be available for that entire time.

Authenticity is the ability to determine that statements, policies, and permissions issued by persons or systems are genuine. If such things can be faked, there is no way to enforce the implied contracts that people and systems engage in when buying and selling items online. Also, a person or system could claim that they did not make such a commitment—they could say that the commitment was made by someone pretending to be them.

Formally, we say that a protocol that achieves such types of authenticity demonstrates nonrepudiation. ***Nonrepudiation*** is the property that authentic statements issued by some person or system cannot be denied.

The chief way that the nonrepudiation property is accomplished is through the use of ***digital signatures***. These are cryptographic computations that allow a person or system to commit to the authenticity of their documents in a unique way that achieves nonrepudiation. We give a more formal definition of digital signatures in Section 1.3.2 and we discuss specific implementations of digital signatures elsewhere in this book, but here it is sufficient to know that a digital signature provides a computational analogue to real-world, so-called blue-ink signatures.

In fact, digital signatures typically have some additional benefits over blue-ink signatures, in that digital signatures also allow to check the integrity of signed documents. That is, if a document is modified, then the signature on that document becomes invalid. An important requirement of authenticity, therefore, is that we need to have reliable ways of electronically identifying people, which is a topic we discuss in Section 1.3 on cryptographic primitives.

The concept we discuss next is instead on the necessary flip side of creating systems that are so tied to personal identities, which is what is required for digital signatures to make any sense.

Anonymity

When people interact with systems in ways that involve their real-world identities, this interaction can have a number of positive benefits, as outlined above. There is an unfortunate side effect from using personal identities in such electronic transactions, however. We end up spreading our identity across a host of digital records, which ties our identity to our medical history, purchase history, legal records, email communications, employment records, etc. Therefore, we have a need for ***anonymity***, which is the property that certain records or transactions not to be attributable to any individual.

If organizations need to publish data about their members or clients, we should expect that they do so in a privacy-preserving fashion, using some of the following tools:

- ***Aggregation***: the combining of data from many individuals so that disclosed sums or averages cannot be tied to any individual. For example, the U.S. Census routinely publishes population breakdowns of zip-code regions by ethnicity, salary, age, etc., but it only does so when such disclosures would not expose details about any individual.

- ***Mixing***: the intertwining of transactions, information, or communications in a way that cannot be traced to any individual. This technique is somewhat technical, but it involves systems that can mix data together in a quasi-random way so that transactions or searches can still be performed, but without the release of any individual identity.

- ***Proxies***: trusted agents that are willing to engage in actions for an individual in a way that cannot be traced back to that person. For example, Internet searching proxies are web sites that themselves provide an Internet browser interface, so that individuals can visit web sites that they might be blocked from, for instance, because of the country they are located in.

- ***Pseudonyms***: fictional identities that can fill in for real identities in communications and transactions, but are otherwise known only to a trusted entity. For example, many online social networking sites allow users to interact with each other using pseudonyms, so that they can communicate and create an online persona without revealing their actual identity.

Anonymity should be a goal that is provided with safeguards whenever possible and appropriate.

1.1.3 Threats and Attacks

Having discussed the various goals of computer security, we should now mention some of the threats and attacks that can compromise these goals:

- ***Eavesdropping***: the interception of information intended for someone else during its transmission over a communication channel. Examples include packet sniffers, which monitor nearby Internet traffic, such as in a wireless access location. This is an attack on confidentiality.

- ***Alteration***: unauthorized modification of information. Examples of alteration attacks include the ***man-in-the-middle*** attack, where a network stream is intercepted, modified, and retransmitted, and computer viruses, which modify critical system files so as to perform some malicious action and to replicate themselves. Alteration is an attack on data integrity.

- ***Denial-of-service***: the interruption or degradation of a data service or information access. Examples include email ***spam***, to the degree that it is meant to simply fill up a mail queue and slow down an email server. Denial of service is an attack on availability.

- ***Masquerading***: the fabrication of information that is purported to be from someone who is not actually the author. Examples of masquerading attacks include ***phishing***, which creates a web site that looks like a real bank or other e-commerce site, but is intended only for gathering passwords, and ***spoofing***, which may involve sending on a network data packets that have false return addresses. Masquerading is an attack on authenticity, and, in the case of phishing, an attempt to compromise confidentiality and/or anonymity.

- ***Repudiation***: the denial of a commitment or data receipt. This involves an attempt to back out of a contract or a protocol that requires the different parties to provide receipts acknowledging that data has been received. This is an attack on assurance.

- ***Correlation*** and ***traceback***: the integration of multiple data sources and information flows to determine the source of a particular data stream or piece of information. This is an attack on anonymity.

There are other types of attacks as well, such as military-level attacks meant to break cryptographic secrets. In addition, there are composite attacks, which combine several of the above types of attacks into one. But those listed above are among the most common types of attacks.

1.1.4 Security Principles

We conclude this section by presenting the ten ***security principles*** listed in a classic 1975 paper by Saltzer and Schroeder. In spite of their age, these principles remain important guidelines for securing today's computer systems and networks.

1. ***Economy of mechanism***. This principle stresses simplicity in the design and implementation of security measures. While applicable to most engineering endeavors, the notion of simplicity is especially important in the security domain, since a simple security framework facilitates its understanding by developers and users and enables the efficient development and verification of enforcement methods for it. Economy of mechanism is thus closely related to implementation and usability issues, which we touch on in Section 1.4.

2. ***Fail-safe defaults***. This principle states that the default configuration of a system should have a conservative protection scheme. For example, when adding a new user to an operating system, the default group of the user should have minimal access rights to files and services. Unfortunately, operating systems and applications often have default options that favor usability over security. This has been historically the case for a number of popular applications, such as web browsers that allow the execution of code downloaded from the web server. Many popular access control models, such as those outlined in Section 1.2, are based on the assumption of a fail-safe permission default. Namely, if no access rights are explicitly specified for a certain subject-object pair (s, o) (e.g., an empty cell of an access control matrix), then all types of access to object o are denied for subject s.

3. ***Complete mediation***. The idea behind this principle is that every access to a resource must be checked for compliance with a protection scheme. As a consequence, one should be wary of performance improvement techniques that save the results of previous authorization checks, since permissions can change over time. For example, an online banking web site should require users to sign on again after a certain amount of time, say, 15 minutes, has elapsed. File systems vary in the way access checks are performed by an application. For example, it can be risky if permissions are checked the first time a program requests access to a file, but subsequent accesses to the same file are not checked again while the application is still running.

4. ***Open design***. According to this principle, the security architecture and design of a system should be made publicly available. Security should rely only on keeping cryptographic keys secret. Open design allows for a system to be scrutinized by multiple parties, which leads to the early discovery and correction of security vulnerabilities caused by design errors. Making the implementation of the system available for inspection, such as in open source software, allows for a more detailed review of security features and a more direct process for fixing software bugs. The open design principle is the opposite of the approach known as ***security by obscurity***, which tries to achieve security by keeping cryptographic algorithms secret and which has been historically used without success by several organizations. Note that while it is straightforward to change a compromised cryptographic key, it is usually infeasible to modify a system whose security has been threatened by a leak of its design.

5. ***Separation of privilege***. This principle dictates that multiple conditions should be required to achieve access to restricted resources or have a program perform some action. In the years since the publishing of the Saltzer-Schroeder paper, the term has come to also imply a separation of the components of a system, to limit the damage caused by a security breach of any individual component.

6. ***Least privilege***. Each program and user of a computer system should operate with the bare minimum privileges necessary to function properly. If this principle is enforced, abuse of privileges is restricted, and the damage caused by the compromise of a particular application or user account is minimized. The military concept of ***need-to-know*** information is an example of this principle. When this principle is ignored, then extra damage is possible from security breaches. For instance, malicious code injected by the attacker into a web server application running with full administrator privileges can do substantial damage to the system. Instead, applying the least privilege principle, the web server application should have the minimal set of permissions that are needed for its operation.

7. ***Least common mechanism***. In systems with multiple users, mechanisms allowing resources to be shared by more than one user should be minimized. For example, if a file or application needs to be accessed by more than one user, then these users should have separate channels by which to access these resources, to prevent unforeseen consequences that could cause security problems.

8. ***Psychological acceptability***. This principle states that user interfaces should be well designed and intuitive, and all security-related settings should adhere to what an ordinary user might expect. Differences in the behavior of a program and a user's expectations may cause security problems such as dangerous misconfigurations of software, so this principle seeks to minimize these differences. Several email applications incorporate cryptographic techniques (Section 1.3) for encrypting and digitally signing email messages, but, despite their broad applicability, such powerful cryptographic features are rarely used in practice. One of the reasons for this state of affairs is believed to be the clumsy and nonintuitive interfaces so far provided by existing email applications for the use of cryptographic features.

9. ***Work factor***. According to this principle, the cost of circumventing a security mechanism should be compared with the resources of an attacker when designing a security scheme. A system developed to protect student grades in a university database, which may be attacked by snoopers or students trying to change their grades, probably needs less sophisticated security measures than a system built to protect military secrets, which may be attacked by government intelligence organizations. Saltzer and Schroeder admit that the work factor principle translates poorly to electronic systems, where it is difficult to determine the amount of work required to compromise security. In addition, technology advances so rapidly that intrusion techniques considered infeasible at a certain time may become trivial to perform within a few years. For example, as discussed in Section 1.4.2, brute-force password cracking is becoming increasingly feasible to perform on an inexpensive personal computer.

10. ***Compromise recording***. Finally, this principle states that sometimes it is more desirable to record the details of an intrusion than to adopt more sophisticated measures to prevent it. Internet-connected surveillance cameras are a typical example of an effective compromise record system that can be deployed to protect a building in lieu of reinforcing doors and windows. The servers in an office network may maintain logs for all accesses to files, all emails sent and received, and all web browsing sessions. Again, the compromise recording principle does not hold as strongly on computer systems, since it may be difficult to detect intrusion and adept attackers may be able to remove their tracks on the compromised machine (e.g., by deleting log entries).

The Ten Security Principles

These ten security principles are schematically illustrated in Figure 1.4. As mentioned above, these principles have been born out time and again as being fundamental for computer security. Moreover, as suggested by the figure, these principles work in concert to protect computers and information. For example, economy of mechanism naturally aids open design, since a simple system is easier to understand and an open system publically demonstrates security that comes from such a simple system.

Figure 1.4: The ten security principles by Saltzer and Schroeder.

1.2 Access Control Models

One of the best ways to defend against attacks is to prevent them in the first place. By providing for a rigorous means of determining who has access to various pieces of information, we can often prevent attacks on confidentiality, integrity, and anonymity. In this section, we discuss some of the most popular means for managing access control.

All of the models assume that there are data managers, data owners, or system administrators who are defining the access control specifications. The intent is that these folks should be restricting access to those who have a need to access and/or modify the information in question. That is, they should be applying the principle of ***least privilege***.

1.2.1 Access Control Matrices

A useful tool for determining access control rights is the ***access control matrix***, which is a table that defines permissions. Each row of this table is associated with a ***subject***, which is a user, group, or system that can perform actions. Each column of the table is associated with an ***object***, which is a file, directory, document, device, resource, or any other entity for which we want to define access rights. Each cell of the table is then filled with the access rights for the associated combination of subject and object. Access rights can include actions such as reading, writing, copying, executing, deleting, and annotating. An empty cell means that no access rights are granted. We show an example access control matrix for part of a fictional file system and a set of users in Table 1.1.

	/etc/passwd	**/usr/bin/**	**/u/roberto/**	**/admin/**
root	read, write	read, write, exec	read, write, exec	read, write, exec
mike	read	read, exec		
roberto	read	read, exec	read, write, exec	
backup	read	read, exec	read, exec	read, exec
...	...	...	...	...

Table 1.1: An example access control matrix. This table lists read, write, and execution (exec) access rights for each of four fictional users with respect to one file, /etc/passwd, and three directories.

Advantages

The nice thing about an access control matrix is that it allows for fast and easy determination of the access control rights for any subject-object pair—just go to the cell in the table for this subject's row and this object's column. The set of access control rights for this subject-object pair is sitting right there, and locating a record of interest can be done with a single operation of looking up a cell in a matrix. In addition, the access control matrix gives administrators a simple, visual way of seeing the entire set of access control relationships all at once, and the degree of control is as specific as the granularity of subject-object pairs. Thus, there are a number of advantages to this access control model.

Disadvantages

There is a fairly big disadvantage to the access control matrix, however—it can get really big. In particular, if we have n subjects and m objects, then the access control matrix has n rows, m columns, and $n \cdot m$ cells. For example, a reasonably sized computer server could easily have 1,000 subjects, who are its users, and 1,000,000 objects, which are its files and directories. But this would imply an access control matrix with 1 billion cells! It is hard to imagine there is a system administrator anywhere on the planet with enough time and patience to fill in all the cells for a table this large! Also, nobody would be able to view this table all at once.

To overcome the lack of scalability of the access control matrix, computer security researchers and system administrators have suggested a number of alternatives to the access control matrix. We discuss three of these models in the remaining part of this section. In particular, we discuss access control lists, capabilities, and role-based access control. Each of these models provides the same functionality as the access control matrix, but in ways that reduce its complexity.

1.2.2 Access Control Lists

The ***access control list*** (***ACL***) model takes an object-centered approach. It defines, for each object, o, a list, L, called o's access control list, which enumerates all the subjects that have access rights for o and, for each such subject, s, gives the access rights that s has for object o.

Essentially, the ACL model takes each column of the access control matrix and compresses it into a list by ignoring all the subject-object pairs in that column that correspond to empty cells. (See Figure 1.5.)

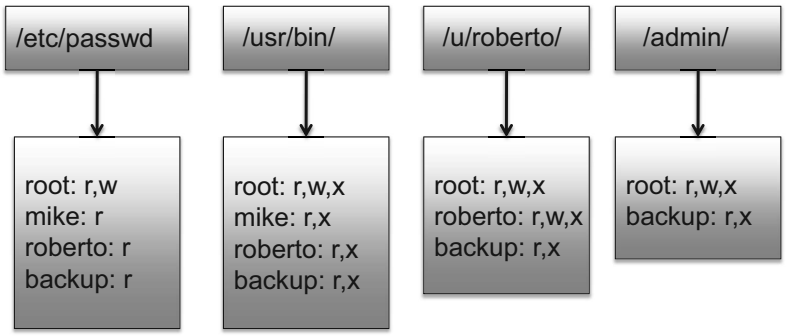


Figure 1.5: The access control lists (ACLs) corresponding to the access control matrix of Table 1.1. We use the shorthand notation of r=read, w=write, and x=execute.

Advantages

The main advantage of ACLs over access control matrices is size. The total size of all the access control lists in a system will be proportional to the number of nonempty cells in the access control matrix, which is expected to be much smaller than the total number of cells in the access control matrix.

Another advantage of ACLs, with respect to secure computer systems, is that the ACL for an object can be stored directly with that object as part of its metadata, which is particularly useful for file systems. That is, the header blocks for files and directories can directly store the access control list of that file or directory. Thus, if the operating system is trying to decide if a user or process requesting access to a certain directory or file in fact has that access right, the system need only consult the ACL of that object.

Disadvantages

The primary disadvantage of ACLs, however, is that they don't provide an efficient way to enumerate all the access rights of a given subject. In order to determine all the access rights for a given subject, s, a secure system based on ACLs would have to search the access control list of every object looking for records involving s. That is, determining such information requires a complete search of all the ACLs in the system, whereas the similar computation with an access control matrix simply involves examining the row for subject s.

Unfortunately, this computation is sometimes necessary. For example, if a subject is to be removed from a system, the administrator needs to remove his or her access rights from every ACL they are in. But if there is no way to know all the access rights for a given subject, the administrator has no choice but to search all the ACLs to find any that contain that subject.

1.2.3 Capabilities

Another approach, known as ***capabilities***, takes a subject-centered approach to access control. It defines, for each subject s, the list of the objects for which s has nonempty access control rights, together with the specific rights for each such object. Thus, it is essentially a list of cells for each row in the access control matrix, compressed to remove any empty cells. (See Figure 1.6.)

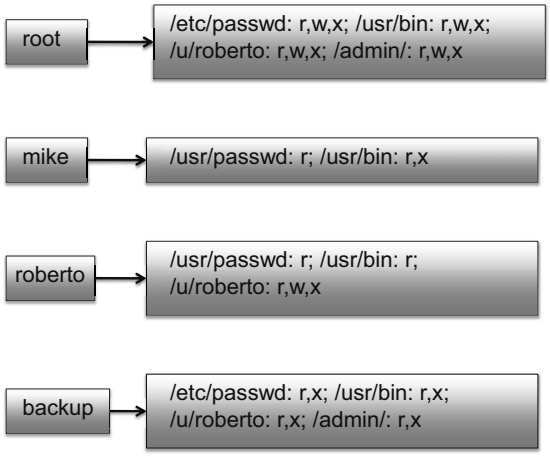


Figure 1.6: The capabilities corresponding to the access control matrix of Table 1.1. We use the shorthand notation of r=read, w=write, and x=execute.

Advantages

The capabilities access control model has the same advantage in space over the access control matrix as the access control list model has. Namely, a system administrator only needs to create and maintain access control relationships for subject-object pairs that have nonempty access control rights. In addition, the capabilities model makes it easy for an administrator to quickly determine for any subject all the access rights that that subject has. Indeed, all she needs to do is read off the capabilities list for that subject. Likewise, each time a subject s requests a particular access right for an object o, the system needs only to examine the complete capabilities list for s looking for o. If s has that right for o, then it is granted it. Thus, if the size of the capabilities list for a subject is not too big, this is a reasonably fast computation.

Disadvantages

The main disadvantage of capabilities is that they are not associated directly with objects. Thus, the only way to determine all the access rights for an object o is to search all the capabilities lists for all the subjects. With the access control matrix, such a computation would simply involve searching the column associated with object o.

1.2.4 Role-Based Access Control

Independent of the specific data structure that represents access control rights, is another approach to access control, which can be used with any of the structures described above. In ***role-based access control*** (***RBAC***), administrators define ***roles*** and then specify access control rights for these roles, rather than for subjects directly.

So, for example, a file system for a university computer science department could have roles for "faculty," "student," "administrative personnel," "administrative manager," "backup agent," "lab manager," "system administrator," etc. Each role is granted the access rights that are appropriate for the class of users associated with that role. For instance, a backup agent should have read and execute access for every object in the file system, but write access only to the backup directory.

Once roles are defined and access rights are assigned to role-object pairs, subjects are assigned to various roles. The access rights for any subject are the union of the access rights for the roles that they have. For example, a student who is working part time as a system administrator's assistant to perform backups on a departmental file system would have the roles "student" and "backup agent," and she would have the union of rights that are conferred to these two roles. Likewise, a professor with the roles "faculty" and"lab manager" would get all the access rights in the union of these roles. The professor who serves as department chair would have in addition other roles, including "administrative manager" and "system administrator."

Role Hierarchies

In addition, a hierarchy can be defined over roles so that access rights propagate up the hierarchy. Namely, if a role R_1 is above role R_2 in the hierarchy, then R_1 inherits the access rights of R_2. That is, the access rights of R_1 include those of R_2. For example, in the role hierarchy for a computer science department, role "system administrator," would be above

role "backup agent" and role "administrative manager," would be above role "administrative personnel."

Hierarchies of roles simplify the definition and management of permissions thanks to the inheritance property. Thy are the main feature that distinguishes roles from groups of users. An example of hierarchy of roles for a computer science department is shown in Figure 1.7. The role-based access control model is described in more detail in Section 9.2.3.

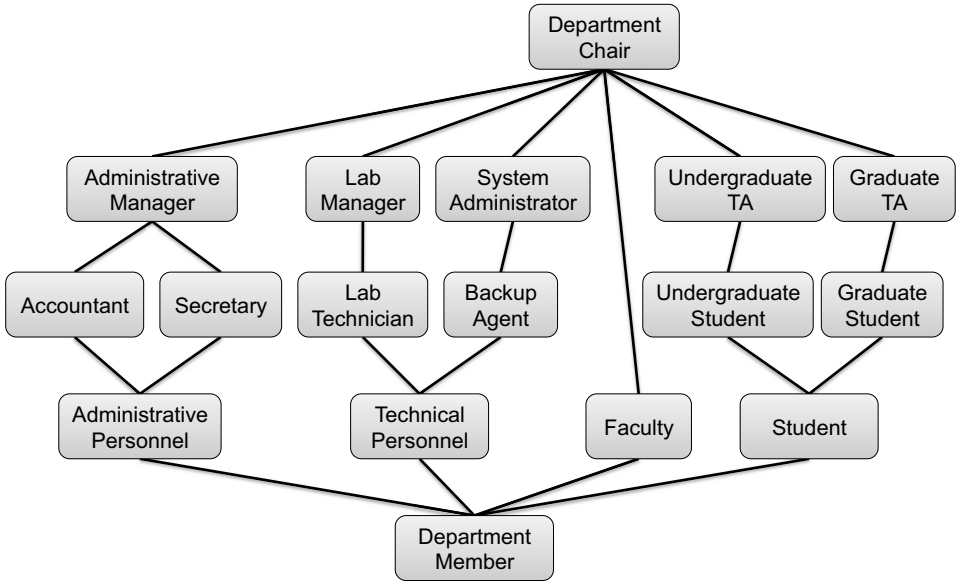


Figure 1.7: Example of hierarchy of roles for a computer science department.

Advantages and Disadvantages

The advantage of role-based access control is that, no matter which access control framework is being used to store access control rights, the total number of rules to keep track of is reduced. That is, the total set of roles should be much smaller than the set of subjects; hence, storing access rights just for roles is more efficient. And the overhead for determining if a subject s has a particular right is small, for all the system needs to do is to determine if one of the roles for s has that access right.

The main disadvantage of the role-based access control model is that it is not implemented in current operating systems.

1.3 Cryptographic Concepts

Computer security policies are worthless if we don't have ways of enforcing them. Laws and economics can play an important role in deterring attacks and encouraging compliance, respectively. However, technological solutions are the primary mechanism for enforcing security policies and achieving security goals.

That's were cryptography comes in. We can use cryptographic techniques to achieve a broad range of security goals, including some that at first might even seem to be impossible. In this section, we give an overview of several fundamental cryptographic concepts. A more detailed coverage of cryptographic principles and techniques is provided in Chapter 8.

1.3.1 Encryption

Traditionally, ***encryption*** is described as a means to allow two parties, customarily called Alice and Bob, to establish confidential communication over an insecure channel that is subject to eavesdropping. It has grown to have other uses and applications than this simple scenario, but let us nevertheless start with the scenario of Alice and Bob wanting to communicate in a confidential manner, as this gives us a foundation upon which we can build extensions later.

Suppose, then, that Alice has a message, M, that she wishes to communicate confidentially to Bob. The message M is called the ***plaintext***, and it is not to be transmitted in this form as it can be observed by other parties while in transit. Instead, Alice will convert plaintext M to an encrypted form using an encryption algorithm E that outputs a ***ciphertext*** C for M. This encryption process is denoted by

$$C = E(M).$$

Ciphertext C will be what is actually transmitted to Bob. Once Bob has received C, he applies a decryption algorithm D to recover the original plaintext M from ciphertext C. This decryption process is denoted

$$M = D(C).$$

The encryption and decryption algorithms are chosen so that it is infeasible for someone other than Alice and Bob to determine plaintext M from ciphertext C. Thus, ciphertext C can be transmitted over an insecure channel that can be eavesdropped by an adversary.

Cryptosystems

The decryption algorithm must use some secret information known to Bob, and possibly also to Alice, but no other party. This is typically accomplished by having the decryption algorithm use as an auxiliary input a secret number or string called ***decryption key***. In this way, the decryption algorithm itself can be implemented by standard, publicly available software and only the decryption key needs to remain secret. Similarly, the encryption algorithm uses as auxiliary input an ***encryption key***, which is associated with the decryption key. Unless it is infeasible to derive the decryption key from the encryption key, the encryption key should be kept secret as well. That is encryption in a nutshell.

But before Alice and Bob even start performing this encrypted communication, they need to agree on the ground rules they will be using. Specifically, a ***cryptosystem*** consists of seven components:

1. The set of possible plaintexts
2. The set of possible ciphertexts
3. The set of encryption keys
4. The set of decryption keys
5. The correspondence between encryption keys and decryption keys
6. The encryption algorithm to use
7. The decryption algorithm to use

Let c be a character of the classical Latin alphabet (which consists of 23 characters) and k be an integer in the range $[-22, +22]$. We denote with $s(c, k)$ the circular shift by k of character c in the Latin alphabet. The shift is forward when $k > 0$ and backward for $k < 0$. For example, $s(D, 3) = G$, $s(R, -2) = P$, $s(Z, 2) = B$, and $s(C, -3) = Z$. In the ***Caesar cipher***, the set of plaintexts and the set of ciphertexts are the strings consisting of characters from the Latin alphabet. The set of encryption keys is $\{3\}$, that is, the set consisting of number 3. The set of decryption keys is $\{-3\}$, that is, the set consisting of number -3. The encryption algorithm consists of replacing each character x in the plaintext with $s(x, e)$, where $e = 3$ is the encryption key. The decryption algorithm consists of replacing each character x in the plaintext with $s(x, d)$, where $d = -3$ is the decryption key. Note the encryption algorithm is the same as the decryption algorithm and that the encryption and decryption keys are one the opposite of the other.

Modern Cryptosystems

Modern cryptosystems are much more complicated than the Caesar cipher, and much harder to break. For example, the ***Advanced Encryption Standard*** (***AES***) algorithm, uses keys that are 128, 196, or 256 bits in length, so that it is practically infeasible for an eavesdropper, Eve, to try all possible keys in a brute-force attempt to discover the corresponding plaintext from a given ciphertext. Likewise, the AES algorithm is much more convoluted than a simple cyclic shift of characters in the alphabet, so we are not going to review the details here (see Section 8.1.6).

Symmetric Encryption

One important property of the AES algorithm that we do note here, however, is that the same key K is used for both encryption and decryption. Such schemes as this, which use the same key for encryption and decryption, are called ***symmetric cryptosystems*** or ***shared-key cryptosystems***, since Alice and Bob have to both share the key K in order for them to communicate a confidential message, M. A symmetric cryptosystem is schematically illustrated in Figure 1.8.

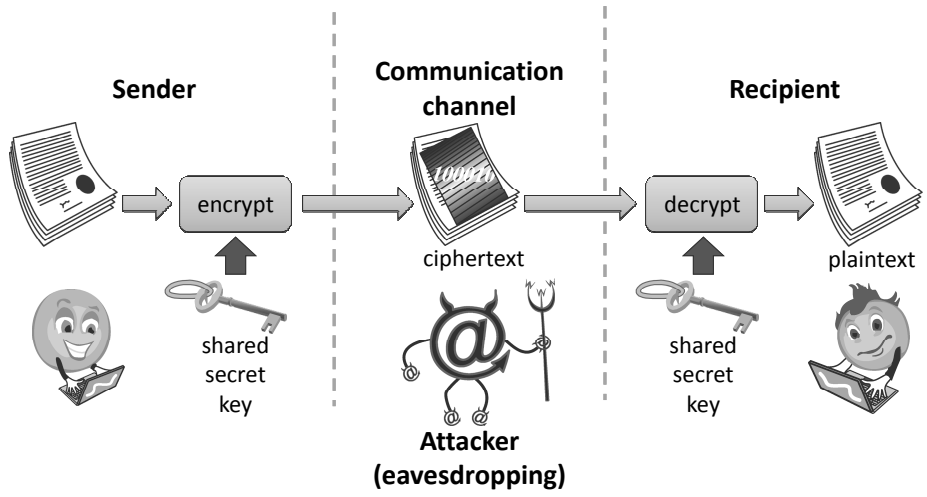


Figure 1.8: A symmetric cryptosystem, where the same secret key, shared by the sender and recipient, is used to encrypt and decrypt. An attacker who eavesdrops the communication channel cannot decrypt the ciphertext (encrypted message) without knowing the key.

Symmetric Key Distribution

Symmetric cryptosystems, including the AES algorithm, tend to run fast, but they require some way of getting the key K to both Alice and Bob without an eavesdropper, Eve, from discovering it. Also, suppose that n parties wish to exchange encrypted messages with each other in such a way that each message can be seen only by the sender and recipient. Using a symmetric cryptosystem, a distinct secret key is needed for each pair of parties, for a total of $n(n-1)/2$ keys, as illustrated in Figure 1.9.

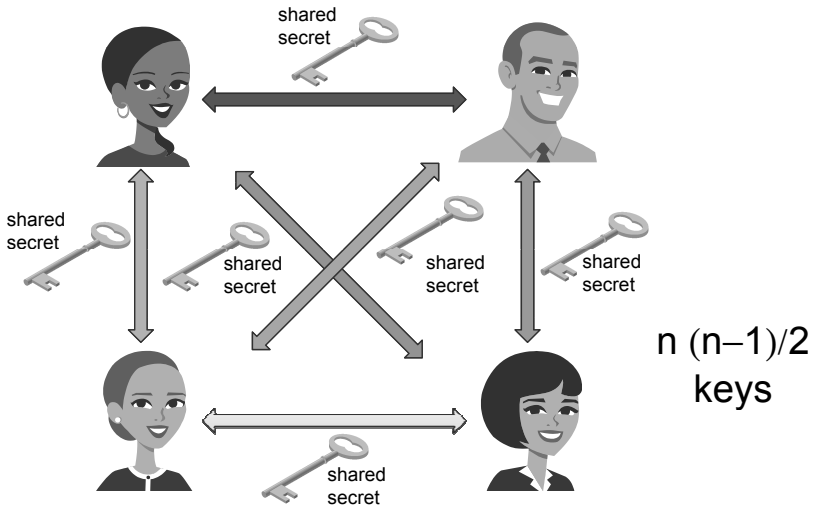


Figure 1.9: Pairwise confidential communication among n users with a symmetric cryptosystem requires $n(n-1)/2$ distinct keys, each shared by two users and kept secret from the other users.

Public-Key Encryption

An alternative approach to symmetric cryptosystems is the concept of a ***public-key cryptosystem***. In such a cryptosystem, Bob has two keys: a ***private key***, S_B, which Bob keeps secret, and a ***public key***, P_B, which Bob broadcasts widely, possibly even posting it on his web page. In order for Alice to send an encrypted message to Bob, she need only obtain his public key, P_B, use that to encrypt her message, M, and send the result, $C = E_{P_B}(M)$, to Bob. Bob then uses his secret key to decrypt the message as

$$M = D_{S_B}(C).$$

A public-key cryptosystem is schematically illustrated in Figure 1.10.

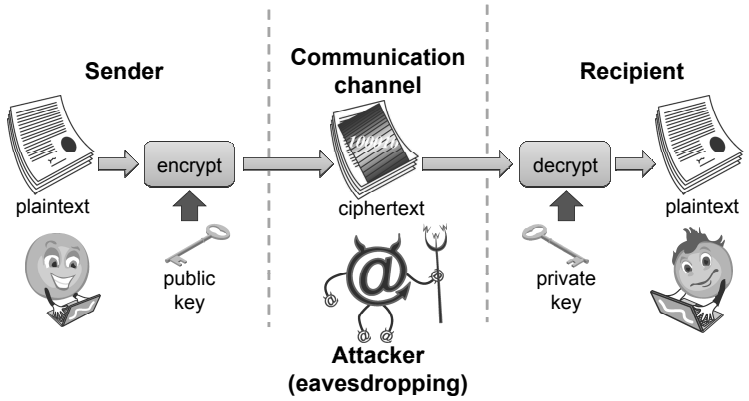


Figure 1.10: In a public-key cryptosystem, the sender uses the public key of the recipient to encrypt and the recipient uses its private key to decrypt. An attacker who eavesdrops the communication channel cannot decrypt the ciphertext (encrypted message) without knowing the private key.

The advantage of public-key cryptosystems is that they sidestep the problem of getting a single shared key to both Alice and Bob. Also, only private keys need to be kept secret, while public keys can be shared with anyone, including the attacker. Finally, public-key cryptosystems support efficient pairwise confidential communication among n users. Namely, only n distinct private/public key pairs are needed, as illustrated in Figure 1.11. This fact represents a significant improvement over the quadratic number of distinct keys required by a symmetric cryptosystem. For example, if we have $1,000$ users, a public-key cryptosystem uses $1,000$ private/public key pairs while a symmetric cryptosystem requires $499,500$ secret keys.

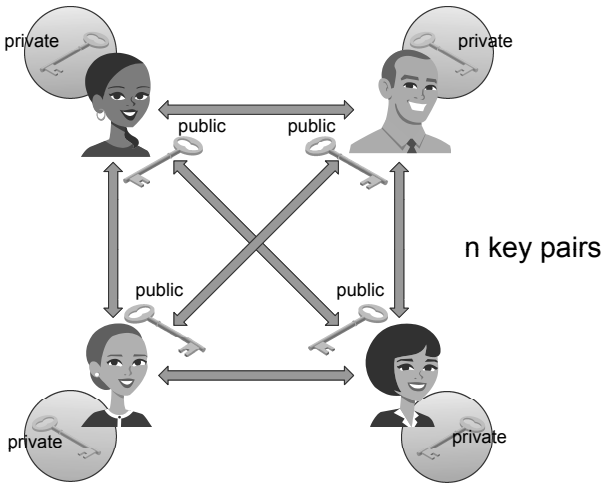


Figure 1.11: Pairwise confidential communication among n users with a public-key cryptosystem requires n key pairs, one per user.

Some Disadvantages of Public-Key Cryptography

The main disadvantage of public-key cryptosystems is that in all of the existing realizations, such as the RSA and ElGamal cryptosystems, the encryption and decryption algorithms are much slower than the those for existing symmetric encryption schemes. In fact, the difference in running time between existing public-key crytosystems and symmetric cryptosystems disourages people for using public-key cryptography for interactive sessions that use a lot of back-and-forth communication.

Also, public-key cryptosystems require in practice a key length that is one order of magnitude larger than that for symmetric cryptosystems. For example, RSA is commonly used with 2,048-bit keys while AES is typically used with 256-bit keys.

In order to work around these disadvantages, public-key cryptosystems are often used in practice just to allow Alice and Bob to exchange a shared secret key, which they subsequently use for communicating with a symmetric encryption scheme, as shown in Figure 1.12.

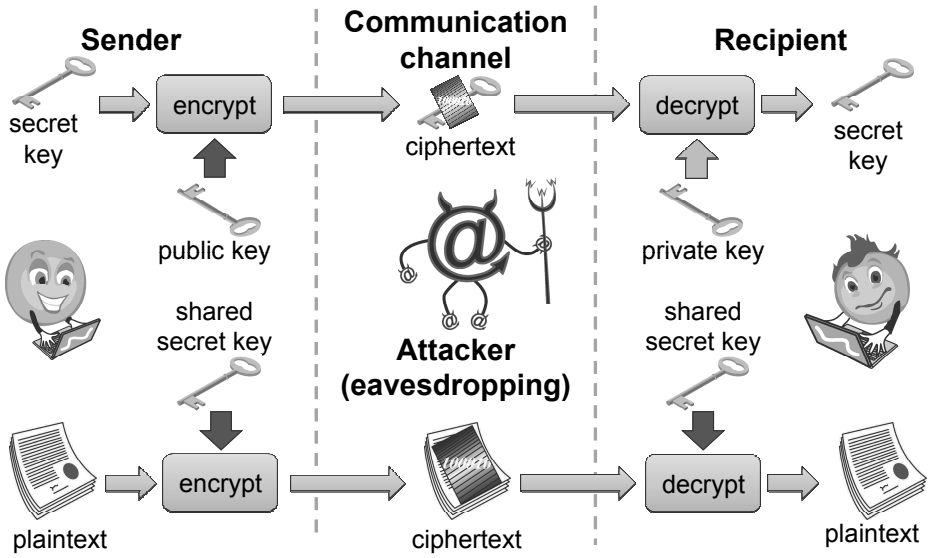


Figure 1.12: Use of a public-key cryptosystem to exchange a shared secret key, which is subsequently employed for communicating with a symmetric encryption scheme. The secret key is the "plaintext" message sent from the sender to the recipient.

1.3.2 Digital Signatures

Another problem that is solved by public-key cryptosystems is the construction of digital signatures. This solution is derived from the fact that in typical public-key encryption schemes, we can reverse the order in which the encryption and decryption algorithms are applied:

$$E_{P_B}(D_{S_B}(M)) = M.$$

That is, Bob can give as input to the decryption algorithm a message, M, and his private key, S_B. Applying the encryption algorithm to the resulting output and Bob's public key, which can be done by anyone, yields back message M.

Using a Private Key for a Digital Signature

This might at first seem futile, for Bob is creating an object that anyone can convert to message M, that is, anyone who knows his public key. But that is exactly the point of a digital signature—only Bob could have done such a decryption. No one else knows his secret key. So if Bob intends to prove that he is the author of message M, he computes his personal decryption of it as follows:

$$S = D_{S_B}(M).$$

This decryption S serves as a digital signature for message M. Bob sends signature S to Alice along with message M. Alice can recover M by encrypting signature S with Bob's public key:

$$M = E_{P_B}(S).$$

In this way, Alice is assured that message M is authored by Bob and not by any other user. Indeed, no one but Bob, who has private key S_B, could have produced such an object S, so that $E_{P_B}(S) = M$.

The only disadvantage of this approach is that Bob's signature will be at least as long as the plaintext message he is signing, so this exact approach is not used in practice. We study digital signatures in more detail in Section 8.4.

1.3.3 Simple Attacks on Cryptosystems

Consider a cryptosystem for n-bit plaintexts. In order to guarantee unique decryption, ciphertexts should have at least n bits or otherwise two or more plaintexts would map to the same ciphertext. In cryptosystems used in practice, plaintexts and ciphertexts have the same length. Thus, for a given symmetric key (or private-public key pair), the encryption and decryption algorithms define a matching among n-bit strings. That is, each plaintext corresponds to a unique ciphertext, and vice versa.

Man-in-the-Middle Attacks

The straightforward use of a cryptosystem presented in Section 1.3.1, which consists of simply transmitting the ciphertext, assures confidentiality. However, it does not guarantee the authenticity and integrity of the message if the adversary can intercept and modify the ciphertext. Suppose that Alice sends to Bob ciphertext C corresponding to a message M. The adversary modifies C into an altered ciphertext C' received by Bob. When Bob decrypts C', he obtains a message M' that is different from M. Thus, Bob is led to believe that Alice sent him message M' instead of M. This man-in-the-middle attack is illustrated in Figure 1.13.

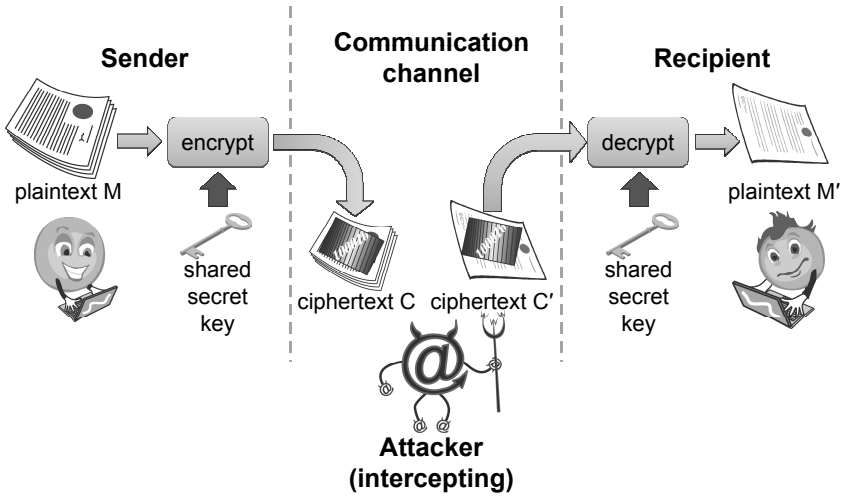


Figure 1.13: A man-in-the-middle attack where the adversary modifies the ciphertext and the recipient decrypts the altered ciphertext into an incorrect message.

Similarly, consider the straightforward use of digital signatures presented in Section 1.3.2. The attacker can modify the signature S created by Bob into a different string S' and send to Alice signature S' together with the encryption M' of S' using Bob's public key. Note that M' will be different from the original message M. When Alice verifies the digital signature S', she obtains message M' by encrypting S'. Thus, Alice is led to believe that Bob has signed M' instead of M.

Note that in the above attacks the adversary can arbitrarily alter the transmitted ciphertext or signature. However, the adversary cannot choose, or even figure out, what would be the resulting plaintext since he does not have the ability to decrypt. Thus, the above attacks are effective only if any arbitrary sequence of bits is a possible message. This scenario occurs, for example, when a randomly generated symmetric key is transmitted encrypted with a public-key cryptosystem.

Brute-Force Decryption Attack

Now, suppose instead that valid messages are English text of up to t characters. With the standard 8-bit ASCII encoding, a message is a binary string of length $n = 8t$. However, valid messages constitute a very small subset of all the possible n-bit strings, as illustrated in Figure 1.14.

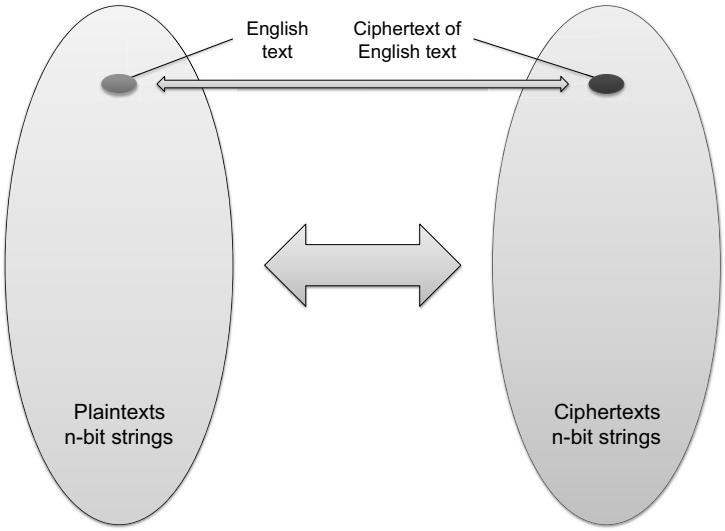


Figure 1.14: Natural-language plaintexts are a very small fraction of the set of possible plaintexts. This fraction tends to zero as the plaintext length grows. Thus, for a given key, it is hard for an adversary to guess a ciphertext that corresponds to a valid message.

Assume that we represent characters with the standard 8-bit ASCII encoding and let $n = 8$ the number of bits in a t-byte array. We have that the total number of possible t-byte arrays is $(2^8)^t = 2^n$. However, it is estimated that each character of English text carries about 1.25 bits of information, i.e., the number of t-byte arrays that correspond to English text is

$$\left(2^{1.25}\right)^t = 2^{1.25t}.$$

So, in terms of the bit length n, the number of n-bit arrays corresponding to English text is approximately $2^{0.16n}$.

More generally, for a ***natural language*** that uses an alphabet instead of ideograms, there is a constant α, with $0 < \alpha < 1$, such that there are $2^{\alpha n}$ texts among all n-bit arrays. The constant α depends on the specific language and character-encoding scheme used. As a consequence, in a natural language the fraction of valid messages out of all possible n-bit plaintexts is about

$$\frac{2^{\alpha n}}{2^n} = \frac{1}{2^{(1-\alpha)n}}.$$

Thus, the fraction of valid messages tends rapidly to zero as n grows. Note that this fraction represents the probability that a randomly selected plaintext corresponds to meaningful text.

The above property of natural languages implies that it is infeasible for an adversary to guess a ciphertext that will decrypt to a valid message or to guess a signature that will encrypt to a valid message.

The previously mentioned property of natural languages has also important implications for ***brute-force decryption*** attacks, where an adversary tries all possible decryption keys and aims at determining which of the resulting plaintexts is the correct one. Clearly, if the plaintext is an arbitrary binary string, this attack cannot succeed, as there is no way for the attacker to distinguish a valid message. However, if the plaintext is known to be text in a natural language, then the adversary hopes that only a small subset of the decryption results (ideally just a single plaintext) will be a meaningful text for the language. Some knowledge about the possible message being sent will then help the attacker pinpoint the correct plaintext.

We know that for some constant $\alpha > 1$, there are $2^{\alpha n}$ valid text messages among the 2^n possible plaintexts. Let k be the length (number of bits) of the decryption key. For a given ciphertext, there are 2^k possible plaintexts, each corresponding to a key. From the previous discussion, each such plaintext is a valid text message with probability $\frac{1}{2^{(1-\alpha)n}}$. Hence, the expected number of plaintexts corresponding to valid text messages is

$$\frac{2^k}{2^{(1-\alpha)n}}.$$

As the key length k is fixed, the above number tends rapidly to zero as the ciphertext length n grows. Also, we expect that there is a unique valid plaintext for the given ciphertext when

$$n = \frac{k}{1-\alpha}.$$

The above threshold value for n is called the ***unicity distance*** for the given language and key length. For the English language and the 256-bit AES cryptosystem, the unicity distance is about 304 bits or 38 ASCII-encoded characters. This is only half a line of text.

From the above discussion, we conclude that brute-force decryption is likely to succeed for messages in natural language that are not too short. Namely, when a key yields a plaintext that is a meaningful text, the attacker has probably recovered the original message.

1.3.4 Cryptographic Hash Functions

To reduce the size of the message that Bob has to sign, we often use cryptographic ***hash functions***, which are checksums on messages that have some additional useful properties. One of the most important of these additional properties is that the function be ***one-way***, which means that it is easy to compute but hard to invert. That is, given M, it should be relatively easy to compute the hash value, $h(M)$. But given only a value y, it should be difficult to compute a message M such that $y = h(M)$. Modern cryptographic hash functions, such as SHA-256, are believed to be one-way functions, and result in values that are only 256 bits long.

Applications to Digital Signatures and File System Integrity

Given a cryptographic hash function, we can reduce the time and space needed for Bob to perform a digital signature by first having him hash the message M to produce $h(M)$ and then have him sign this value, which is sometimes called the ***digest*** of M. That is, Bob compute the following signature:

$$S = E_{S_B}(h(M)).$$

Now to verify signature S on a message M, Alice computes $h(M)$, which is easy, and then checks that

$$D_{P_B}(S) = h(M).$$

Signing a cryptographic digest of the message not only is more efficient than signing the message itself, but also defends against the man-in-the-middle attack described in Section 1.3.3. Namely, thanks to the one-way

property of the cryptographic hash function h, it is no longer possible for the attacker to forge a message-signature pair without knowledge of the private key. The encryption of the forged signature S' now yields a digest y' for which the attacker needs to find a corresponding message M' such that $y' = h(M')$. This computation is unfeasible because h is one-way.

In addition, cryptographic hash functions also have another property that is useful in the context of digital signatures—they are ***collision resistant***—which implies that, given M, it is difficult to find a different message, M', such that $h(M) = h(M')$. This property makes the forger's job even more difficult, for not only it is hard for him to fake Bob's signature on any message, it is also hard for him, given a message M and its signature S created by Bob, to find another message, M', such that S is also a signature for M'.

Another application of cryptographic hash functions in secure computer systems is that they can be used to protect the integrity of critical files in an operating system. If we store the cryptographic hash value of each such file in protected memory, we can check the authenticity of any such file just by computing its cryptographic hash and comparing that value with the one stored in secure memory. Since such hash functions are collision resistant, we can be confident that if the two values match it is highly likely that the file has not been tampered with. In general, hash functions have applications any time we need a compact digest of information that is hard to forge.

Message Authentication Codes

A cryptographic hash function h can be used in conjunction with a secret key shared by two parties to provide integrity protection to messages exchanged over an insecure channel, as illustrated in Figure 1.15. Suppose Alice and Bob share a secret key K. When Alice wants to send a message M to Bob, she computes the hash value of the key K concatenated with message M:

$$A = h(K||M).$$

This value A is called a ***message authentication code*** (***MAC***). Alice then sends the pair (M, A) to Bob. Since the communication channel is insecure, we denote with (M', A') the pair received by Bob. Since Bob knows the secret K, he computes the authentication code for the received message M himself:

$$A'' = h(K||M').$$

If this computed MAC A'' is equal to the received MAC A', then Bob is assured that M' is the message sent by Alice, i.e., $A'' = A'$ implies $M' = M$.

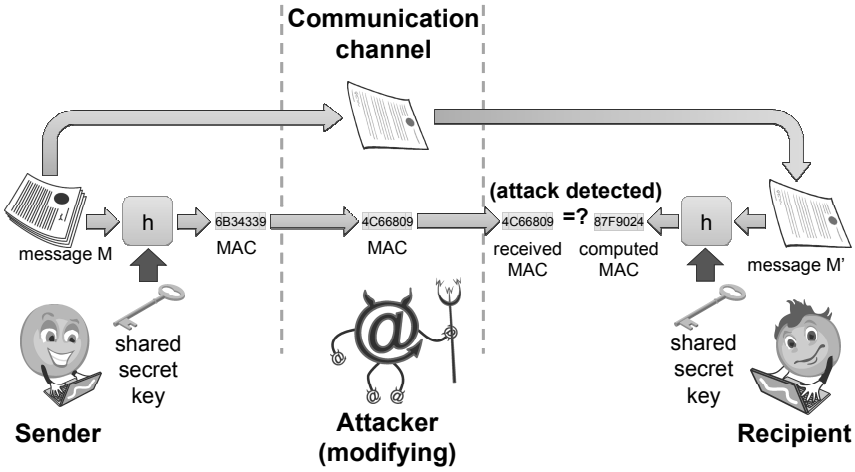


Figure 1.15: Using a message authentication code to verify the integrity of a message.

Consider an attacker who alters the message and MAC while in transit. Since the hash function is one-way, it is infeasible for the attacker to recover the key k from the MAC $A = h(K||M)$ and the message M sent by Alice. Thus, the attacker cannot modify the message and compute a correct MAC A' for the modified message M'.

1.3.5 Digital Certificates

As illustrated in Figure 1.12, public-key cryptography solves the problem of how to get Alice and Bob to share a common secret key. That is, Alice can simply encrypt secret key K using Bob's public key, P_B, and send the ciphertext to him. But this solution has a flaw: How does Alice know that the public key, P_B, that she used is really the public key for Bob? And if there are lots of Bobs, how can she be sure she used the public key for the right one?

Fortunately, there is a fix to this flaw. If there is a trusted authority who is good at determining the true identities of people, then that authority can digitally sign a statement that combines each person's identity with their public key. That is, this trusted authority could sign a statement like the following:

> "The Bob who lives on 11 Main Street in Gotham City was born on August 4, 1981, and has email address bob@gotham.com, has the public key P_B, and I stand by this certification until December 31, 2011."

Such a statement is called a ***digital certificate*** so long as it combines a public key with identifying information about the subject who has that public key. The trusted authority who issues such a certificate is called a ***certificate authority*** (***CA***).

Now, rather than simply trusting on blind faith that P_B is the public key for the Bob she wants to communicate with, Alice needs only to trust the certificate authority. In addition, Alice needs to know the public key for the CA, since she will use that to verify the CA's signature on the digital certificate for Bob. But there are likely to be only a small number of CAs, so knowing all their public keys is a reasonable assumption. In practice, the public keys of commonly accepted CAs come with the operating system. Since the digital certificate is strong evidence of the authenticity of Bob's public key, Alice can trust it even if it comes from an unsigned email message or is posted on a third-party web site.

For example, the digital certificate for a web site typically includes the following information:

- Name of the certification authority (e.g., Thawte).
- Date of issuance of the certificate (e.g., 1/1/2009).
- Expiration date of the certificate (e.g., 12/31/2011).
- Address of the website (e.g., mail.google.com).
- Name of the organization operating the web site (e.g., "Google, Inc.").
- Public key used of the web server (e.g., an RSA 1,024-bit key).
- Name of the cryptographic hash function used (e.g., SHA-256).
- Digital signature.

In fact, when an Internet browser "locks the lock" at a secure web site, it is doing so based on a key exchange that starts with the browser downloading the digital certificate for this web server, matching its name to a public key. Thus, one approach to defend against a phishing attack for encrypted web sites is to check that the digital certificate contains the name of the organization associated with the website.

There are a number of other cryptographic concepts, including such things a zero-knowledge proofs, secret sharing schemes, and broadcast encryption methods, but the topics covered above are the most common cryptographic concepts used in computer security applications.

1.4 Implementation and Usability Issues

In order for computer security solutions to be effective, they have to be implemented correctly and used correctly. Thus, when computer security solutions are being developed, designers should keep both the programmers and users in mind.

1.4.1 Efficiency and Usability

Computer security solutions should be efficient, since users don't like systems that are slow. This rule is the prime justification, for example, for why a public-key cryptosystem is often used for a one-time exchange of a secret key that is then used for communication with a symmetric encryption scheme.

An Example Scenario Involving Usability and Access Control

Efficiency and ease of use are also important in the context of access control. Many systems allow only administrators to make changes to the files that define access control rights, roles, or entities. So, for example, it is not possible in some operating systems, including several Linux versions, for users to define the access control rights for their own files beyond coarse-grained categories such as "everyone" and "people in my group." Because of this limitation, it is actually a cumbersome task to define a new work group and give access rights to that group. So, rather than going through the trouble of asking an administrator to create a new group, a user may just give full access rights to everyone, thus compromising data confidentiality and integrity.

For example, suppose a group of students decides to work on a software project together for a big schoolwide contest. They are probably going to elect a project leader and have her create a subdirectory of her home directory for all the project code to reside. Ideally, it should be easy for the leader to define the access control for this directory and allow her partners to have access to it, but no one else. Such control is often not possible without submitting a request to an overworked system administrator, who may or may not respond to such requests from students. So, what should the project leader do?

Possible Solutions

One solution is to have the leader maintain the reference version of the code in the project directory and require the team members to email her all their code updates. On receipt of an update from a team member, the leader would then perform the code revisions herself on the reference version of the code and would distribute the modified files to the rest of the team. This solution provides a reasonable level of security, as it is difficult (though not impossible) to intercept email messages. However, the solution is very inefficient, as it implies a lot of work for the leader who would probably regret being selected for this role.

Another possibility is for the project leader to take an easy way out by hiding the project directory somewhere deep in her home directory, making that directory be accessible by all the users in the system, and hoping that none of the competing teams will discover this unprotected directory. This approach is, in fact, an example of ***security by obscurity***, which is the approach of deriving security from a fact that is not generally known rather than employing sound computer security principles (as discussed in Sections 1.1.1 and 1.1.2). History has taught us again and again, however, that security by obscurity fails miserably. Thus, the leader of our software team is forced into choosing between the lesser of two evils, rather than being given the tools to build a secure solution to her problem.

Users should clearly not have to make such choices between security and efficiency, of course. But this requirement implies that system designers need to anticipate how their security decisions will impact users. If doing the safe thing is too hard, users are going to find a workaround that is easy but probably not very secure.

Let us now revisit our example of the school programming team. The most recent versions of Linux and Microsoft Windows allow the owner of a folder to directly define an access control list for it (see Section 1.2.2), without administrator intervention. Also, by default such permissions are automatically applied to all the files and subfolders created within the folder. Thus, our project leader could simply add an access control list to the project folder that specifies read, write, and execute rights for each of the team members. Team members can now securely share the project folder without the risk of snooping by competing teams. Also, the project leader needs to create this access control list only once, for the project folder. Any newly added files and subfolders will automatically inherit this access control list. This solution is both efficient and easy to use. More details on advanced file permissions are given in Section 3.3.3.

1.4.2 Passwords

One of the most common means for authenticating people in computer systems is through the use of usernames and passwords. Even systems based on cryptographic keys, physical tokens, and biometrics often augment the security of these techniques with passwords. For example, the secret key used in a symmetric cryptosystem may be stored on the hard drive in encrypted form, where the decryption key is derived from a password. In order for an application to use the secret key, the user will have to enter her password for the key. Thus, a critical and recurring issue in computer security circles is password security.

Ideally, passwords should be easy to remember and hard to guess. Unfortunately, these two goals are in conflict with each other. Passwords that are easy to remember are things like English words, pet names, birthdays, anniversaries, and last names. Passwords that are hard to guess are random sequences of characters that come from a large alphabet, such as all the possible characters that can be typed on a keyboard, including lowercase and uppercase letters, numbers, and symbols. In addition, the longer a password is used the more it is at risk. Thus, some system administrators require that users frequently change their passwords, which makes them even more difficult to remember.

Dictionary Attack

The problem with the typical easy-to-remember password is that it belongs to a small set of possibilities. Moreover, computer attackers know all these passwords and have built dictionaries of them. For example, for the English language, there are less than 50,000 common words, 1,000 common human first names, 1,000 typical pet names, and 10,000 common last names. In addition, there are only 36,525 birthdays and anniversaries for almost all living humans on the planet, that is, everyone who is 100 years old or younger. So an attacker can compile a dictionary of all these common passwords and have a file that has fewer than 100,000 entries.

Armed with this dictionary of common passwords, one can perform an attack that is called, for obvious reasons, a ***dictionary attack***. If an attcker can try the words in his dictionary at the full speed of a modern computer, he can attack a password-protected object and break its protections in just a few minutes. Specifically, if a computer can test one password every millisecond, which is probably a gross overestimate for a standard computer with a clock speed of a gigahertz, then it can complete the dictionary attack in 100 seconds, which is less than 2 minutes. Indeed, because of this risk, many systems introduce a multiple-second delay before reporting

password failures and some systems lock out users after they have had a number of unsuccessful password attempts above some threshold.

Secure Passwords

Secure passwords, on the other hand, take advantage of the full potential of a large alphabet, thus slowing down dictionary attacks. For instance, if a system administrator insists on each password being an arbitrary string of at least eight printable characters that can by typed on a typical American keyboard, then the number of potential passwords is at least $94^8 = 6\,095\,689\,385\,410\,816$, that is, at least 6 quadrillion. Even if a computer could test one password every nanosecond, which is about as fast as any computer could, then it would take, on average, at least 3 million seconds to break one such password, that is, at least 1 month of nonstop attempts.

The above back-of-the-envelope calculation could be the reason why paranoid system administrators ask users to change their passwords every month. If each attempt takes at least a microsecond, which is more realistic, then breaking such a password would take at least 95 years on average. So, realistically, if someone can memorize a complex password, and never leak it to any untrustworthy source, then it is probably good for a long time.

There are several tricks for memorizing a complex password. Needless to say in a book on computer security, one of those ways is definitely not writing the password down on a post-it note and sticking it on a computer screen! A better way is to memorize a silly or memorable sentence and then take every first letter of each word, capitalizing some, and then folding in some special characters. For example, a user, who we will call "Mark," could start with the sentence

> "Mark took Lisa to Disneyland on March 15,"

which might be how Mark celebrated his anniversary with Lisa. Then this sentence becomes the string

> MtLtDoM15,

which provides a pretty strong password. However, we can do even better. Since a t looks a lot like the plus sign, Mark can substitute "+" for one of the t's, resulting in the password

> MtL+DoM15,

which is even stronger. If Mark is careful not to let this out, this password could last a lifetime.

1.4.3 Social Engineering

The three B's of espionage—burglary, bribery, and blackmail—apply equally well to computer security. Add to these three techniques good old fashion trickery and we come up with one of the most powerful attacks against computer security solutions—***social engineering***. This term refers to techniques involving the use of human insiders to circumvent computer security solutions.

Pretexting

A classic example of a social engineering attack, for instance, involves an attacker, Eve, calling a helpdesk and telling them that she has forgotten her password, when she is actually calling about the account of someone else, say, someone named "Alice." The helpdesk agent might even ask Eve a few personal questions about Alice, which, if Eve has done her homework, she can answer with ease. Then the courteous helpdesk agent will likely reset the password for Alice's account and give the new password to Eve, thinking that she is Alice. Even counting in the few hours that it takes Eve to discover some personal details about Alice, such as her birthday, mother's maiden name, and her pet's name, such an attack works faster than a brute-force password attack by orders of magnitudes, and it doesn't require any specialized hardware or software. Such an attack, which is based on an invented story or pretext, is known as ***pretexting***.

Baiting

Another attack, known as ***baiting***, involves using some kind of "gift" as a bait to get someone to install malicious software. For example, an attacker could leave a few USB drives in the parking lot of a company with an otherwise secure computer system, even marking some with the names of popular software programs or games. The hope is that some unsuspecting employee will pick up a USB drive on his lunch break, bring it into the company, insert it into an otherwise secure computer, and unwittingly install the malicious software.

Quid Pro Quo

Yet another social engineering attack is the ***quid pro quo***, which is Latin for "something for something." For example, an attacker, "Bob," might call a victim, "Alice," on the phone saying that he is a helpdesk agent who was referred to Alice by a coworker. Bob then asks Alice if she has been

having any trouble with her computer or with her company's computer system in general. Or he could ask Alice if she needs any help in coming up with a strong password now that it is time to change her old one. In any case, Bob offers Alice some legitimate help. He may even diagnose and solve a problem she has been having with her computer. This is the "something" that Bob has now offered Alice, seemingly without asking for anything in return. At that point, Bob then asks Alice for her password, possibly offering to perform future fixes or offering to do an evaluation of how strong her password is. Because of the social pressure that is within each of us to want to return a favor, Alice may feel completely at ease at this point in sharing her password with Bob in return for his "free" help. If she does so, she will have just become a victim of the *quid pro quo* attack.

To increase the chances of succeeding in his attack, Bob may use a voice-over-IP (VoIP) telephone service that allows for caller-ID spoofing. Thus, he could supply as his caller-ID the phone number and name of the actual helpdesk for Alice's company, which will increase the likelihood that Alice will believe Bob's story. This is an instance of another type of attack called ***vishing***, which is short for VoIP phishing.

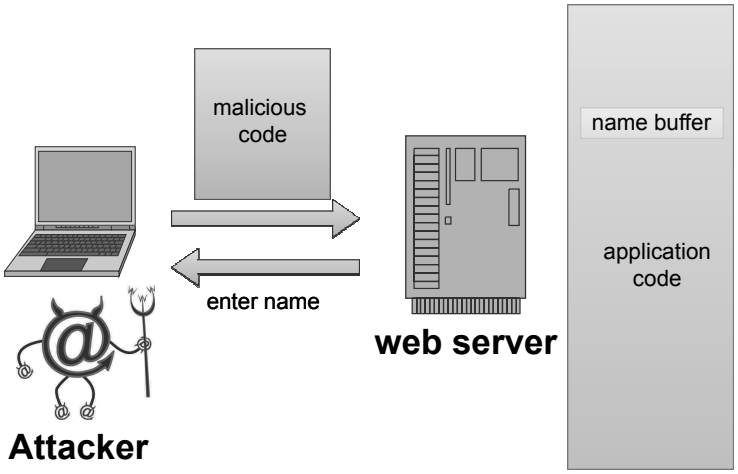
In general, social engineering attacks can be very effective methods to circumvent strong computer security solutions. Thus, whenever a system designer is implementing an otherwise secure system, he or she should keep in mind the way that people will interact with that system and the risks it may have to social engineering attacks.

1.4.4 Vulnerabilities from Programming Errors

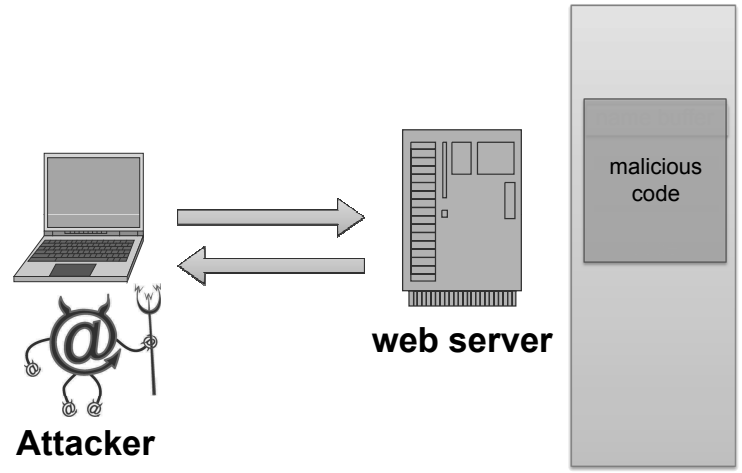
The programmers should be given clear instructions on how to produce the secure system and a formal description of the security requirements that need to be satisfied. Also, an implementation should be tested against all the security requirements. Special attention must be paid to sections of the program that handle network communication and process inputs provided by users. Indeed, any interaction of the program with the external world should be examined to guarantee that the system will remain in a secure state even if the external entity communicating with the system performs unexpected actions.

There are many examples of systems that enter into a vulnerable state when a user supplies a malformed input. For example, the classic ***buffer overflow*** attack (see Figure 1.16) injects code written by a malicious user into a running application by exploiting the common programming error of not checking whether an input string read by the application is larger than the variable into which it is stored (the buffer). Thus, a large input

provided by the attacker can overwrite the data and code of the application, which may result in the application performing malicious actions specified by the attacker. Web servers and other applications that communicate over the Internet have been often attacked by remote users by exploiting buffer overflow vulnerabilities in their code. For a more detailed description of how buffer overflow attacks work, see Section 3.4.3.



(a)



(b)

Figure 1.16: A buffer overflow attack on a web server. (a) A web server accepts user input from a name field on a web page into an unchecked buffer variable. The attacker supplies as input some malicious code. (b) The malicious code read by the server overflows the buffer and part of the application code. The web server now runs the malicious code.

1.5 Exercises

For help with exercises, please visit **securitybook.net**.

Reinforcement

R-1.1 Compare and contrast the C.I.A. concepts for information security with the A.A.A. concepts.

R-1.2 What is the ciphertext in an English version of the Caesar cipher for the plaintext "ALL ZEBRAS YELP."

R-1.3 Explain why someone need not worry about being a victim of a social engineering attack through their cell phone if they are inside of a Faraday cage.

R-1.4 What are some of the techniques that are used to achieve confidentiality?

R-1.5 What is the most efficient technique for achieving data integrity?

R-1.6 With respect to the C.I.A. and A.A.A. concepts, what risks are posed by spam?

R-1.7 With respect to the C.I.A. and A.A.A. concepts, what risks are posed by Trojan horses?

R-1.8 With respect to the C.I.A. and A.A.A. concepts, what risks are posed by computer viruses?

R-1.9 With respect to the C.I.A. and A.A.A. concepts, what risks are posed by packet sniffers, which monitor all the packets that are transmitted in a wireless Internet access point?

R-1.10 With respect to the C.I.A. and A.A.A. concepts, what risks are posed by someone burning songs from an online music store onto a CD, then ripping those songs into their MP3 player software system and making dozens of copies of these songs for their friends?

R-1.11 With respect to the C.I.A. and A.A.A. concepts, what risks are posed by someone making so many download requests from an online music store that it prevents other users from being able to download any songs?

R-1.12 Compare and contrast symmetric encryption with public-key encryption, including the strengths and weaknesses of each.

R-1.13 List at least three security risks that could arise when someone has their laptop stolen.

R-1.14 Suppose the author of an online banking software system has programmed in a secret feature so that program emails him the account information for any account whose balance has just gone over $10,000. What kind of attack is this and what are some of its risks?

R-1.15 Suppose an Internet service provider (ISP) has a voice over IP (VoIP) telephone system that it manages and sells. Suppose further that this ISP is deliberately dropping 25% of the packets used in its competitors VoIP system when those packets are going through this ISP's routers. What kind of an attack is this?

R-1.16 Give an example of the false sense of security that can come from using the "security by obscurity" approach.

R-1.17 The English language has an information content of about 1.25 bits per character. Thus, when using the standard 8-bit ASCII encoding, about 6.75 bits per character are redundant. Compute the probability that a random array of t bytes corresponds to English text.

R-1.18 Suppose that a symmetric cryptosystem with 32-bit key length is used to encrypt messages written in English and encoded in ASCII. Given that keys are short, an attacker is using a brute-force exhaustive search method to decrypt a ciphertext of t bytes. Estimate the probability of uniquely recovering the plaintext corresponding to the ciphertext for the following values of t: 8, 64, and 512.

R-1.19 Suppose you could use all 128 characters in the ASCII character set in a password. What is the number of 8-character passwords that could be constructed from such a character set? How long, on average, would it take an attacker to guess such a password if he could test a password every nanosecond?

R-1.20 Doug's laptop computer was just infected with some malicious software that uses his laptop's built-in camera to take a video each time it senses movement and then upload the video to a popular video-sharing web site. What type of attack does this involve and what concepts of computer security does it violate?

R-1.21 The Honyota Corporation has a new car out, the Nav750, which transmits its GPS coordinates to the Honyota Corporation computers every second. An owner can then locate their car any time, just by accessing this site using a password, which is a concatenation of their last name and favorite ice cream flavor. What are some security concerns for the Nav750? What are some privacy concerns, say, if the car's owner is the spouse, parent, or employer of the car's principle driver?

R-1.22 The HF Corporation has a new refrigerator, the Monitator, which has a camera that takes a picture of the contents of the refrigerator and uploads it to the HF Corporation's web site. The Monitator's owner can then access this web site to see what is inside their refrigerator without opening the door. For security reasons, the HF Corporation encrypts this picture using a proprietary algorithm and gives the 4-digit PIN to decrypt this picture to the Monitator's owner, so he or she can get access to the pictures of their Monitator's interior. What are the security concerns and principles that this solution does and doesn't support?

R-1.23 During the 2008 U.S. Presidential campaign, hackers were able to gain access to an email account of Vice Presidential candidate, Sarah Palin. Their attack is said to have involved tricking the mail system to reset Governor Palin's password, claiming they were really Palin and had forgotten this password. The system asked the hackers a number of personal questions regarding Palin's identity, including her birthday, zip code, and a personal security question—"Where did you meet your spouse?"—all of which the hackers were able to answer using data available on the Internet. What kind of attack is this an example of? Also, what degree of security is provided by a password reset feature such as this?

Creativity

C-1.1 Describe an architecture for an email password reset system that is more secure than the one described in Exercise R-1.23, but is still highly usable.

C-1.2 Describe an instance of a file that contains evidence of its own integrity and authenticity.

C-1.3 Suppose an Internet service provider (ISP) has a voice over IP (VoIP) telephone system that it manages and sells. Suppose further that this ISP is deliberately dropping 25% of the packets used in its competitors VoIP system when those packets are going through this ISP's routers. Describe how a user could discover that his ISP is doing this.

C-1.4 Computer viruses, by their very nature, have to be able to replicate themselves. Thus, a computer virus must store a copy of itself inside its own code. Describe how this property of computer viruses could be used to discover that a computer virus has infected a certain operating system file.

C-1.5 Suppose that you are a computer virus writer; hence, you know that you need to store a copy of the code for your virus inside the virus itself. Moreover, suppose you know that a security administrator is also aware of this fact and will be using it to detect the presence of your virus in operating systems files, as described in the previous problem. Explain how you can hide the embedded copy of your virus so that it is difficult for the security administrator to find it.

C-1.6 Describe a hybrid scheme for access control that combines both the access control list and capabilities models. Explain how the records for this hybrid model can be cross-linked to support object removal and subject removal in time proportional to their number of associated access rights; hence, not in time proportional to all the subject-object access right pairs.

C-1.7 Give two examples of attacks that compromise the integrity of the meta data of files or directories in a file system.

C-1.8 A ***rootkit*** is a piece of malicious software that installs itself into an operating system and then alters all of the operating system utility programs that would normally be able to detect this software so that they do not show its presence. Describe the risks that would be posed by such software, how it could actually be discovered, and how such an infection could be repaired.

C-1.9 Benny is a thief who tried to break into an Automated Teller Machine (ATM) using a screwdriver, but was only able to break five different keys on the numeric keypad and jam the card reader, at which point he heard Alice coming, so he hid. Alice walked up, put in her ATM card, successfully entered her 4-digit PIN, and took some cash. But she was not able to get her card back, so she drove off to find help. Benny then went back to the ATM, and started entering numbers to try to discover Alice's PIN and steal money from her account. What is the worst-case number of PINs that Benny has to enter before correctly discovering Alice's PIN?

C-1.10 As soon as Barack took office, he decided to embrace modern technology by communicating with cabinet members over the Internet using a device that supports cryptographic protocols. In a first attempt, Barack exchanges with Tim brief text messages, encrypted with public-key cryptography, to decide the exact amounts of bailout money to give to the largest 10 banks in the country. Let p_B and p_T be the public keys of Barack and Tim, respectively. A message m sent by Barack to Tim is transmitted as $E_{p_T}(m)$ and the reply r from Tim to Barack is transmitted as $E_{p_B}(r)$. The attacker

can eavesdrop the communication and knows the following information:

- Public keys p_B and p_T and the encryption algorithm
- The total amount of bailout money authorized by congress is \$900B
- The names of the largest 10 banks
- The amount each bank will get is a multiple of \$1B
- Messages and replies are terse exchanges of the following form:

  Barack: How much to Citibank?
  Tim: \$144B.
  Barack: How much to Bank of America?
  Tim: \$201B.
  . . .

Describe how the attacker can learn the bailout amount for each bank even if he cannot derive the private keys.

C-1.11 As a result of the above attack, Barack decides to modify the protocol of Exercise C-1.10 for exchanging messages. Describe two simple modifications of the protocol that are not subject to the above attack. The first one should use random numbers and the second one should use symmetric encryption.

C-1.12 Barack often sends funny jokes to Hillary. He does not care about confidentiality of these messages but wants to get credit for the jokes and prevent Bill from claiming authorship of or modifying them. How can this be achieved using public-key cryptography?

C-1.13 As public-key cryptography is computationally intensive and drains the battery of Barack's device, he comes up with an alternative approach. First, he shares a secret key k with Hillary but not with Bill. Next, together with a joke x, he sends over the value $d = h(k||x)$, where h is a cryptographic hash function. Does value d provide assurance to Hillary that Barack is the author of x and that x was not modified by Bill? Justify your answer.

C-1.14 Barack periodically comes up with brilliant ideas to stop the financial crisis, provide health care to every citizen, and save the polar bears. He wants to share these ideas with all the cabinet members but also get credit for the ideas. Extending the above approach, he shares a secret key k with all the cabinet members. Next, he broadcasts each idea z followed by value $h(k||z)$. Does this approach work or can Tim claim that he came up with the ideas instead of Barack? Justify your answer.

C-1.15 Describe a method that allows a client to authenticate multiple times to a server with the following requirements:

a. The client and server use constant space for authentication.
b. Every time the client authenticates to the server, a different random value for authentication is used (for example, if you have n authentication rounds, the client and the server have to use n different random values—this means that sharing a key initially and using it for every round of authentication is not a valid solution).

Can you find any vulnerabilities for this protocol?

C-1.16 Consider the following method that establishes a secret session key k for use by Alice and Bob. Alice and Bob already share a secret key K_{AB} for a symmetric cryptosystem.

a. Alice sends a random value N_A to Bob along with her id, A.
b. Bob sends encrypted message $E_{K_{AB}}(N_A), N_B$ to Alice, where N_B is a random value chosen by Bob.
c. Alice sends back $E_{K_{AB}}(N_B)$.
d. Bob generates session key k and sends $E_{K_{AB}}(k)$ to Alice.
e. Now Alice and Bob exchange messages encrypted with the new session key k.

Suppose that the random values and the keys have the same number of bits. Describe a possible attack for this authentication method.

Can we make the method more secure by lifting the assumption that the random values and the keys have the same number of bits? Explain.

C-1.17 Alice and Bob shared an n-bit secret key some time ago. Now they are no longer sure they still have the same key. Thus, they use the following method to communicate with each other over an insecure channel to verify that the key K_A held by Alice is the same as the key K_B held by Bob. Their goal is to prevent an attacker from learning the secret key.

a. Alice generates a random n-bit value R.
b. Alice computes $X = K_A \oplus R$, where $\oplus$ denotes the exclusive-or boolean function, and sends X to Bob.
c. Bob computes $Y = K_B \oplus X$ and sends Y to Alice.
d. Alice compares X and Y. If $X = Y$, she concludes that $K_A = K_B$, that is, she and Bob have indeed the same secret key.

Show how an attacker eavesdropping the channel can gain possession of the shared secret key.

C-1.18 Many Internet browsers "lock the lock" on an encrypted web site so long as the digital certificate offered for this site matches the name for this web server. Explain how this could lead to a false sense of security in the case of a phishing attack.

C-1.19 Explain the risks to Bob if he is willing to sign a seemingly random string using his private key.

C-1.20 Describe a good solution to the problem of having a group of students collaborate on a software construction project using the directory of one of the group members in such a way that it would be difficult for nonmembers to discover and would not require the help from a system administrator, assuming that the only access rights the group leader can modify are those for "everyone." You may assume that access rights for directories are "read," "write," and "exec," where "read" means the files and subdirectories in that directory can be listed, "write" means members of that directory can be inserted, deleted, or renamed, and "exec" on a directory or subdirectory means the user can change his location to that directory or subdirectory so long as he specifies its exact name.

C-1.21 Suppose an operating system gives users an automatic second chance functionality, so that any time a user asks to delete a file it actually gets put into a special "recycle bin" directory, which is shared by all users, with its access rights defined so that users can get their files back even if they forget their names. Describe the security risks that such a functionality poses.

C-1.22 Suppose, in a scenario based on a true story, a network computer virus is designed so as soon as it is copied onto a computer, X, it simply copies itself to six of X's neighboring computers, each time using a random file name, so as to evade detection. The virus itself does no other harm, in that it doesn't read any other files and it doesn't delete or modify any other files either. What harm would be done by such a virus and how would it be detected?

Projects

P-1.1 Implement a "toy" file system, with about a dozen different users and at least that many directories and files, that uses an access control matrix to manage access control rights.

P-1.2 Perform the project of Problem P-1.1, but use access control lists.

P-1.3 Perform the project of Problem P-1.1, but use capabilities to define the access control rights of each user.

P-1.4 Perform a statistical analysis of all the spam you get in a week, classifying each as to the types of attacks they are.

P-1.5 Implement a toy symmetric cryptosystem based on the following method.

a. Keys are 16-bit values.
b. Messages are strings with an even number of characters. One can always add a blank at the end of an odd-length string.
c. The encryption of a message M of length n (in bytes) is given by
$$E_K(M) = M \oplus (K||K|| \cdots),$$
where the key K is repeated $n/2$ times.
d. The decryption algorithm for a ciphertext C is the same as the encryption algorithm:
$$D_K(C) = C \oplus (K||K|| \cdots).$$

Implement a brute-force decryption attack for this cryptosystem and test it on randomly generated English text messages. Automate the process of detecting whether a decrypted message is English test.

Chapter Notes

The ten principles of computer security are from the seminal paper by Saltzer and Schroeder [86], who caution that the last two principles (work factor and compromise recording) are derived from physical security systems and "apply only imperfectly to computer systems." The open design principle was first formulated in a paper by 19th-century French cryptographer Auguste Kerckhoffs [47]. Bruce Schneier's Crypto-Gram Newsletter has a well-written article on secrecy, security, and obscurity [88]. A contemporary introduction to cryptography and its use in computer systems is given in the book by Ferguson, Schneier and Konho [30]. The redundancy of natural language was first formally studied by Claude Shannon in his pioneering paper defining the information-theoretic concept of entropy [91]. Usability issues for email encryption are the subject of an experimental study by Whitten and Tygar [107].