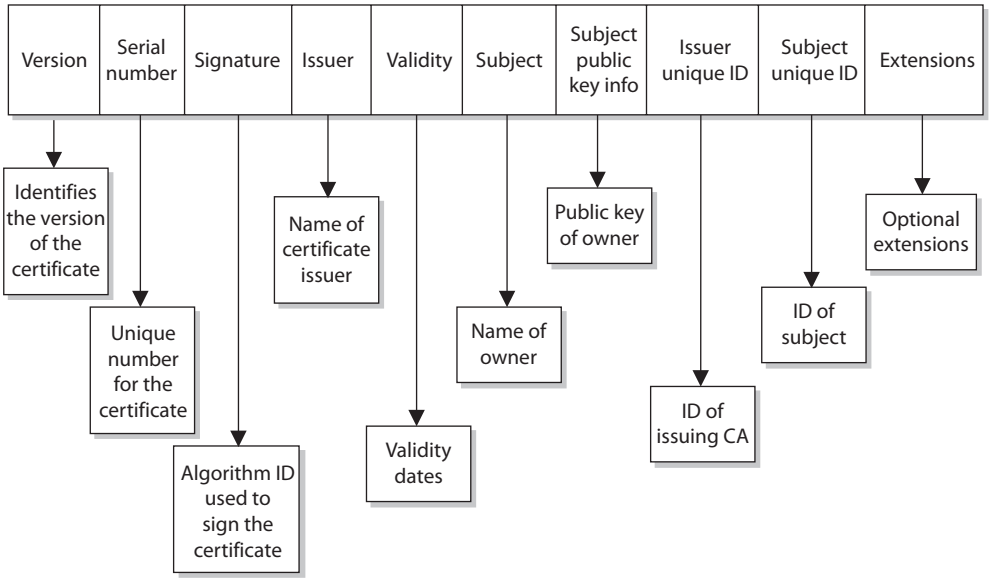


Figure 8-19 Each certificate has a structure with all the necessary identifying information in it.

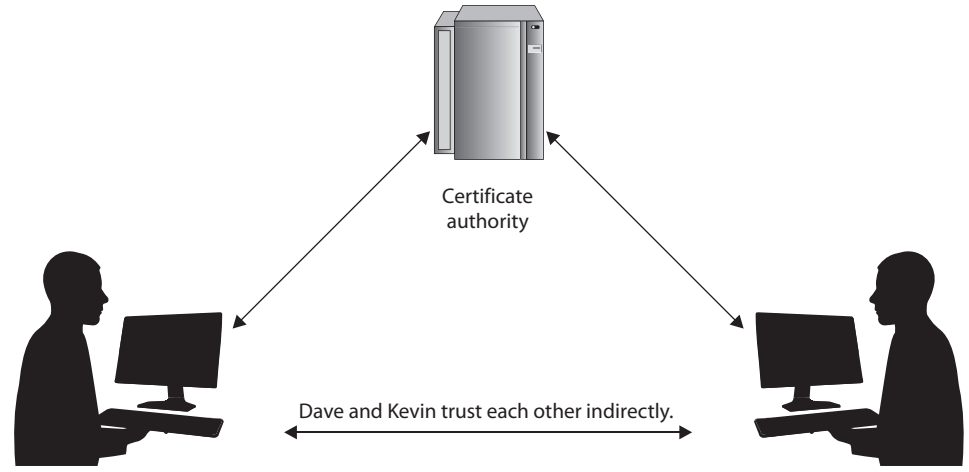
Certificate Authorities

A *certificate authority (CA)* is a trusted third party that vouches for the identity of a subject, issues a certificate to that subject, and then digitally signs the certificate to assure its integrity. When the CA signs the certificate, it binds the subject's identity to the public key, and the CA takes liability for the authenticity of that subject. It is this trusted third party (the CA) that allows people who have never met to authenticate to each other and to communicate in a secure method. If Kevin has never met Dave but would like to communicate securely with him, and they both trust the same CA, then Kevin could retrieve Dave's digital certificate and start the process.

A CA is a trusted organization (or server) that maintains and issues digital certificates. When a person requests a certificate, a *registration authority (RA)* verifies that individual's identity and passes the certificate request off to the CA. The CA constructs the certificate, signs it, sends it to the requester, and maintains the certificate over its lifetime. When another person wants to communicate with this person, the CA basically vouches for that person's identity. When Dave receives a digital certificate from Kevin, Dave goes through steps to validate it. Basically, by providing Dave with his digital certificate, Kevin is stating, "I know you don't know or trust me, but here is this document that was created by someone you do know and trust. The document says I am legitimate and you should trust I am who I claim to be."

Once Dave validates the digital certificate, he extracts Kevin's public key, which is embedded within it. Now Dave knows this public key is bound to Kevin. He also knows

that if Kevin uses his private key to create a digital signature and Dave can properly decrypt it using this public key, it did indeed come from Kevin.



The CA can be internal to an organization. Such a setup would enable the organization to control the CA server, configure how authentication takes place, maintain the certificates, and recall certificates when necessary. Other CAs are organizations dedicated to this type of service, and other individuals and companies pay them to supply it. Some well-known CAs are Symantec and GeoTrust. All browsers have several well-known CAs configured by default. Most are configured to trust dozens or hundreds of CAs.

NOTE More and more organizations are setting up their own internal PKIs. When these independent PKIs need to interconnect to allow for secure communication to take place (either between departments or between different companies), there must be a way for the two root CAs to trust each other. The two CAs do not have a CA above them they can both trust, so they must carry out cross-certification. *Cross-certification* is the process undertaken by CAs to establish a trust relationship in which they rely upon each other's digital certificates and public keys as if they had issued them themselves. When this is set up, a CA for one company can validate digital certificates from the other company and vice versa.

The CA is responsible for creating and handing out certificates, maintaining them, and revoking them if necessary. Revocation is handled by the CA, and the revoked certificate information is stored on a *certificate revocation list (CRL)*. This is a list of every certificate that has been revoked. This list is maintained and periodically updated by the issuing CA. A certificate may be revoked because the key holder's private key was compromised or because the CA discovered the certificate was issued to the wrong person. An analogy

for the use of a CRL is how a driver's license is used by a police officer. If an officer pulls over Sean for speeding, the officer will ask to see Sean's license. The officer will then run a check on the license to find out if Sean is wanted for any other infractions of the law and to verify the license has not expired. The same thing happens when a person compares a certificate to a CRL. If the certificate became invalid for some reason, the CRL is the mechanism for the CA to let others know this information.

NOTE CRLs are the thorn in the side of many PKI implementations. They are challenging for a long list of reasons. By default, web browsers do not check a CRL to ensure that a certificate is not revoked. So when you are setting up a secure connection to an e-commerce site, you could be relying on a certificate that has actually been revoked. Not good.

The *Online Certificate Status Protocol (OCSP)* is being used more and more rather than the cumbersome CRL approach. When using just a CRL, either the user's browser must check a central CRL to find out if the certification has been revoked, or the CA has to continually push out CRL values to the clients to ensure they have an updated CRL. If OCSP is implemented, it does this work automatically in the background. It carries out real-time validation of a certificate and reports back to the user whether the certificate is valid, invalid, or unknown. OCSP checks the CRL that is maintained by the CA. So the CRL is still being used, but now we have a protocol developed specifically to check the CRL during a certificate validation process.

Registration Authorities

The previously introduced *registration authority (RA)* performs the certification registration duties. The RA establishes and confirms the identity of an individual, initiates the certification process with a CA on behalf of an end user, and performs certificate lifecycle management functions. The RA cannot issue certificates, but can act as a broker between the user and the CA. When users need new certificates, they make requests to the RA, and the RA verifies all necessary identification information before allowing a request to go to the CA. In many cases, the role of CA and RA are fulfilled by different teams in the same organization.

PKI Steps

Now that you know some of the main pieces of a PKI and how they actually work together, let's walk through an example. First, suppose that John needs to obtain a digital certificate for himself so he can participate in a PKI. The following are the steps to do so:

1. John makes a request to the RA.
2. The RA requests certain identification information from John, such as a copy of his driver's license, his phone number, his address, and other identifying information.
3. Once the RA receives the required information from John and verifies it, the RA sends his certificate request to the CA.

4. The CA creates a certificate with John's public key and identity information embedded. (The private/public key pair is generated either by the CA or on John's machine, which depends on the systems' configurations. If it is created at the CA, his private key needs to be sent to him by secure means. In most cases, the user generates this pair and sends in his public key during the registration process.)

Now John is registered and can participate in a PKI. John and Diane decide they want to communicate, so they take the following steps, shown in Figure 8-20:

1. John requests Diane's public key from a public directory.
2. The directory, sometimes called a repository, sends Diane's digital certificate.
3. John verifies the digital certificate and extracts her public key. John uses this public key to encrypt a session key that will be used to encrypt their messages. John sends the encrypted session key to Diane. John also sends his certificate, containing his public key, to Diane.
4. When Diane receives John's certificate, she verifies that she trusts the CA that digitally signed it. Diane trusts this CA and, after she verifies the certificate, decrypts the session key John sent her using her private key. Now, both John and Diane can communicate securely using symmetric key encryption and the shared session key.

A PKI may be made up of the following entities and functions:

- Certification authority
- Registration authority
- Certificate repository
- Certificate revocation system

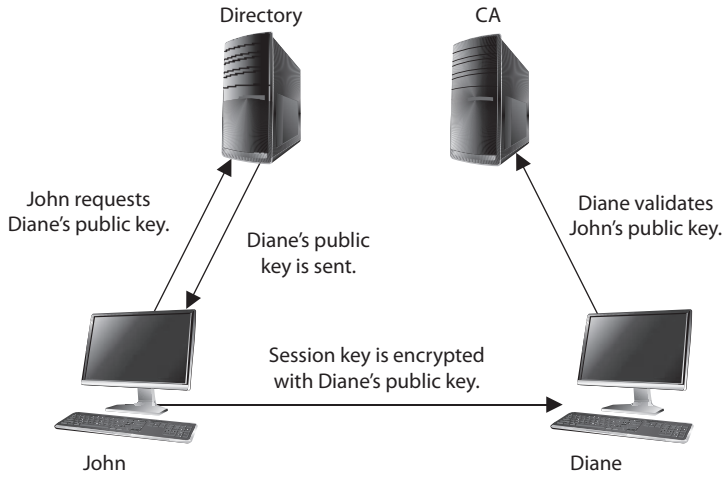


Figure 8-20
CA and user relationships

- Key backup and recovery system
- Automatic key update
- Management of key histories
- Timestamping
- Client-side software

PKI supplies the following security services:

- Confidentiality
- Access control
- Integrity
- Authentication
- Nonrepudiation

A PKI must retain a key history, which keeps track of all the old and current public keys that have been used by individual users. For example, if Kevin encrypted a symmetric key with Dave's old public key, there should be a way for Dave to still access this data. This can only happen if the CA keeps a proper history of Dave's old certificates and keys.

NOTE Another important component that must be integrated into a PKI is a reliable time source that provides a way for secure timestamping. This comes into play when true nonrepudiation is required.

Key Management

Cryptography can be used as a security mechanism to provide confidentiality, integrity, and authentication, but not if the keys are compromised in any way. The keys can be captured, modified, corrupted, or disclosed to unauthorized individuals. Cryptography is based on a trust model. Individuals must trust each other to protect their own keys; trust the CA who issues the keys; and trust a server that holds, maintains, and distributes the keys.

Many administrators know that key management causes one of the biggest headaches in cryptographic implementation. There is more to key maintenance than using them to encrypt messages. The keys must be distributed securely to the right entities and updated as needed. They must also be protected as they are being transmitted and while they are being stored on each workstation and server. The keys must be generated, destroyed, and recovered properly. Key management can be handled through manual or automatic processes.

The keys are stored before and after distribution. When a key is distributed to a user, it does not just hang out on the desktop. It needs a secure place within the file system to be stored and used in a controlled method. The key, the algorithm that will use the key, configurations, and parameters are stored in a module that also needs to be protected.

If an attacker is able to obtain these components, she could masquerade as another user and decrypt, read, and re-encrypt messages not intended for her.

The Kerberos authentication protocol (which we will describe in Chapter 17) uses a Key Distribution Center (KDC) to store, distribute, and maintain cryptographic session and secret keys. This provides an automated method of key distribution. The computer that wants to access a service on another computer requests access via the KDC. The KDC then generates a session key to be used between the requesting computer and the computer providing the requested resource or service. The automation of this process reduces the possible errors that can happen through a manual process, but if the KDC gets compromised in any way, then all the computers and their services are affected and possibly compromised.

In some instances, keys are still managed through manual means. Unfortunately, although many organizations use cryptographic keys, they rarely, if ever, change them, either because of the hassle of key management or because the network administrator is already overtaxed with other tasks or does not realize the task actually needs to take place. The frequency of use of a cryptographic key has a direct correlation to how often the key should be changed. The more a key is used, the more likely it is to be captured and compromised. If a key is used infrequently, then this risk drops dramatically. The necessary level of security and the frequency of use can dictate the frequency of key updates. A mom-and-pop diner might only change its cryptography keys every month, whereas an information warfare military unit might change them daily. The important thing is to change the keys using a secure method.

Key management is the most challenging part of cryptography and also the most crucial. It is one thing to develop a very complicated and complex algorithm and key method, but if the keys are not securely stored and transmitted, it does not really matter how strong the algorithm is.

Key Management Principles

Keys should not be in cleartext outside the cryptography device. As stated previously, many cryptography algorithms are known publicly, which puts more stress on protecting the secrecy of the key. If attackers know how the actual algorithm works, in many cases, all they need to figure out is the key to compromise a system. This is why keys should not be available in cleartext—the key is what brings secrecy to encryption.

These steps, and all of key distribution and maintenance, should be automated and hidden from the user. These processes should be integrated into software or the operating system. It only adds complexity and opens the doors for more errors when processes are done manually and depend upon end users to perform certain functions.

Keys are at risk of being lost, destroyed, or corrupted. Backup copies should be available and easily accessible when required. If data is encrypted and then the user accidentally loses the necessary key to decrypt it, this information would be lost forever if there were not a backup key to save the day. The application being used for cryptography may have key recovery options, or it may require copies of the keys to be kept in a secure place.

Different scenarios highlight the need for key recovery or backup copies of keys. For example, if Bob has possession of all the critical bid calculations, stock value information,

and corporate trend analysis needed for tomorrow's senior executive presentation, and Bob has an unfortunate confrontation with a bus, someone is going to need to access this data after the funeral. As another example, if an employee leaves the company and has encrypted important documents on her computer before departing, the company would probably still want to access that data later. Similarly, if the vice president did not know that running a large magnet over the USB drive that holds his private key was not a good idea, he would want his key replaced immediately instead of listening to a lecture about electromagnetic fields and how they rewrite sectors on media.

Of course, having more than one key increases the chance of disclosure, so an organization needs to decide whether it wants to have key backups and, if so, what precautions to put into place to protect them properly. An organization can choose to have multiparty control for emergency key recovery. This means that if a key must be recovered, more than one person is needed for this process. The key recovery process could require two or more other individuals to present their private keys or authentication information. These individuals should not all be members of the IT department. There should be a member from management, an individual from security, and one individual from the IT department, for example. All of these requirements reduce the potential for abuse and would require collusion for fraudulent activities to take place.

Rules for Keys and Key Management

Key management is critical for proper protection. The following are responsibilities that fall under the key management umbrella:

- The key length should be long enough to provide the necessary level of protection.
- Keys should be stored and transmitted by secure means.
- Keys should be random, and the algorithm should use the full spectrum of the keyspace.
- The key's lifetime should correspond with the sensitivity of the data it is protecting. (Less secure data may allow for a longer key lifetime, whereas more sensitive data might require a shorter key lifetime.)
- The more the key is used, the shorter its lifetime should be.
- Keys should be backed up or escrowed in case of emergencies.
- Keys should be properly destroyed when their lifetime comes to an end.

Key escrow is a process or entity that can recover lost or corrupted cryptographic keys; thus, it is a common component of key recovery operations. When two or more entities are required to reconstruct a key for key recovery processes, this is known as *multiparty key recovery*. Multiparty key recovery implements dual control, meaning that two or more people have to be involved with a critical task.

Of course, this creates a bit of a problem if two (or three, or whatever) people are required for key recovery but one of them is missing. What do you do at a point of crisis if Carlos is one of the people required to recover the key but he's on a cruise in the middle of the Pacific Ocean? To solve this, you can use an approach called *m-of-n control* (or *quorum authentication*), in which you designate a group of (*n*) people as recovery agents

The Web of Trust

An alternative approach to using certificate authorities is called the *web of trust*, which was introduced by Phil Zimmermann for use in Pretty Good Privacy (PGP) cryptosystems. In a web of trust, people sign each other's certificates if they have verified their identity and trust them. This can happen, for example, at key-signing parties where people meet and sign each other's certificates. Thereafter, anyone who has signed a certificate can either share it with others as trusted or subsequently vouch for that certificate if asked to. This decentralized approach is popular among many security practitioners but is not practical for most commercial applications.

and only need a subset (*m*) of them for key recovery. So, you could choose three people in your organization (n = 3) as key recovery agents, but only two of them (m = 2) would need to participate in the actual recovery process for it to work. In this case, your m-of-n control would be 2-of-3.

NOTE More detailed information on key management best practices can be found in NIST Special Publication 800-57, Part 1 Revision 5, *Recommendation for Key Management: Part 1 – General*.

Attacks Against Cryptography

We've referred multiple times in this chapter to adversaries attacking our cryptosystems, but how exactly do they carry out those attacks? Sometimes, it's as simple as listening to network traffic and picking up whatever messages they can. Eavesdropping and sniffing data as it passes over a network are considered *passive attacks* because the attacker is not affecting the protocol, algorithm, key, message, or any parts of the encryption system. Passive attacks are hard to detect, so in most cases methods are put in place to try to prevent them rather than to detect and stop them.

Altering messages, modifying system files, and masquerading as another individual are acts that are considered *active attacks* because the attacker is actually doing something instead of sitting back and gathering data. Passive attacks are usually used to gain information prior to carrying out an active attack.

The common attack vectors in cryptography are key and algorithm, implementation, data, and people. We should assume that the attacker knows what algorithm we are using and that the attacker has access to all encrypted text. The following sections address some active attacks that relate to cryptography.

Key and Algorithm Attacks

The first class of attack against cryptosystems targets the algorithms themselves or the keyspace they use. Except for brute forcing, these approaches require a significant level of knowledge of the mathematical principles underpinning cryptography. They are relatively rare among attackers, with the possible exception of state actors with significant

intelligence capabilities. They are, however, much more common when a new algorithm is presented to the cryptographic community for analysis prior to their adoption.

Brute Force

Sometimes, all it takes to break a cryptosystem is to systematically try all possible keys until you find the right one. This approach is called a *brute-force* attack. Of course, cryptographers know this and develop systems that are resistant to brute forcing. They do the math and ensure that brute forcing takes so long to work that it is computationally infeasible for adversaries to try this approach and succeed. But there's a catch: computational power and, importantly, improved techniques to more efficiently use it are growing each year. We can make assumptions about where these capabilities will be in five or ten years, but we really can't be sure how long it'll be until that key that seemed strong enough all of a sudden is crackable through brute force.

Ciphertext-Only Attacks

In a *ciphertext-only attack*, the attacker has the ciphertext of one or more messages, each of which has been encrypted using the same encryption algorithm and key. The attacker's goal is to discover the key used in the encryption process. Once the attacker figures out the key, she can decrypt all other messages encrypted with the same key.

A ciphertext-only attack is the most common type of active attack because it is very easy to get ciphertext by sniffing someone's traffic, but it is the hardest attack to carry out successfully because the attacker has so little information about the encryption process. Unless the attackers have nation-state resources at their disposal, it is very unlikely that this approach will work.

Known-Plaintext Attacks

In a *known-plaintext attack*, the attacker has the plaintext and corresponding ciphertext of one or more messages and wants to discover the key used to encrypt the message(s) so that he can decipher and read other messages. This attack can leverage known patterns in message composition. For example, many corporate e-mail messages end with a standard confidentiality disclaimer, which the attacker can easily acquire by getting an unencrypted e-mail from anyone in that organization. In this instance, the attacker has some of the plaintext (the data that is the same on each message) and can capture encrypted messages, knowing that some of the ciphertext corresponds to this known plaintext. Rather than having to cryptanalyze the entire message, the attacker can focus on that part of it that is known. Some of the first encryption algorithms used in computer networks would generate the same ciphertext when encrypting the same plaintext with the same key. Known-plaintext attacks were used by the United States against the Germans and the Japanese during World War II.

Chosen-Plaintext Attacks

In a *chosen-plaintext attack*, the attacker has the plaintext and ciphertext, but can choose the plaintext that gets encrypted to see the corresponding ciphertext. This gives the attacker more power and possibly a deeper understanding of the way the encryption process works so that she can gather more information about the key being used. Once the attacker discovers the key, she can decrypt other messages encrypted with that key.

How would this be carried out? Doris can e-mail a message to you that she thinks you not only will believe, but will also panic about, encrypt, and send to someone else. Suppose Doris sends you an e-mail that states, "The meaning of life is 42." You may think you have received an important piece of information that should be concealed from others, everyone except your friend Bob, of course. So you encrypt Doris's message and send it to Bob. Meanwhile Doris is sniffing your traffic and now has a copy of the plaintext of the message, because she wrote it, and a copy of the ciphertext.

Chosen-Ciphertext Attacks

In a *chosen-ciphertext attack*, the attacker can choose the ciphertext to be decrypted and has access to the resulting decrypted plaintext. Again, the goal is to figure out the key. This is a harder attack to carry out compared to the previously mentioned attacks, and the attacker may need to have control of the system that contains the cryptosystem.

NOTE All of these attacks have a derivative form, the names of which are the same except for putting the word "adaptive" in front of them, such as adaptive chosen-plaintext and adaptive chosen-ciphertext. What this means is that the attacker can carry out one of these attacks and, depending upon what she gleaned from that first attack, modify her next attack. This is the process of reverse-engineering or cryptanalysis attacks: using what you learned to improve your next attack.

Differential Cryptanalysis

This type of attack also has the goal of uncovering the key that was used for encryption purposes. A *differential cryptanalysis attack* looks at ciphertext pairs generated by encryption of plaintext pairs with specific differences and analyzes the effect and result

Public vs. Secret Algorithms

The public mainly uses algorithms that are known and understood versus the secret algorithms where the internal processes and functions are not released to the public. In general, cryptographers in the public sector feel as though the strongest and best-engineered algorithms are the ones released for peer review and public scrutiny, because a thousand brains are better than five, and many times some smarty-pants within the public population can find problems within an algorithm that the developers did not think of. This is why vendors and companies have competitions to see if anyone can break their code and encryption processes. If someone does break it, that means the developers must go back to the drawing board and strengthen this or that piece.

Not all algorithms are released to the public, such as the ones developed by the NSA. Because the sensitivity level of what the NSA encrypts is so important, it wants as much of the process to be as secret as possible. The fact that the NSA does not release its algorithms for public examination and analysis does not mean its algorithms are weak. Its algorithms are developed, reviewed, and tested by many of the top cryptographic pros around, and are of very high quality.

of those differences. One such attack was effectively used in 1990 against the Data Encryption Standard, but turned out to also work against other block algorithms.

The attacker takes two messages of plaintext and follows the changes that take place to the blocks as they go through the different S-boxes. (Each message is being encrypted with the same key.) The differences identified in the resulting ciphertext values are used to map probability values to different possible key values. The attacker continues this process with several more sets of messages and reviews the common key probability values. One key value will continue to show itself as the most probable key used in the encryption processes. Since the attacker chooses the different plaintext messages for this attack, it is considered a type of chosen-plaintext attack.

Frequency Analysis

A *frequency analysis*, also known as a *statistical attack*, identifies statistically significant patterns in the ciphertext generated by a cryptosystem. For example, the number of zeroes may be significantly higher than the number of ones. This could show that the pseudorandom number generator (PRNG) in use may be biased. If keys are taken directly from the output of the PRNG, then the distribution of keys would also be biased. The statistical knowledge about the bias could be used to reduce the search time for the keys.

Implementation Attacks

All of the attacks we have covered thus far have been based mainly on the mathematics of cryptography. We all know that there is a huge difference between the theory of how something should work and how the widget that comes off the assembly line actually works. *Implementation flaws* are system development defects that could compromise a real system, and *implementation attacks* are the techniques used to exploit these flaws. With all the emphasis on developing and testing strong algorithms for encryption, it should come as no surprise that cryptosystems are far likelier to have implementation flaws than to have algorithmic flaws.

One of the best-known implementation flaws is the Heartbleed bug discovered in 2014 in the OpenSSL cryptographic software library estimated to have been in use by two-thirds of the world's servers. Essentially, a programmer used an insecure function call that allowed an attacker to copy arbitrary amounts of data from the victim computer's memory, including encryption keys, usernames, and passwords.

There are multiple approaches to finding implementation flaws, whether you are an attacker trying to exploit them or a defender trying to keep them from being exploited. In the sections that follow, we look at some of the most important techniques to keep in mind.

Source Code Analysis

The first, and probably most common, approach to finding implementation flaws in cryptosystems is to perform *source code analysis*, ideally as part of a large team of researchers, and look for bugs. Through a variety of software auditing techniques (which we will cover in Chapter 25), source code analysis examines each line of code and branch of execution to determine whether it is vulnerable to exploitation. This is most practical when the code is open source or you otherwise have access to its source. Sadly, as with Heartbleed, this approach can fail to reveal major flaws for many years.

Reverse Engineering

Another approach to discovering implementation flaws in cryptosystems involves taking a product and tearing it apart to see how it works. This is called *reverse engineering* and can be applied to both software and hardware products. When you buy software, you normally get binary executable programs, so you can't do the source code analysis discussed in the previous section. However, there are a number of ways in which you can disassemble those binaries and get code that is pretty close to the source code. Software reverse engineering requires a lot more effort and skill than regular source code analysis, but it is more common that most would think.

A related practice, which applies to hardware and firmware implementations, involves something called *hardware reverse engineering*. This means the researcher is directly probing integrated circuit (IC) chips and other electronic components. In some cases, chips are actually peeled apart layer by layer to show internal interconnections and even individual bits that are set in memory structures. This approach oftentimes requires destroying the device as it is dissected, probed, and analyzed. The effort, skill, and expense required can sometimes yield implementation flaws that would be difficult or impossible to find otherwise.

Side-Channel Attacks

Using plaintext and ciphertext involves high-powered mathematical tools that are needed to uncover the key used in the encryption process. But what if we took a different approach? What if we paid attention to what happens around the cryptosystem as it does its business? As an analogy, burglars can unlock a safe and determine its combination by feeling the change in resistance as they spin the dial and listening to the mechanical clicks inside the lock.

Similarly, in cryptography, we can review facts and infer the value of an encryption key. For example, we could detect how much power consumption is used for encryption and decryption (the fluctuation of electronic voltage). We could also intercept the radiation emissions released and then calculate how long the processes took. Looking around the cryptosystem, or its attributes and characteristics, is different from looking into the cryptosystem and trying to defeat it through mathematical computations.

If Omar wants to figure out what you do for a living, but he doesn't want you to know he is doing this type of reconnaissance work, he won't ask you directly. Instead, he will find out when you go to work and when you come home, the types of clothing you wear, the items you carry, and whom you talk to—or he can just follow you to work. These are examples of *side channels*.

So, in cryptography, gathering "outside" information with the goal of uncovering the encryption key is just another way of attacking a cryptosystem. An attacker could measure power consumption, radiation emissions, and the time it takes for certain types of data processing. With this information, he can work backward by reverse-engineering the process to uncover an encryption key or sensitive data. A power attack reviews the amount of heat released. This type of attack has been successful in uncovering confidential information from smart cards.

In 1995, RSA private keys were uncovered by measuring the relative time cryptographic operations took. This type of side-channel attack is also called a *timing attack* because

it uses time measurements to determine the inner workings, states, and even data flows within a cryptosystem. Timing attacks can also result in theft of sensitive information, including keys. Although the Meltdown and Spectre attacks of 2017 were not technically examples of cryptanalysis, they could be used to steal keys and are probably the best-known examples of timing attacks.

The idea is that instead of attacking a device head on, just watch how it performs to figure out how it works. In biology, scientists can choose to carry out a noninvasive experiment, which involves watching an organism eat, sleep, mate, and so on. This type of approach learns about the organism through understanding its behaviors instead of killing it and looking at it from the inside out.

Fault Injection

Cryptanalysts can deliberately introduce conditions that are designed to cause the system to fail in some way. This can be done in connection with one of the previous implementation techniques, or on its own. *Fault injection attacks* attempt to cause errors in a cryptosystem in an attempt to recover or infer the encryption key. Though this attack is fairly rare, it received a lot of attention in 2001 after it was shown to be effective after only one injection against the RSA using Chinese Remainder Theorem (RSA-CRT). According to some experts, a fault injection attack is a special case of a side-channel attack.

Other Attacks

Unless you or your adversary are skilled cryptanalysts or are employed by a national intelligence organization, you are fairly unlikely to be on the receiving end of one of the previous attacks. You may, as happened with Heartbleed, be caught up in a broader attack, however. We now turn our attention to a set of attacks that are much more likely to be targeted against you or your organization.

Replay Attacks

A big concern in distributed environments is the *replay attack*, in which an attacker captures some type of data and resubmits it with the hopes of fooling the receiving device into thinking it is legitimate information. Many times, the data captured and resubmitted is authentication information, and the attacker is trying to authenticate herself as someone else to gain unauthorized access.

Pass the hash is a well-known replay attack that targets Microsoft Windows Active Directory (AD) single sign-on environments. As we will explore more deeply in Chapters 16 and 17, *single sign-on (SSO)* is any authentication approach that requires the user to authenticate only once and then automatically provides access to network resources as requested without manual user reauthentication. Microsoft implements SSO by storing a hash of the user password locally and then automatically using that for future service requests without any user interaction. The Local Security Authority Subsystem Service (LSASS) is a process in Microsoft Windows that is responsible for verifying user logins, handling password changes, and managing access tokens such as password hashes. Any user with local admin rights can dump LSASS memory from a Windows computer and recover password hashes for any user who has recently logged into that system.

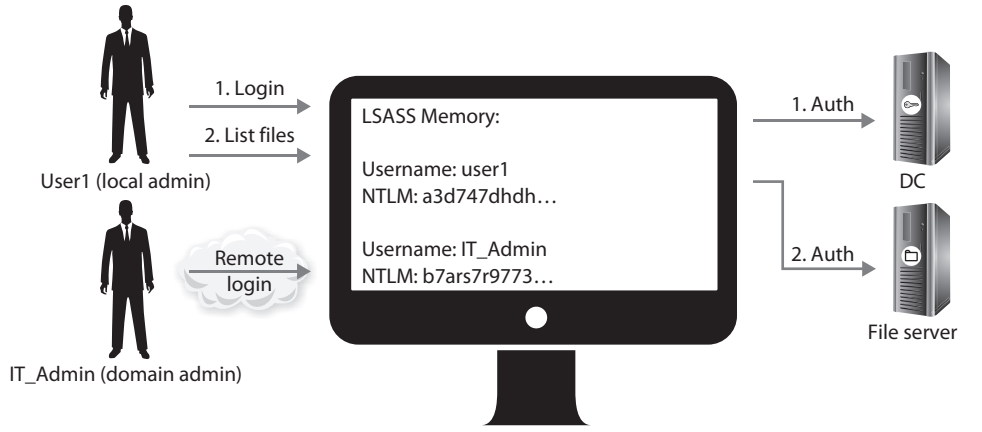


Figure 8-21 Single sign-on in Microsoft Windows Active Directory

In the example shown in Figure 8-21, User1 logs into the system locally. LSASS authenticates the user with the domain controller (DC) and then stores the username and New Technology LAN Manager (NTLM) password hash in memory for future use. User1 later browses files on the file server and, rather than having to reenter credentials, LSASS authenticates User1 automatically with the file server using the cached username and hash. A domain admin has also logged in remotely to update the host, so her username and hash are cached in memory, too.

Now, suppose an attacker sends a malicious attachment to User1, who then opens it and compromises the host, as shown in Figure 8-22. The attacker can now interact

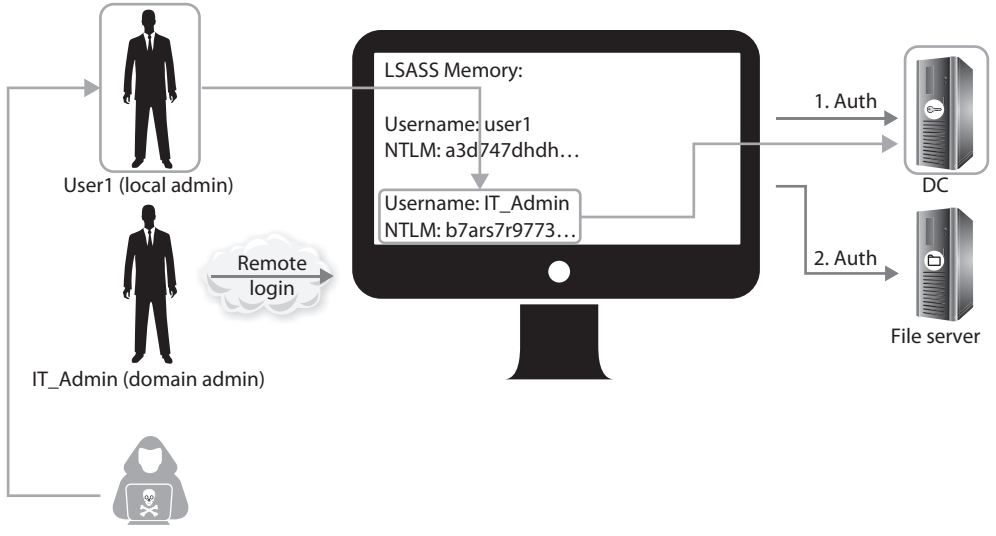


Figure 8-22 Pass-the-hash attack

with the compromised system using User1's permissions and, because that user is a local admin, is able to dump hashes from LSASS memory. Now the attacker has the hash of the domain admin's password, which grants him access to the domain controller without having to crack any passwords at all.

Timestamps and sequence numbers are two countermeasures to replay attacks. Packets can contain sequence numbers, so each machine will expect a specific number on each receiving packet. If a packet has a sequence number that has been previously used, that is an indication of a replay attack. Packets can also be timestamped. A threshold can be set on each computer to only accept packets within a certain timeframe. If a packet is received that is past this threshold, it can help identify a replay attack.

Man-in-the-Middle

Hashes are not the only useful things you can intercept if you can monitor network traffic. If you can't compromise the algorithms or implementations of cryptosystems, the next best thing is to insert yourself into the process by which secure connections are established. In *man-in-the-middle (MitM)* attacks, threat actors intercept an outbound secure connection request from clients and relay their own requests to the intended servers, terminating both and acting as a proxy. This allows attackers to defeat encrypted channels without having to find vulnerabilities in the algorithms or their implementations.

Figure 8-23 shows a MitM attack used in a phishing campaign. First, the attacker sends an e-mail message enticing the victim to click a link that looks like a legitimate one but leads to a server controlled by the attacker instead. The attacker then sends her own request to the server and establishes a secure connection to it. Next, the attacker completes the connection requested by the client but using her own certificate instead of the intended server's. The attacker is now invisibly sitting in the middle of two separate, secure connections. From this vantage point, the attacker can either relay information from one end to the other, perhaps copying some of it (e.g., credentials, sensitive documents, etc.) for later use. The attacker can also selectively modify the information sent from one end to the other. For example, the attacker could change the destination account for a funds transfer.

You will notice in Figure 8-23 that the certificate of the legitimate site (goodsite .com) is different than the one used by the attacker (g00dsite.com, which has two zeroes

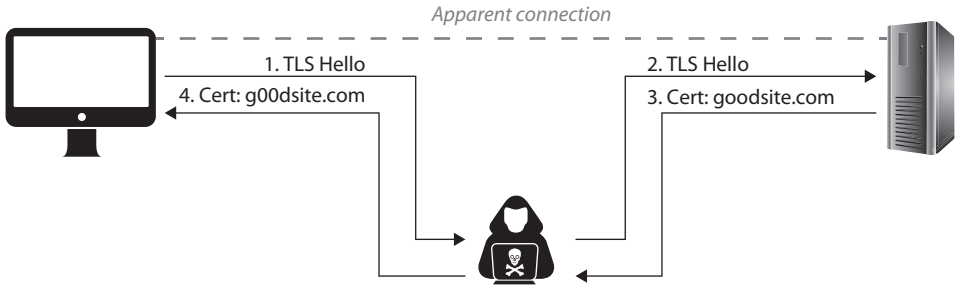


Figure 8-23 A web-based man-in-the-middle attack

instead of letters "o"). The attacker needs to present her own certificate because she needs the corresponding private key to complete the connection with the client and be able to share the secret session key. There are a few ways in which the attacker can make this less noticeable to the user. The first is to send the link to the server in an e-mail to the user. The HTML representation of the link is the legitimate site, while the actual (hidden) link points to the almost identical but malicious domain. A more sophisticated attacker can use a variety of techniques to compromise DNS resolution and have the client go to the malicious site instead of the legitimate one. Either way, the browser is likely to generate a warning letting the user know that something is not right. Fortunately for the attackers (and unfortunately for us), most users either do not pay attention to or actively disregard such warnings.

Social Engineering Attacks

It should come as no surprise that people can be fooled by clever attackers who can trick them into providing their cryptographic key material through various social engineering attack types. As discussed in earlier chapters, social engineering attacks are carried out on people with the goal of tricking them into divulging some type of sensitive information that can be used by the attacker. For example, an attacker may convince a victim that the attacker is a security administrator who requires the cryptographic data for some type of operational effort. The attacker could then use the data to decrypt and gain access to sensitive data. Social engineering attacks can be carried out through deception, persuasion, coercion (rubber-hose cryptanalysis), or bribery (purchase-key attack).

Ransomware

Ransomware is a type of malware that typically encrypts victims' files and holds them ransom until a payment is made to an account controlled by the attacker. When the victim pays, the attacker usually (but not always) provides the secret key needed to decrypt the files. It is not so much an attack against cryptography as it is an attack *employing* cryptography. Ransomware attacks are typically delivered through a phishing e-mail that contains a malicious attachment. After the initial compromise, however, the ransomware may be able to move laterally across the victim's network, infecting other hosts.

Chapter Review

Cryptographic algorithms provide the underlying tools to most security protocols used in today's infrastructures. They are, therefore, an integral tool for cybersecurity professionals. The cryptographic algorithms work off of mathematical functions and provide various types of functionality and levels of security. Every algorithm has strengths and weaknesses, so we tend to use them in hybrid systems such as public key infrastructures. Symmetric and asymmetric key cryptography, working with hashing functions, provide a solid foundation on which to build the security architectures we'll discuss in the next chapter.

Of course, there are many ways to attack these cryptosystems. Advanced adversaries may find vulnerabilities in the underlying algorithms. Others may target the manner in which these algorithms are implemented in software and hardware. Most attackers, however, simply attempt to bypass cryptography by replaying authentication data,

inserting themselves in the middle of a trusted communications channel, or simply targeting the people involved through social engineering.

Quick Review

- Cryptography is the practice of storing and transmitting information in a form that only authorized parties can understand.
- A readable message is in a form called plaintext, and once it is encrypted, it is in a form called ciphertext.
- Cryptographic algorithms are the mathematical rules that dictate the functions of enciphering and deciphering.
- Cryptanalysis is the name collectively given to techniques that aim to weaken or defeat cryptography.
- Nonrepudiation is a service that ensures the sender cannot later falsely deny sending a message.
- The range of possible keys is referred to as the keyspace. A larger keyspace and the full use of the keyspace allow for more-random keys to be created. This provides more protection.
- The two basic types of encryption mechanisms used in symmetric ciphers are substitution and transposition. Substitution ciphers change a character (or bit) out for another, while transposition ciphers scramble the characters (or bits).
- A polyalphabetic cipher uses more than one alphabet to defeat frequency analysis.
- A key is a random string of bits inserted into an encryption algorithm. The result determines what encryption functions will be carried out on a message and in what order.
- In symmetric key algorithms, the sender and receiver use the same key for encryption and decryption purposes.
- In asymmetric key algorithms, the sender and receiver use different keys for encryption and decryption purposes.
- The biggest challenges in employing symmetric key encryption are secure key distribution and scalability. However, symmetric key algorithms perform much faster than asymmetric key algorithms.
- Symmetric key algorithms can provide confidentiality, but not authentication or nonrepudiation.
- Examples of symmetric key algorithms include AES and ChaCha20.
- Asymmetric algorithms are typically used to encrypt keys, and symmetric algorithms are typically used to encrypt bulk data.
- Asymmetric key algorithms are much slower than symmetric key algorithms but can provide authentication and nonrepudiation services.
- Examples of asymmetric key algorithms include RSA, ECC, and DSA.

- Two main types of symmetric algorithms are stream ciphers and block ciphers. Stream ciphers use a keystream generator and encrypt a message one bit at a time. A block cipher divides the message into groups of bits and encrypts them.
- Many algorithms are publicly known, so the secret part of the process is the key. The key provides the necessary randomization to encryption.
- RSA is an asymmetric algorithm developed by Rivest, Shamir, and Adleman and is the de facto standard for digital signatures.
- Elliptic curve cryptosystems (ECCs) are used as asymmetric algorithms and can provide digital signatures, secure key distribution, and encryption functionality. ECCs use fewer resources, which makes them better for wireless device and cell phone encryption use.
- Quantum cryptography is the field of scientific study that applies quantum mechanics to perform cryptographic functions. The most immediate application of this field is quantum key distribution (QKD), which generates and securely distributes encryption keys of any length between two parties.
- When symmetric and asymmetric key algorithms are used together, this is called a hybrid system. The asymmetric algorithm encrypts the symmetric key, and the symmetric key encrypts the data.
- A session key is a symmetric key used by the sender and receiver of messages for encryption and decryption purposes. The session key is only good while that communication session is active and then it is destroyed.
- A public key infrastructure (PKI) is a framework of programs, procedures, communication protocols, and public key cryptography that enables a diverse group of individuals to communicate securely.
- A certificate authority (CA) is a trusted third party that generates and maintains user certificates, which hold their public keys.
- The CA uses a certification revocation list (CRL) to keep track of revoked certificates.
- A certificate is the mechanism the CA uses to associate a public key to a person's identity.
- A registration authority (RA) validates the user's identity and then sends the request for a certificate to the CA. The RA cannot generate certificates.
- A one-way function is a mathematical function that is easier to compute in one direction than in the opposite direction.
- RSA is based on a one-way function that factors large numbers into prime numbers. Only the private key knows how to use the trapdoor and how to decrypt messages that were encrypted with the corresponding public key.
- Hashing algorithms provide data integrity only.
- When a hash algorithm is applied to a message, it produces a message digest, and this value is signed with a private key to produce a digital signature.
- Some examples of hashing algorithms include SHA-1, SHA-2, SHA-3, and MD5.
- SHA produces a 160-bit hash value and is used in DSS.

- A birthday attack is an attack on hashing functions through brute force. The attacker tries to create two messages with the same hashing value.
- A one-time pad uses a pad with random values that are XORed against the message to produce ciphertext. The pad is at least as long as the message itself and is used once and then discarded.
- A digital signature is the result of a user signing a hash value with a private key. It provides authentication, data integrity, and nonrepudiation. The act of signing is the actual encryption of the value with the private key.
- Key management is one of the most challenging pieces of cryptography. It pertains to creating, maintaining, distributing, and destroying cryptographic keys.
- Brute-force attacks against cryptosystems systematically try all possible keys against given ciphertext in hopes of guessing the key that was used.
- Ciphertext-only attacks against cryptosystems involve analyzing the ciphertext of one or more messages encrypted with the same algorithm and key in order to discover the key that was used.
- In a known-plaintext attack, the attacker has the plaintext and corresponding ciphertext of one or more messages and wants to discover the key that was used.
- A chosen-plaintext attack is like a known-plaintext attack but the attacker chooses the plaintext that gets encrypted to see the corresponding ciphertext.
- A chosen-ciphertext attack is like a chosen-plaintext attack except that the attacker chooses the ciphertext and then gets to see the corresponding decrypted plaintext.
- A frequency analysis, also known as a statistical attack, identifies statistically significant patterns in the ciphertext generated by a cryptosystem.
- Implementation attacks are the techniques used to exploit defects in the implementation of a cryptosystem.
- Side-channel attacks analyze changes in the environment around a cryptosystem in an attempt to infer an encryption key whose processing causes those changes.
- Timing attacks are side-channel attacks that use time measurements to determine the inner workings, states, and even data flows within a cryptosystem.
- Fault injection attacks attempt to cause errors in a cryptosystem in an attempt to recover or infer the encryption key.
- In man-in-the-middle (MitM) attacks, threat actors intercept an outbound secure connection request from clients and relay their own requests to the intended servers, terminating both and acting as a proxy.
- Pass the hash is a type of attack against Microsoft Windows Active Directory in which the attacker resubmits cached authentication tokens to gain illicit access to resources.
- Ransomware is a type of malware that encrypts victims' files and holds them ransom until a payment is made to an account controlled by the attacker.

Questions

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

1. What is the goal of cryptanalysis?
   - **A.** To determine the strength of an algorithm
   - **B.** To increase the substitution functions in a cryptographic algorithm
   - **C.** To decrease the transposition functions in a cryptographic algorithm
   - **D.** To determine the permutations used
2. Why has the frequency of successful brute-force attacks increased?
   - **A.** The use of permutations and transpositions in algorithms has increased.
   - **B.** As algorithms get stronger, they get less complex, and thus more susceptible to attacks.
   - **C.** Processor speed and power have increased.
   - **D.** Key length reduces over time.
3. Which of the following is not a property or characteristic of a one-way hash function?
   - **A.** It converts a message of arbitrary length into a value of fixed length.
   - **B.** Given the digest value, finding the corresponding message should be computationally infeasible.
   - **C.** Deriving the same digest from two different messages should be impossible or rare.
   - **D.** It converts a message of fixed length to an arbitrary length value.
4. What would indicate that a message had been modified?
   - **A.** The public key has been altered.
   - **B.** The private key has been altered.
   - **C.** The message digest has been altered.
   - **D.** The message has been encrypted properly.
5. Which of the following is a U.S. federal government algorithm developed for creating secure message digests?
   - **A.** Data Encryption Algorithm
   - **B.** Digital Signature Standard
   - **C.** Secure Hash Algorithm
   - **D.** Data Signature Algorithm

6. What is an advantage of RSA over DSA?
   - **A.** It can provide digital signature and encryption functionality.
   - **B.** It uses fewer resources and encrypts faster because it uses symmetric keys.
   - **C.** It is a block cipher rather than a stream cipher.
   - **D.** It employs a one-time encryption pad.
7. What is used to create a digital signature?
   - **A.** The receiver's private key
   - **B.** The sender's public key
   - **C.** The sender's private key
   - **D.** The receiver's public key
8. Which of the following best describes a digital signature?
   - **A.** A method of transferring a handwritten signature to an electronic document
   - **B.** A method to encrypt confidential information
   - **C.** A method to provide an electronic signature and encryption
   - **D.** A method to let the receiver of the message prove the source and integrity of a message
9. Why would a certificate authority revoke a certificate?
   - **A.** If the user's public key has become compromised
   - **B.** If the user changed over to using the PEM model that uses a web of trust
   - **C.** If the user's private key has become compromised
   - **D.** If the user moved to a new location
10. Which of the following best describes a certificate authority?
    - **A.** An organization that issues private keys and the corresponding algorithms
    - **B.** An organization that validates encryption processes
    - **C.** An organization that verifies encryption keys
    - **D.** An organization that issues certificates
11. Which of the following is a true statement pertaining to data encryption when it is used to protect data?
    - **A.** It verifies the integrity and accuracy of the data.
    - **B.** It requires careful key management.
    - **C.** It does not require much system overhead in resources.
    - **D.** It requires keys to be escrowed.

12. What is the definition of an algorithm's work factor?

- **A.** The time it takes to encrypt and decrypt the same plaintext
- **B.** The time it takes to break the encryption
- **C.** The time it takes to implement 16 rounds of computation
- **D.** The time it takes to apply substitution functions

13. What is the primary purpose of using one-way hashing on user passwords?

- **A.** It minimizes the amount of primary and secondary storage needed to store passwords.
- **B.** It prevents anyone from reading passwords in plaintext.
- **C.** It avoids excessive processing required by an asymmetric algorithm.
- **D.** It prevents replay attacks.

14. Which of the following is based on the fact that it is hard to factor large numbers into two original prime numbers?

- **A.** ECC
- **B.** RSA
- **C.** SHA
- **D.** MD5

15. What is the name given to attacks that involve analyzing changes in the environment around a cryptosystem to infer the encryption key whose processing causes those changes?

- **A.** Side-channel attack
- **B.** Timing attack
- **C.** Implementation attack
- **D.** Fault injection attack

Answers

1. A. Cryptanalysis is the process of trying to reverse-engineer a cryptosystem, with the possible goal of uncovering the key used. Once this key is uncovered, all other messages encrypted with this key can be accessed. Cryptanalysis is carried out by the white hats to test the strength of the algorithm.

2. C. A brute-force attack is resource intensive. It tries all values until the correct one is obtained. As computers have more powerful processors added to them, attackers can carry out more powerful brute-force attacks.

3. **D.** A hashing algorithm will take a string of variable length (the message can be any size) and compute a fixed-length value. The fixed-length value is the message digest. The MD family creates the fixed-length value of 128 bits, and SHA creates one of 160 bits.
4. **C.** Hashing algorithms generate message digests to detect whether modification has taken place. The sender and receiver independently generate their own digests, and the receiver compares these values. If they differ, the receiver knows the message has been altered.
5. **C.** SHA was created to generate secure message digests. Digital Signature Standard (DSS) is the standard to create digital signatures, which dictates that SHA must be used. DSS also outlines the digital signature algorithms that can be used with SHA: RSA, DSA, and ECDSA.
6. **A.** RSA can be used for data encryption, key exchange, and digital signatures. DSA can be used only for digital signatures.
7. **C.** A digital signature is a message digest that has been encrypted with the sender's private key. A sender, or anyone else, should never have access to the receiver's private key.
8. **D.** A digital signature provides authentication (knowing who really sent the message), integrity (because a hashing algorithm is involved), and nonrepudiation (the sender cannot deny sending the message).
9. **C.** The reason a certificate is revoked is to warn others who use that person's public key that they should no longer trust the public key because, for some reason, that public key is no longer bound to that particular individual's identity. This could be because an employee left the company or changed his name and needed a new certificate, but most likely it is because the person's private key was compromised.
10. **D.** A registration authority (RA) accepts a person's request for a certificate and verifies that person's identity. Then the RA sends this request to a certificate authority (CA), which generates and maintains the certificate.
11. **B.** Data encryption always requires careful key management. Most algorithms are so strong today that it is much easier to go after key management than to launch a brute-force attack. Hashing algorithms are used for data integrity, encryption does require a good amount of resources, and keys do not have to be escrowed for encryption.
12. **B.** The work factor of a cryptosystem is the amount of time and resources necessary to break the cryptosystem or its encryption process. The goal is to make the work factor so high that an attacker could not be successful in breaking the algorithm or cryptosystem.
13. **B.** Passwords are usually run through a one-way hashing algorithm so that the actual password is not transmitted across the network or stored on a system in plaintext. This greatly reduces the risk of an attacker being able to obtain the actual password.

14. **B.** The RSA algorithm's security is based on the difficulty of factoring large numbers into their original prime numbers. This is a one-way function. Calculating the product is easier than identifying the prime numbers used to generate that product.
15. **A.** Side-channel attack is the best answer. The question could also describe a timing attack, which is a kind of side-channel attack, but because there is no specific mention of timing, this option is not the best answer. An argument could also be made for this being a fault injection attack but, again, there is no specific mention of the attacker deliberately trying to cause errors. This question is representative of the harder questions in the CISSP exam, which provide one or more answers that could be correct but are not the best ones.

This page intentionally left blank

CHAPTER 9

Security Architectures

This chapter presents the following:

- Threat modeling
- Security architecture design principles
- Security models
- Addressing security requirements
- Security capabilities of information systems

Security is a process, not a product.

—Bruce Schneier

Having discussed the various information system architectures (in Chapter 7) and cryptology (in Chapter 8), our next step is to put these together as we develop an enterprise security architecture. We already discussed the various frameworks for this in Chapter 4, so what we need to do now is to apply tried and true principles, models, and system capabilities. Before we do this, however, we will explore threat modeling in a fair amount of detail, since it informs everything else we do.

Threat Modeling

Before we can develop effective defenses, it is imperative to understand the assets that we value, as well as the threats against which we are protecting them. Though multiple definitions exist for the term, for the purposes of our discussion we define *threat modeling* as the process of describing probable adverse effects on our assets caused by specific threat sources. That's quite a mouthful, so let's break it down.

When we build a model of the threats we face, we want to ground them in reality, so it is important to only consider dangers that are reasonably likely to occur. To do otherwise would dilute our limited resources to the point of making us unable to properly defend ourselves. Next, we want to focus our work on the potential impact of those threats to organizational assets or, in other words, to things and people that are of value to the organization. Lastly, the model needs to specify the threat sources if we are to develop effective means to thwart them. We must understand their capabilities and motivations if we are to make sense of their actions.

Attack Trees

An *attack tree* is a graph showing how individual actions by attackers can be chained together to achieve their goals. This methodology is based on the observation that, typically, there are multiple ways to accomplish a given objective. For example, if a disgruntled employee wanted to steal the contents of the president's mailbox, she could accomplish this by either accessing the e-mail server, obtaining the password, or stealing the president's smartphone. Accessing the e-mail server could be accomplished by using administrative credentials or by hacking in. To get the credentials, she could use brute force or social engineering. The options available to the attacker create the branches in the tree, an example of which is shown in Figure 9-1. Each of the leaf nodes represents a specific condition that must be met in order for the parent node to be effective. For instance, to effectively obtain the mailbox credentials, the disgruntled employee could have stolen a network access token. Given that the employee has met the condition of having the credentials, she would then be able to steal the contents of the president's mailbox. A successful attack, then, is one in which the attacker traverses from a leaf node all the way to the root node at the top of the tree, which represents the ultimate objective.

NOTE The terms "attack chain" and "kill chain" are commonly used. They refer to a specific type of attack tree that has no branches and simply proceeds from one stage or action to the next. The attack tree is much more expressive in that it shows many ways in which an attacker can accomplish each objective.

Reduction Analysis

The generation of attack trees for an organization usually requires a large investment of resources. Each vulnerability-threat-attack triad can be described in detail using an attack tree, so you end up with as many trees as you do triads. To defeat each of the attacks

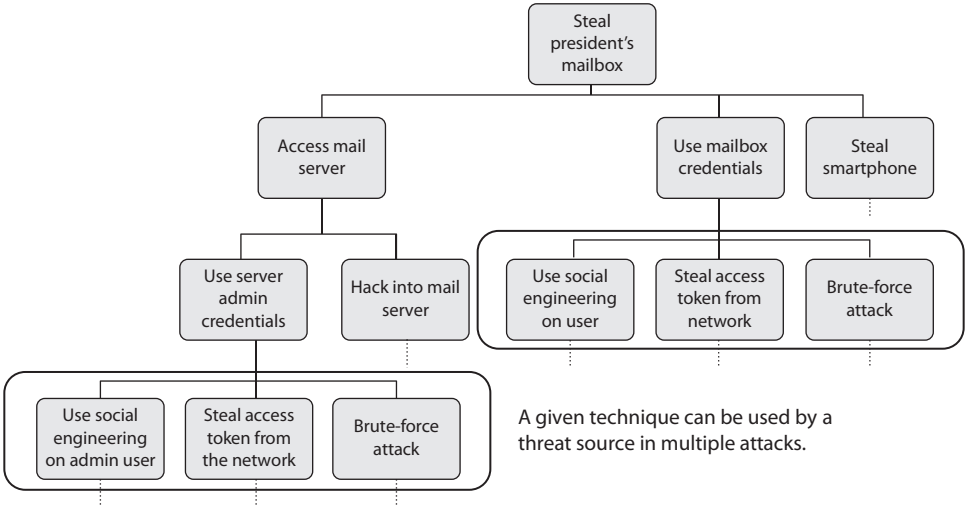


Figure 9-1 A simplified attack tree

you identify, you would typically need a control or countermeasure at each leaf node. Since one attack generates many leaf nodes, this has a multiplicative effect that could make it very difficult to justify the whole exercise. However, attack trees lend themselves to a methodology known as *reduction analysis*.

There are two aspects of reduction analysis in the context of threat modeling: one aspect is to reduce the number of attacks we have to consider, and the other is to reduce the threat posed by the attacks. The first aspect is evidenced by the commonalities in the example shown in Figure 9-1. To satisfy the conditions for logging into the mail server or the user's mailbox, an attacker can use the exact same three techniques. This means we can reduce the number of conditions we need to mitigate by finding these commonalities. When you consider that these three sample conditions apply to a variety of other attacks, you realize that we can very quickly cull the number of conditions to a manageable number.

The second aspect of reduction analysis is the identification of ways to mitigate or negate the attacks we've identified. This is where the use of attack trees can really benefit us. Recall that each tree has only one root but many leaves and internal nodes. The closer you are to the root when you implement a mitigation technique, the more leaf conditions you will defeat with that one control. This allows you to easily identify the most effective techniques to protect your entire organization. These techniques are typically called *controls* or *countermeasures*.

STRIDE

STRIDE is a threat modeling framework that evaluates a system's design using flow diagrams, system entities, and events related to a system. STRIDE is not an acronym, but a mnemonic for the six categories of security threats outlined in Table 9-1. STRIDE was developed in 1999 by Microsoft as a tool to help secure systems as they were being developed. STRIDE is among the most widely used threat-modeling frameworks and is suitable for application to logical and physical systems alike.

The Lockheed Martin Cyber Kill Chain

Threat modeling is really nothing new. It probably has its oldest roots in military operations, where it is used to anticipate the intent and actions of an enemy and then develop a plan to get inside their decision loop and defeat them. The term *kill chain* evolved to describe the process of identifying a target, determining the best way to engage it, amassing the required forces against it, engaging it, and destroying it. In 2011, Lockheed Martin released a paper defining a *Cyber Kill Chain* based on this methodology. It identifies the steps that threat actors generally must complete to achieve their objectives. This model specifies seven distinct stages of a cyberattack:

1. **Reconnaissance** The attacker selects a target, researches it, and attempts to identify vulnerabilities in the target network.
2. **Weaponization** The attacker either adapts an existing remote access malware weapon or creates a new one, tailored to one or more vulnerabilities identified in the previous step.

Threat	Property Affected	Definition	Example
Spoofing	Authentication	Impersonating someone or something else	An outside sender pretending to be an HR employee in an e-mail
Tampering	Integrity	Modifying data on disk, in memory, or elsewhere	A program modifying the contents of a critical system file
Repudiation	Nonrepudiation	Claiming to have not performed an action or to have no knowledge of who performed it	A user claiming that she did not receive a request
Information disclosure	Confidentiality	Exposing information to parties not authorized to see it	An analyst accidentally revealing the inner details of the network to outside parties
Denial of service	Availability	Denying or degrading service to legitimate users by exhausting resources needed for a service	A botnet flooding a website with thousands of requests a second, causing it to crash
Elevation of privilege	Authorization	Gaining capabilities without the proper authorization to do so	A user bypassing local restrictions to gain administrative access to a workstation

Table 9-1 STRIDE Threat Categories

3. **Delivery** The attacker transmits the weapon to the target (e.g., via e-mail attachments, links to malicious websites, or USB drives).
4. **Exploitation** The malware weapon triggers, which takes action on the target to exploit one or more vulnerabilities and compromise the host.
5. **Installation** The malware weapon installs an access point (e.g., backdoor) usable by the attacker.
6. **Command and Control** The malware enables the attacker to have "hands on the keyboard" persistent access to the target network.
7. **Actions on Objective** The attacker takes actions to achieve their goals, such as data exfiltration, data destruction, or encryption for ransom.

One of Lockheed Martin's key goals of developing this model was to allow defenders to map defensive measures to each stage and ensure that they have sufficient coverage to detect, deny, disrupt, degrade, deceive, or contain the attack. The earlier in the kill chain this is done, the better, because the adversary will not have attained their objective yet. Another key idea in the model is that of identifying indicators of adversarial activity at each stage of a cyberattack, which would allow defenders to detect the activities but also

determine whether the defensive measures at a particular stage were effective. Though the Cyber Kill Chain is a high-level framework, it is one of the most commonly used ones for modeling threat activities.

The MITRE ATT&CK Framework

The MITRE Corporation developed a framework of adversarial tactics, techniques, and common knowledge called ATT&CK as a comprehensive matrix of tactics and techniques used by threat actors. It is a widely used tool to construct models of complex campaigns and operations using reusable common components. Like the Cyber Kill Chain, ATT&CK breaks down actions into (generally) sequential groupings called *tactics*, which can be mapped to those in the Lockheed Martin model. Each of the 14 tactics contains a number of *techniques* by which the adversary may achieve a particular purpose. The techniques, in turn, contain specific *sub-techniques* used by named threat actors.

For example, the eleventh tactic (T0011), Command and Control, describes techniques used by adversaries when trying to communicate with compromised systems to control them. One of these techniques (T1071) deals with the use of application layer protocols to establish communications in a discrete manner. One of the sub-techniques (T1071.004) involves the use of the Domain Name System (DNS) to send and receive messages covertly. This sub-technique, in turn, contains examples of procedures used by various known threat actors, such as OceanLotus (also known as APT32) using a DNS tunneling for command and control on its Denis Trojan.

Why Bother with Threat Modeling

A scientific model is a simplified representation of something that is difficult to understand. The model is built in a way that makes certain parts of the subject easier to observe and analyze. We use these models to study complex phenomena like the spread of disease, global financial markets, and, of course, cybersecurity threats. Threat modeling allows us to simplify some of the activities of our adversaries so we can drill into the parts that really matter to us as defenders. There are just too many threat actors doing too many discrete things in too many places for us to study each in detail. Instead, we look for likely attackers and patterns that allow us to mitigate a bunch of different attacks by defeating the techniques that they all have in common.

Let's continue our previous example of the Denis Trojan using DNS tunneling. Should we care about it? That depends on what organization we belong to. We would probably care if we happen to be part of a media and human rights organizations in or around Vietnam, where OceanLotus/APT32 appears to be focused. This is the power of threat modeling: it allows us to focus keenly on specific actors and specific techniques based on who and where our organization is.

A typical threat modeling effort would start by identifying threat actors that are likelier to target our organization. This can be general classes of actors, such as opportunistic ransomware gangs, or more specific ones, like OceanLotus. Where do we get this information? Maybe we read the news and look for attackers that are targeting organizations such as ours. Or maybe we are subscribed to threat intelligence services

that specifically look for those who are coming after us. Either way, we start by answering a series of questions:

- Why might someone want to target our organization? (Motive)
- How could they go about accomplishing their objectives? (Means)
- When and where would they attack us? (Opportunity)

The answer to the first question comes from our asset inventory. What do we have that is of value to others? Intellectual property would be valuable to competitors. Cybercriminals would like to get any financial data. Foreign governments would be after national security information. Once we know what kind of threat actors would be interested in us, we can study how they go about attacking organizations like ours. Going back to our example, suppose we are in an organization that might be of interest to OceanLotus. We can research any available open or private sources to learn about their tactics, techniques, and procedures (TTPs). The MITRE ATT&CK framework, for instance, lists over 40 techniques and at least 13 tools used by OceanLotus. Knowing their motivation and means, we can examine our systems and determine where they could use those techniques. This is the power of threat modeling: it allows us to focus our attention on understanding the threats that are likeliest to be of concern to our organizations. Out of the countless TTPs used by adversaries around the world, we can focus our attention on those that matter most to us.

Secure Design Principles

Understanding our threats is foundational to building secure architectures, but so is applying the collective wisdom developed by the security community. This wisdom exists in certain secure design principles that are widely recognized as best practices. The sections that follow address the secure design principles that you should know for the CISSP exam. Think of them as a solid starting point, not as an all-inclusive list.

EXAM TIP You should be able to describe each of the 11 secure design principles (which include the previously covered threat modeling) in some detail and recognize when they are (and are not) being followed in a given scenario.

Defense in Depth

One of the bedrock principles in designing security architectures is the assumption that our jobs are not to determine *whether* our systems can be compromised, but rather to prepare for the inevitable reality that this *will* happen. Consequently, we want to provide *defense in depth*, which is the coordinated use of multiple security controls in a layered approach, as shown in Figure 9-2. A multilayered defense system reduces the probability of successful penetration and compromise because an attacker would have to get through several different types of protection mechanisms before she gained access to the

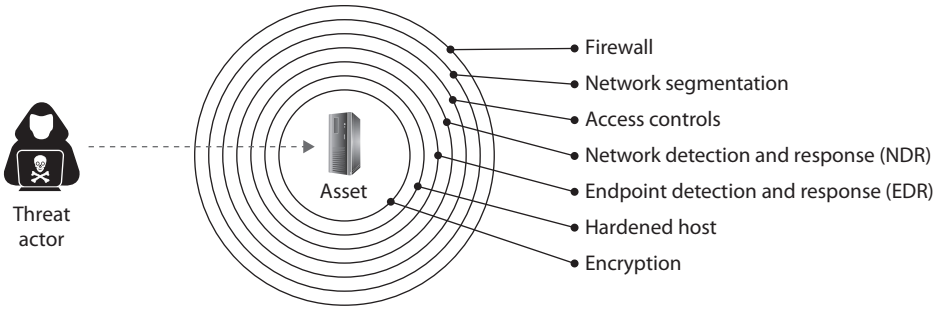


Figure 9-2 Defense in depth

critical assets. She may succeed at penetrating the perimeter and may get a foothold in the environment, but security controls are arranged in a way that will slow her down, improve the odds of detecting her, and provide means for defeating the attack or quickly recovering from it.

A well-designed security architecture considers the interplay of physical, technical, and administrative controls. For example, Company A can protect its facilities by using physical security controls such as the following:

- Perimeter fencing
- External lights
- Locked external and internal doors
- Security guard
- Security cameras

These measures would force the attacker, in many cases, to conduct the attack through the network. Technical controls would then have to provide adequate protection to the various assets in the company. Like their physical counterparts, you can think of these technical controls as creating concentric circles of protection around your assets. For example, as shown in Figure 9-2, you could have a perimeter firewall on the outer ring. If the attacker circumvents this, she'd have to navigate through a segmented network with strictly enforced access controls everywhere. Beyond this she'd have to defeat network detection and response (NDR) systems that can detect and stop the attack as it moves across your network. Deeper still, there are endpoint detection and response (EDR) systems that are deployed on hardened hosts. Finally, all data is encrypted, which makes exfiltrating anything useful out of the company's environment more difficult.

Physical and technical controls are integrated and augmented by administrative controls such as policies, procedures, and standards. The types of controls that are actually implemented must map to the threats the company faces, and the number of layers that are put into place must map to the sensitivity of the asset. The rule of thumb is the more sensitive the asset, the more layers of protection that must be put into place.

Zero Trust

The principle of defense in depth is certainly useful, but some people may think it suggests that attacks always follow the pattern of external threat actors sequentially penetrating each defensive circle until they get to the asset being protected. In reality, attacks can come from any direction (even internally) and proceed nonlinearly. Some adversaries lay dormant in our networks for days, weeks, or even months waiting to complete their attacks from the inside. This reality, compounded by the threat of malicious or just careless insiders, has led many security professionals to a place of healthy paranoia.

A *zero trust* model is one in which every entity is considered hostile until proven otherwise. It considers trust as a vulnerability and tries to reduce or eliminate it in order to make it harder for threat actors (external or internal) to accomplish their objectives. If defense in depth looks at security from the outside in, zero trust architectures are built from the inside out. These are not mutually exclusive approaches, however. It is best to incorporate both principles as we design our systems.

The catch is that it is very hard to implement a zero trust model throughout an enterprise environment because it would hinder productivity and efficiency. For this reason, this approach is most often focused only on a relatively small group of critical assets, access to which defines a "protect surface," which is where the tightest controls are placed.

Trust But Verify

Another principle of designing secure architectures is *trust but verify*, which basically means that, even when an entity and its behaviors are trusted, we should double-check both. It could seem that this is an alternative to (or even incompatible with) the zero trust model, but that is not necessarily true. You could, for instance, take a zero trust approach to defending your most critical assets, while allowing certain trust (that you verify) elsewhere. Of course, the zero trust assets would also be verified, and frequently. In this manner, the two approaches can coexist happily. On the other hand, you could take a hardline approach to one or the other and build your entire security architecture around it. It really depends on your organization and its environment.

At the heart of the trust but verify principle is the implementation of mechanisms that allow you to audit everything that happens on your systems. How else could you verify them, right? But what is equally important is the development of processes by which you pay the right amount of attention to auditing different elements of your environment. You won't have the ability to examine everything all the time so you have to figure out how you'll be able to analyze some things most of the time, and most things some of the time. The risk is that we leave some parts of our environment unexamined and therefore create a safe haven for attackers. Procedures matter just as much as technology.

Shared Responsibility

While the previous three principles (defense in depth, zero trust, and trust but verify) can apply equally well to traditional, cloud, and hybrid environments, cloud computing adds a bit of complexity in terms of defensive roles. *Shared responsibility* refers to the

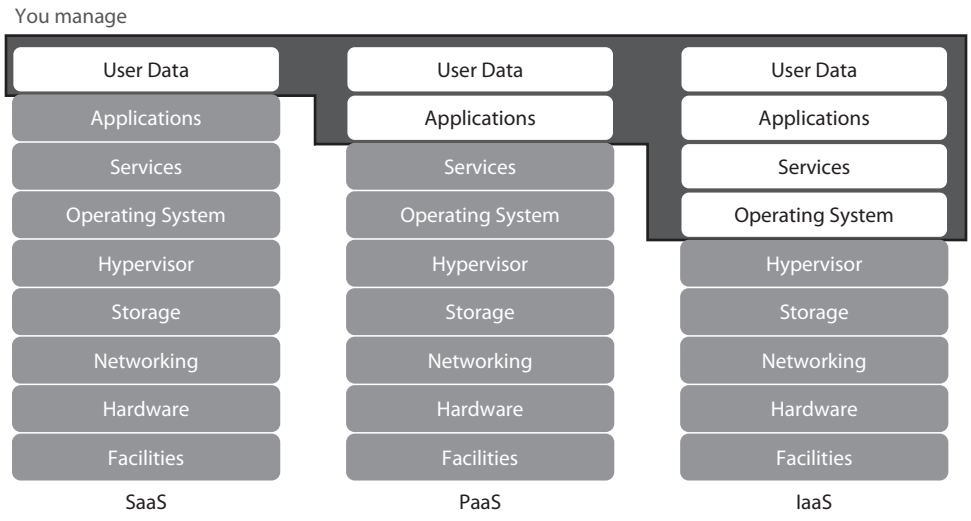


Figure 9-3 Shared responsibility in different cloud computing services

situation in which a service provider is responsible for certain security controls, while the customer is responsible for others. Typically, the service provider is responsible for security of the "thing" that they are providing. Think back to the three predominant cloud computing models we discussed in Chapter 7: Software as a Service (SaaS), Infrastructure as a Service (IaaS), and Platform as a Service (PaaS). Figure 9-3 shows the breakdown of responsibilities in each model.

The figure is, of course, a generalization. You should carefully read the fine print in the service agreements and discuss with your provider exactly what they will be doing for you and what they assume you'll do for yourself. Then, you need to periodically ensure that these parameters have not changed due to changes in the environment, a revised agreement, or new features offered by your provider. Many cloud compromises can be traced to organizations not understanding they were responsible for providing a certain aspect of security.

Separation of Duties

Regardless of your physical or logical architecture, people will be responsible for performing various business, infrastructure, and security functions based on their roles within the organization. It is important to consider the human element in designing our security architectures. Granting access rights to our teammates should be based on the level of trust the organization has in them and in their need to know and do certain things. Just because a company completely trusts Joyce with its files and resources does not mean she fulfills the need-to-know criteria to access the company's tax returns and profit margins. If Maynard fulfills the need-to-know criteria to access employees' work histories, it does not mean the company trusts him to access all of the company's other files. These issues

must be identified and integrated into the access criteria. The different access criteria can be enforced by roles, groups, location, time, and transaction types.

Using *roles* is an efficient way to assign rights to a type of user who performs a certain task. This role is based on a job assignment or function. If there is a position within a company for a person to audit transactions and audit logs, the role this person fills would only need a read function to those types of files. If several users require the same type of access to information and resources, putting them into a *group* and then assigning rights and permissions to that group is easier to manage than assigning rights and permissions to each and every individual separately. If a specific printer is available only to the accounting group, when a user attempts to print to it, the group membership of the user will be checked to see if she is indeed in the accounting group.

Transaction-type restrictions can be used to further control what data is accessed during certain types of functions and what commands can be carried out on the data. A purchasing agent in a company may be able to make purchases of up to $2,000, but would need a supervisor's approval to buy more expensive items. The supervisor, conversely, might not be allowed to make any purchases at all, simply approve those submitted by a subordinate. This is an example of the principle of *separation of duties (SoD)*, in which important functions are divided among multiple individuals to ensure that no one person has the ability to intentionally or accidentally cause serious losses to the organization.

Least Privilege

A related principle, *least privilege*, states that people are granted exactly the access and authority that they require to do their jobs, and nothing more. The purchasing agent's supervisor from the example in the previous section is not really expected to swipe his corporate credit card as part of his daily duties, so why should he even have a card? His purchasing agent is expected to make small purchases on a regular basis, but anything over $2,000 would be fairly rare, so why have a purchase limit any higher than that?

The *need-to-know* principle is similar to the least-privilege principle. It is based on the concept that individuals should be given access only to the information they absolutely require in order to perform their job duties. Giving any more rights to a user just asks for headaches and the possibility of that user abusing the permissions assigned to him. An administrator wants to give a user the least amount of privileges she can, but just enough for that user to be productive when carrying out tasks. Management decides what a user needs to know, or what access rights are necessary, and the administrator configures the access control mechanisms to allow this user to have that level of access and no more, and thus the least privilege.

For example, if management has decided that Dan, the intern, needs to know where the files he needs to copy are located and needs to be able to print them, this fulfills Dan's need-to-know criteria. Now, an administrator could give Dan full control of all the files he needs to copy, but that would not be practicing the least-privilege principle. The administrator should restrict Dan's rights and permissions to only allow him to read and print the necessary files, and no more. Besides, if Dan accidentally deletes all the files on

Authorization Creep

As employees work at an organization over time and move from one department to another, they often are assigned more and more access rights and permissions. This is commonly referred to as *authorization creep*. It can be a large risk for an organization, because too many users have too much privileged access to organizational assets. In the past, it has usually been easier for network administrators to give more access than less, because then the user would not come back and require more work to be done on her profile. It is also difficult to know the exact access levels different individuals require. This is why user management and user provisioning are becoming more prevalent in identity management products today and why organizations are moving more toward role-based access control implementation. Enforcing least privilege on user accounts should be an ongoing job, which means each user's rights are permissions that should be reviewed to ensure the organization is not putting itself at risk.

the whole file server, who do you think management will hold ultimately responsible? Yep, the administrator.

It is important to understand that it is management's job to determine the security requirements of individuals and how access is authorized. The security administrator configures the security mechanisms to fulfill these requirements, but it is not her job to determine security requirements of users. Those should be left to the owners. If there is a security breach, management will ultimately be held responsible, so it should make these decisions in the first place.

Keep It Simple

The secure design principles we've discussed thus far can introduce complexity into our information systems. As many of us have learned the hard way, the more complex a system is, the more difficult it is to understand and protect it. This is where the principle of *simplicity* comes in: make everything as simple as possible and periodically check things to ensure we are not adding unnecessary complexity.

This principle has long been known in the software development community, where one of the metrics routinely tracked is errors per 1,000 lines of code, also known as defects per KLOC (kilo-lines of code). The idea is that the more code you write, the greater the odds you'll make an error without noticing it. Similarly, the more unique hosts, facilities, or policies you have, the greater the odds that a vulnerability will be introduced inadvertently.

Obviously, we can't force a global enterprise to use a small network, but we can simplify things by standardizing configurations. We can have 10,000 endpoints around the world, all configured in just a handful of ways with no exceptions allowed. We can similarly ensure that all our facilities use the same security protocols and that our policies

are few in number, simple to understand, and uniformly enforced. The fewer variables we have to track, the simpler our systems become and, by extension, the easier they are to secure.

Secure Defaults

As many people in the technology field know, out-of-the-box implementations are oftentimes far from secure. Many systems come out of the box in an insecure state because settings have to be configured to properly integrate a system into different environments, and this is a friendlier way of installing the product for users. For example, if Mike is installing a new software package that continually throws messages of "Access Denied" when he is attempting to configure it to interoperate with other applications and systems, his patience might wear thin, and he might decide to hate that vendor for years to come because of the stress and confusion inflicted upon him.

Yet again, we are at a hard place for developers and architects. When a security application or device is installed, it should default to "No Access." This means that when Laurel installs a firewall, it should not allow any traffic to pass into the network that was not specifically granted access. A fine balance exists between security, functionality, and user friendliness. If an application is extremely user friendly or has lots of features, it is probably not as secure. It is up to us as security professionals to help our organizations balance these three competing needs.

NOTE Most operating systems nowadays ship with reasonably secure default settings, but users are still able to override the majority of these settings. This brings us closer to "default with no access," but we still have a ways to go.

The principle of *secure defaults* means that every system starts off in a state where security trumps user friendliness and functionality. It is later that, as deliberate decisions, security personnel can relax some restrictions, enable additional features, and generally make the system more user friendly. These decisions are integrated into the risk management program (see Chapter 2), typically through the configuration management processes (which we'll discuss in Chapter 20). The goal of secure defaults is to start everything in a place of extreme security and then intentionally loosen things until users can get their jobs done, but no further.

Fail Securely

A design principle that is related to secure defaults deals with the manner in which failures are handled. In the event of an error, information systems ought to be designed to behave in a predictable and noncompromising manner. This is also generally referred to as *failing securely*. We already saw how violating this principle can give adversaries an advantage when we discussed some of the cryptanalysis approaches in the previous chapter. If adversaries can induce a fault in a cryptosystem, they could recover a partial key or otherwise compromise it. The same holds for any information system.

A fail-secure system defaults to the highest level of security when it encounters a fault. For example, a firewall that is powered off unexpectedly may block all traffic when it reboots until an administrator can verify that it is still configured and operating securely. An EDR system may lock out a system if it encounters certain critical errors. Finally, a web browser could prevent a user from visiting a website if the connection is not secure or the digital certificate doesn't match, is not trusted, or is expired or revoked. Many a successful phishing campaign could've been defeated by implementing this last example alone!

Privacy by Design

The best way to ensure privacy of user data is to incorporate data protection as an integral part of the design of an information system, not as an afterthought or later-stage feature. This is the principle of *privacy by design* in a nutshell. Apart from making sense (we hope), this principle is part of the European Union's General Data Protection Regulation (GDPR), which means your organization may be required to abide by it already.

Privacy by design is not a new concept. It was originally introduced in the 1990s and formally described in a joint report by the Dutch Data Protection Authority and the Ontario Information Commissioner, titled *Privacy by Design: Delivering the Promises*, in 2010. This document describes the seven foundational principles of privacy by design, which are listed here:

1. Proactive, not reactive; preventative, not remedial
2. Privacy as the default setting
3. Privacy embedded into design
4. Full functionality—positive-sum, not zero-sum
5. End-to-end security—full life-cycle protection
6. Visibility and transparency—keep it open
7. Respect for user privacy—keep it user-centric

Security Models

A security model is a more formal way to capture security principles. Whereas a principle is a rule of thumb that can be adapted to different situations, the security models we describe here are very specific and verifiable. A security model is usually represented in mathematics and analytical ideas, which are mapped to system specifications and then implemented in software and/or hardware by product developers. So we have a policy that encompasses security goals, such as "each subject must be authenticated and authorized before accessing an object." The security model takes this requirement and provides the necessary mathematical formulas, relationships, and logic structure to be followed to accomplish this goal. From there, specifications are developed per operating system type

(Unix, Windows, macOS, and so on), and individual vendors can decide how they are going to implement mechanisms that meet these necessary specifications.

Several security models have been developed to enforce security policies. The following sections provide overviews of the models with which you must be familiar as a CISSP.

Bell-LaPadula Model

The *Bell-LaPadula model* enforces the *confidentiality* aspects of access control. It was developed in the 1970s to prevent secret information from being accessed in an unauthorized manner. It was the first mathematical model of a multilevel security policy used to define the concept of secure modes of access and outlined rules of access. Its development was funded by the U.S. government to provide a framework for computer systems that would be used to store and process sensitive information. A system that employs the Bell-LaPadula model is called a *multilevel security system* because users with different clearances use the system, and the system processes data at different classification levels.

EXAM TIP The Bell-LaPadula model was developed to make sure secrets stay secret; thus, *it provides and addresses confidentiality only*. This model does not address the integrity of the data the system maintains—only who can and cannot access the data and what operations can be carried out.

Three main rules are used and enforced in the Bell-LaPadula model:

- Simple security rule
- *-property (star property) rule
- Strong star property rule

The *simple security rule* states that a subject at a given security level cannot read data that resides at a higher security level. For example, if Bob is given the security clearance of secret, this rule states he cannot *read* data classified as top secret. If the organization wanted Bob to be able to read top-secret data, it would have given him that clearance in the first place.

The **-property rule* (star property rule) states that a subject in a given security level cannot *write* information to a lower security level. The simple security rule is referred to as the "no read up" rule, and the *-property rule is referred to as the "no write down" rule.

The *strong star property rule* states that a subject who has read and write capabilities can only perform both of those functions at the same security level; nothing higher and nothing lower. So, for a subject to be able to read and write to an object, the subject's clearance and the object classification must be equal.

Biba Model

The *Biba model* is a security model that addresses the *integrity* of data within a system. It is not concerned with security levels and confidentiality. The Biba model uses integrity levels to prevent data at any integrity level from flowing to a higher integrity level. Biba has three main rules to provide this type of protection:

- ***-integrity axiom** A subject cannot write data to an object at a higher integrity level (referred to as "no write up").
- **Simple integrity axiom** A subject cannot read data from a lower integrity level (referred to as "no read down").
- **Invocation property** A subject cannot request service (invoke) at a higher integrity.

A simple example might help illustrate how the Biba model could be used in a real context. Suppose that Indira and Erik are on a project team and are writing two documents: Indira is drafting meeting notes for internal use and Erik is writing a report for the CEO. The information Erik uses in writing his report must be very accurate and reliable, which is to say it must have a high level of integrity. Indira, on the other hand, is just documenting the internal work being done by the team, including ideas, opinions, and hunches. She could use unconfirmed and maybe even unreliable sources when writing her document. The **-integrity axiom* dictates that Indira would not be able to contribute (write) material to Erik's report, though there's nothing to say she couldn't use Erik's (higher integrity) information in her own document. The *simple integrity axiom*, on the other hand, would prevent Erik from even reading Indira's document because it could potentially introduce lower integrity information into his own (high integrity) report.

The *invocation property* in the Biba model states that a subject cannot invoke (call upon) a subject at a higher integrity level. How is this different from the other two Biba rules? The *-integrity axiom (no write up) dictates how subjects can *modify* objects. The simple integrity axiom (no read down) dictates how subjects can *read* objects. The invocation property dictates how one subject can communicate with and initialize other

Bell-LaPadula vs. Biba

The Bell-LaPadula and Biba models are informational flow models because they are most concerned about data flowing from one level to another. Bell-LaPadula uses security levels to provide data *confidentiality*, and Biba uses integrity levels to provide data *integrity*.

It is important for CISSP test takers to know the rules of Bell-LaPadula and Biba, and their rules sound similar. Both have "simple" and "* (star)" rules—one writing one way and one reading another way. A tip for how to remember them is if the word "simple" is used, the rule is about *reading*. If the rule uses * or "star," it is about *writing*. So now you just need to remember the reading and writing directions per model.

subjects at run time. An example of a subject invoking another subject is when a process sends a request to a procedure to carry out some type of task. Subjects are only allowed to invoke tools at a lower integrity level. With the invocation property, the system is making sure a dirty subject cannot invoke a clean tool to contaminate a clean object.

Clark-Wilson Model

The *Clark-Wilson model* was developed after Biba and takes some different approaches to protecting the integrity of information. This model uses the following elements:

- **Users** Active agents
- **Transformation procedures (TPs)** Programmed abstract operations, such as read, write, and modify
- **Constrained data items (CDIs)** Can be manipulated only by TPs
- **Unconstrained data items (UDIs)** Can be manipulated by users via primitive read and write operations
- **Integrity verification procedures (IVPs)** Check the consistency of CDIs with external reality

A distinctive feature of the Clark-Wilson model is that it focuses on well-formed transactions and separation of duties. A *well-formed transaction* is a series of operations that transforms a data item from one consistent state to another. Think of a consistent state as one wherein we know the data is reliable. This consistency ensures the integrity of the data and is the job of the TPs. Separation of duties is implemented in the model by adding a type of procedure (the IVPs) that audits the work done by the TPs and validates the integrity of the data.

When a system uses the Clark-Wilson model, it separates data into one subset that needs to be highly protected, which is referred to as a constrained data item (CDI), and another subset that does not require a high level of protection, which is called an unconstrained data item (UDI). Users cannot modify critical data (CDI) directly. Instead, software procedures (TPs) carry out the operations on behalf of the users. This is referred to as *access triple*: subject (user), program (TP), and object (CDI). A user cannot modify a CDI without using a TP. The UDI does not require such a high level of protection and can be manipulated directly by the user.

Remember that this is an integrity model, so it must have something that ensures that specific integrity rules are being carried out. This is the job of the IVP. The IVP ensures that all critical data (CDI) manipulation follows the application's defined integrity rules.

Noninterference Model

Multilevel security properties can be expressed in many ways, one being *noninterference*. This concept is implemented to ensure any actions that take place at a higher security level do not affect, or interfere with, actions that take place at a lower level. This type of model does not concern itself with the flow of data, but rather with what a subject

knows about the state of the system. So, if an entity at a higher security level performs an action, it cannot change the state for the entity at the lower level. If a lower-level entity was aware of a certain activity that took place by an entity at a higher level and the state of the system changed for this lower-level entity, the entity might be able to deduce too much information about the activities of the higher state, which, in turn, is a way of leaking information.

Let's say that Tom and Kathy are both working on a multilevel mainframe at the same time. Tom has the security clearance of secret, and Kathy has the security clearance of top secret. Since this is a central mainframe, the terminal Tom is working at has the context of secret, and Kathy is working at her own terminal, which has a context of top secret. This model states that nothing Kathy does at her terminal should directly or indirectly affect Tom's domain (available resources and working environment). The commands she executes or the resources she interacts with should not affect Tom's experience of working with the mainframe in any way.

The real intent of the noninterference model is to address covert channels. The model looks at the shared resources that the different users of a system will access and tries to identify how information can be passed from a process working at a higher security clearance to a process working at a lower security clearance. Since Tom and Kathy are working on the same system at the same time, they will most likely have to share some types of resources. So the model is made up of rules to ensure that Kathy cannot pass data to Tom through covert channels.

Covert Channels

A *covert channel* is a way for an entity to receive information in an unauthorized manner. These communications can be very difficult to detect. Covert channels are of two types: storage and timing. In a *covert storage channel*, processes are able to communicate through some type of storage space on the system. For example, suppose Adam wants to leak classified information to Bob. Adam could create a user account on a web system. Bob pretends he will create an account on the same system and checks to see if the username is available. If it is available, that is the equivalent of a zero (no account existed). Otherwise, he records a one, and aborts the creation of the account. Either way, Bob waits a given amount of time. Adam either removes the account, effectively writing a zero, or ensures one exists (which would be a one). Bob tries again, recording the next bit of covertly communicated information.

In a *covert timing channel*, one process relays information to another by modulating its use of system resources. Adam could tie up a shared resource (such as a communications bus). Bob tries to access the resource and, if successful, records it as a zero bit (no wait). Otherwise, he records a one and waits a predetermined amount of time. Adam, meanwhile, is encoding his covert message by selectively tying up or freeing the shared resource. Think of this as a type of Morse code, but using some type of system resource.

Brewer and Nash Model

The *Brewer and Nash model*, also called the *Chinese Wall model*, states that a subject can write to an object if, and only if, the subject cannot read another object that is in a different dataset. It was created to provide access controls that can change dynamically depending upon a user's previous actions. The main goal of the model is to protect against conflicts of interest by users' access attempts. Suppose Maria is a broker at an investment firm that also provides other services to Acme Corporation. If Maria were able to access Acme information from the other service areas, she could learn of a phenomenal earnings report that is about to be released. Armed with that information, she could encourage her clients to buy shares of Acme, confident that the price will go up shortly. The Brewer and Nash Model is designed to mitigate the risk of this situation happening.

Graham-Denning Model

Remember that these are all models, so they are not very specific in nature. Each individual vendor must decide how it is going to actually meet the rules outlined in the chosen model. Bell-LaPadula and Biba do not define how the security and integrity levels are defined and modified, nor do they provide a way to delegate or transfer access rights. The *Graham-Denning model* addresses some of these issues and defines a set of basic rights in terms of commands that a specific subject can execute on an object. This model has eight primitive protection rights, or rules of how these types of functionalities should take place securely:

- How to securely create an object
- How to securely create a subject
- How to securely delete an object
- How to securely delete a subject
- How to securely provide the read access right
- How to securely provide the grant access right
- How to securely provide the delete access right
- How to securely provide transfer access rights

These functionalities may sound insignificant, but when you're building a secure system, they are critical. If a software developer does not integrate these functionalities in a secure manner, they can be compromised by an attacker and the whole system can be at risk.

Harrison-Ruzzo-Ullman Model

The *Harrison-Ruzzo-Ullman (HRU)* model deals with access rights of subjects and the integrity of those rights. A subject can carry out only a finite set of operations on an object. Since security loves simplicity, it is easier for a system to allow or disallow authorization of operations if one command is restricted to a single operation. For example, if a subject sent command X that only requires the operation of Y, this is pretty straightforward and the system can allow or disallow this operation to take place. But if a subject

Security Models Recap

All of these models can seem confusing. Most people are not familiar with all of them, which can make the information even harder to absorb. The following are the core concepts of the different models.

Bell-LaPadula Model This is the first mathematical model of a multilevel security policy that defines the concept of a secure state and necessary modes of access. It ensures that information only flows in a manner that does not violate the system policy and is confidentiality focused.

- **The simple security rule** A subject cannot read data within an object that resides at a higher security level (no read up).
- **The *-property rule** A subject cannot write to an object at a lower security level (no write down).
- **The strong star property rule** For a subject to be able to read and write to an object, the subject's clearance and the object's classification must be equal.

Biba Model A model that describes a set of access control rules designed to ensure data integrity.

- **The simple integrity axiom** A subject cannot read data at a lower integrity level (no read down).
- **The *-integrity axiom** A subject cannot modify an object at a higher integrity level (no write up).

Clark-Wilson Model This integrity model is implemented to protect the integrity of data and to ensure that properly formatted transactions take place. It addresses all three goals of integrity:

- Subjects can access objects only through authorized programs (access triple).
- Separation of duties is enforced.
- Auditing is required.

Noninterference Model This formal multilevel security model states that commands and activities performed at one security level should not be seen by, or affect, subjects or objects at a different security level.

Brewer and Nash Model This model allows for dynamically changing access controls that protect against conflicts of interest. Also known as the Chinese Wall model.

Graham-Denning Model This model shows how subjects and objects should be created and deleted. It also addresses how to assign specific access rights.

Harrison-Ruzzo-Ullman Model This model shows how a finite set of procedures can be available to edit the access rights of a subject.

sent a command M and to fulfill that command, operations N, B, W, and P have to be carried out, then there is much more complexity for the system to decide if this command should be authorized.

Also the integrity of the access rights needs to be ensured; thus, in this example, if one operation cannot be processed properly, the whole command fails. So although it is easy to dictate that subject A can only read object B, it is not always so easy to ensure each and every function supports this high-level statement. The HRU model is used by software designers to ensure that no unforeseen vulnerability is introduced and the stated access control goals are achieved.

Security Requirements

Whether we are building enterprise security architectures or software systems or anything in between, we always want to start from a set of requirements. These are what ultimately tell us whether or not the thing we built meets our needs. Security requirements should flow out of the organizational risk management processes, be informed by threat models, and be grounded in the principles and models we discussed earlier in this chapter. These requirements are then addressed within the frameworks we discussed in Chapter 4, and the whole thing is then assessed over time using a maturity model like the Capability Maturity Model Integration (CMMI), also discussed in Chapter 4.

One of the key tasks in addressing security requirements in our enterprise architectures is selecting the right controls for each requirement and then implementing, documenting, and verifying them. The exact process will vary depending on the framework you choose, but you may want to review the CRM example we discussed in Chapter 4 that applies NIST SP 800-53 (see Table 4-2 in particular).

Security Capabilities of Information Systems

Satisfying security requirements has become easier over the years as most vendors now incorporate advanced security capabilities into their products, particularly physical ones. After all, we can go to great lengths to ensure that the software we develop is secure, to run it on operating systems that have been hardened by a myriad of security controls, and to monitor everything using advanced security tools, but if the physical devices on which these systems run are untrustworthy, then all our efforts are for naught. In the sections that follow, we discuss a variety of hardware-based capabilities of many information systems that you should know.

Trusted Platform Module

A *Trusted Platform Module (TPM)* is a hardware component installed on the motherboard of modern computers that is dedicated to carrying out security functions involving the storage of cryptographic keys and digital certificates, symmetric and asymmetric encryption, and hashing. The TPM was devised by the Trusted Computing Group (TCG), an organization that promotes open standards to help strengthen computing platforms against security weaknesses and attacks.

The essence of a TPM lies in a protected and encapsulated microcontroller security chip that provides a safe haven for storing and processing critical security data such as keys, passwords, and digital certificates. The use of a dedicated and encoded hardware-based platform drastically improves the root of trust of the computing system, while allowing for a vastly superior implementation and integration of security features. The introduction of TPM has made it much harder to access information on computing devices without proper authorization and allows for effective detection of malicious configuration changes to a computing platform.

TPM Uses

The most common usage scenario of a TPM is to bind a hard disk drive, where the content of a given hard disk drive is affixed with a particular computing system. The content of the hard disk drive is encrypted, and the decryption key is stored away in a TPM chip. To ensure safe storage of the decryption key, it is further "wrapped" with another encryption key. Binding a hard disk drive makes its content basically inaccessible to other systems, and any attempt to retrieve the drive's content by attaching it to another system will be very difficult. However, in the event of a TPM chip's failure, the hard drive's content will be rendered useless, unless a backup of the key has been escrowed.

Another application of a TPM is *sealing* a system's state to a particular hardware and software configuration. Sealing a computing system through TPM is used to deter any attempts to tamper with a system's configurations. In practice, this is similar to how hashes are used to verify the integrity of files shared over the Internet (or any other untrusted medium). Sealing a system is fairly straightforward. The TPM generates hash values based on the system's configuration files and stores them in its memory. A sealed system will be activated only after the TPM verifies the integrity of the system's configuration by comparing it with the original "sealing" value.

A TPM is essentially a securely designed microcontroller with added modules to perform cryptographic functions. These modules allow for accelerated storage and processing of cryptographic keys, hash values, and pseudonumber sequences. A TPM's internal storage is based on random access memory (RAM), which retains its information when power is turned off and is therefore termed *nonvolatile RAM (NVRAM)*. A TPM's internal memory is divided into two different segments: persistent (static) and versatile (dynamic) memory modules, as shown in Figure 9-4.

Persistent Memory Two kinds of keys are present in the static memory:

- **Endorsement Key (EK)** A public/private key pair that is installed in the TPM at the time of manufacture and cannot be modified. The private key is always present inside the TPM, while the public key is used to verify the authenticity of the TPM itself. The EK, installed in the TPM, is unique to that TPM and its platform.
- **Storage Root Key (SRK)** The master wrapping key used to secure the keys stored in the TPM.

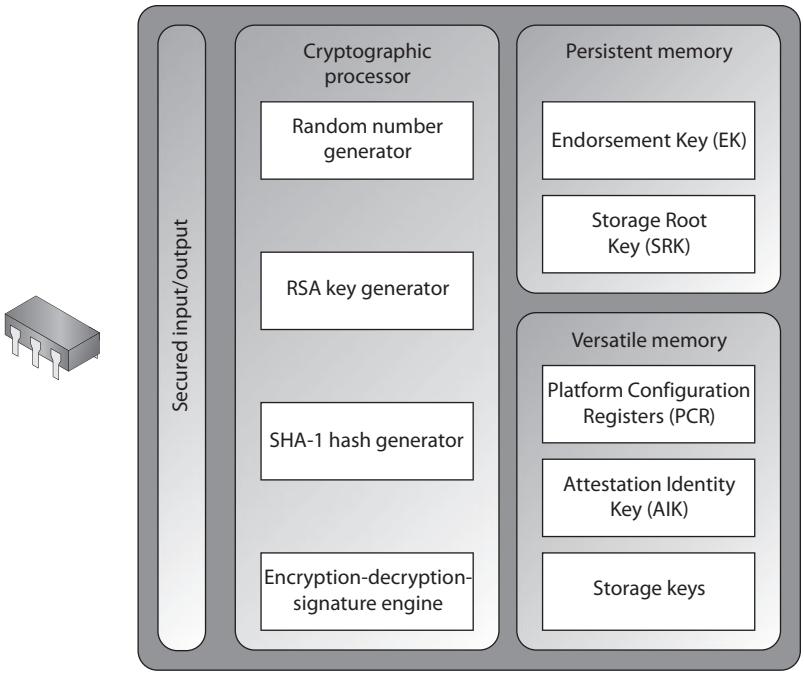


Figure 9-4 Functional components of a Trusted Platform Module

Versatile Memory Three kinds of keys (or values) are present in the versatile memory:

- **Platform Configuration Registers (PCRs)** Used to store cryptographic hashes of data used for TPM's sealing functionality.
- **Attestation Identity Keys (AIKs)** Used for the attestation of the TPM chip itself to service providers. The AIK is linked to the TPM's identity at the time of development, which in turn is linked to the TPM's EK. Therefore, the AIK ensures the integrity of the EK.
- **Storage keys** Used to encrypt the storage media of the computer system.

Hardware Security Module

Whereas a TPM is a microchip installed on a motherboard, a *hardware security module (HSM)* is a removable expansion card or external device that can generate, store, and manage cryptographic keys. HSMs are commonly used to improve encryption/decryption performance by offloading these functions to a specialized module, thus freeing up the general-purpose microprocessor to take care of, well, general-purpose tasks. HSMs have become critical components for data confidentiality and integrity in digital business transactions. The U.S. Federal Information Processing Standard (FIPS) 140-2 is perhaps the most widely recognized standard for evaluating the security of an HSM. This evaluation is important, because so much digital commerce nowadays relies on protections provided by HSM.

As with so many other cybersecurity technologies, the line between TPMs and HSMs gets blurred. TPMs are typically soldered onto a motherboard, but they can be added through a header. HSMs are almost always external devices, but you will occasionally see them as Peripheral Component Interconnect (PCI) cards. In general, however, TPMs are permanently mounted and used for hardware-based assurance and key storage, while HSMs are removable (or altogether external) and are used for both hardware-accelerated cryptography and key storage.

Self-Encrypting Drive

Full-disk encryption (FDE) refers to approaches used to encrypt the entirety of data at rest on a disk drive. This can be accomplished in either software or hardware. A *self-encrypting drive (SED)* is a hardware-based approach to FDE in which a cryptographic module is integrated with the storage media into one package. Typically, this module is built right into the disk controller chip. Most SEDs are built in accordance with the TCG Opal 2.0 standard specification.

NOTE Although SEDs can use onboard TPMs, that is not the norm.

The data stored in an SED is encrypted using symmetric key encryption, and it's decrypted dynamically whenever the device is read. A write operation works the other way—that is, the plaintext data arrives at the drive and is encrypted automatically before being stored on disk. Because the SED has its own hardware-based encryption engine, it tends to be faster than software-based approaches.

Encryption typically uses the Advanced Encryption Standard (AES) and a 128- or 256-bit key. This secret key is stored in nonvolatile memory within the cryptographic module and is itself encrypted with a password chosen by the user. If the user changes the password, the same secret key is encrypted with the new password, which means the whole disk doesn't have to be decrypted and then re-encrypted. If ever there is a need to securely wipe the contents of the SED, the cryptographic module is simply told to generate a new secret key. Since the drive contents were encrypted with the previous (now overwritten) key, that data is effectively wiped. As you can imagine, wiping an SED is almost instantaneous.

Bus Encryption

While the self-encrypting drive protects the data as it rests on the drive, it decrypts the data prior to transferring it to memory for use. This means that an attacker has three opportunities to access the plaintext data: on the external bus connecting the drive to the motherboard (which is sometimes an external cable), in memory, or on the bus between memory and the CPU. What if we moved the cryptographic module from the disk controller to the CPU? This would make it impossible for attackers to access the plaintext data outside the CPU itself, making their job that much harder.

Bus encryption means data and instructions are encrypted prior to being put on the internal bus, which means they are also encrypted everywhere else except when data is being processed. This approach requires a specialized chip, a *cryptoprocessor*, that combines traditional CPU features with a cryptographic module and specially protected memory for keys. If that sounds a lot like a TPM, it's because it usually is.

You won't see bus encryption in general-purpose computers, mostly because the cryptoprocessors are both more expensive and less capable (performance-wise) than regular CPUs. However, bus encryption is a common approach to protecting highly sensitive systems such as automated teller machines (ATMs), satellite television boxes, and military weapon systems. Bus encryption is also widely used for smart cards. All these examples are specialized systems that don't require a lot of processing power but do require a lot of protection from any attacker who gets his or her hands on them.

Secure Processing

By way of review, data can exist in one of three states: at rest, in transit, or in use. While we've seen how encryption can help us protect data in the first two states, it becomes a bit trickier when it is in use. The reason is that processors almost always need unencrypted code and data to work on.

There are three common ways to protect data while it's in use. The first is to create a specially protected part of the computer in which only trusted applications can run with little or no interaction with each other or those outside the trusted environment. Another approach is to build extensions into the processors that enable them to create miniature protected environments for each application (instead of putting them all together in one trusted environment). Finally, we can just write applications that temporarily lock the processor and/or other resources to ensure nobody interferes with them until they're done with a specific task. Let's take a look at these approaches in order.

Trusted Execution Environment

A *trusted execution environment (TEE)* is a software environment in which special applications and resources (such as files) have undergone rigorous checks to ensure that they are trustworthy and remain protected. Some TEEs, particularly those used in Apple products, are called *secure enclaves*, but the two terms are otherwise interchangeable. TEEs exist in parallel with untrusted *rich execution environments (REEs)* on the same platform, as shown in Figure 9-5. TEEs are widely used in mobile devices and increasingly included in embedded and IoT devices as well, to ensure that certain critical applications and their data have guaranteed confidentiality, integrity, and availability. TEEs are also starting to show up in other places, such as microservices and cloud services, where hardware resources are widely shared.

A TEE works by creating a trust boundary around itself and strictly controlling the way in which the untrusted REE interacts with the trusted applications. The TEE typically has its own hardware resources (such as a processor core, memory, and persistent storage) that are unavailable to the REE. It also runs its own trusted OS that is separate from and independent of the one in the REE. The two environments interact through a

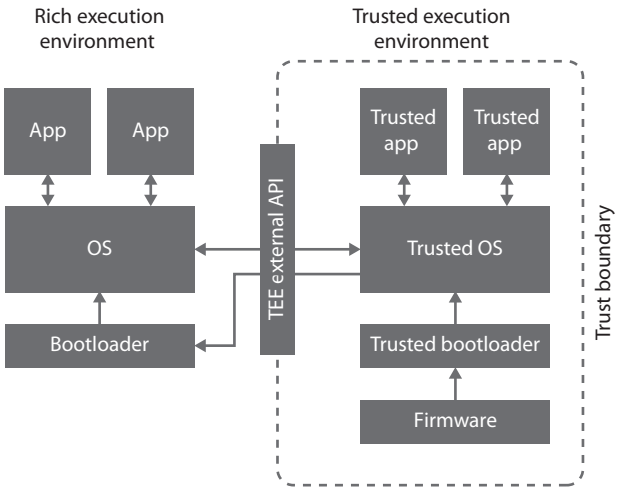


Figure 9-5
A typical TEE and its related REE

restricted external application programming interface (API) that enables the rich OS to call a limited set of services provided by the REE.

NOTE The term "secure enclave" is most commonly associated with Apple products such as the iPhone, but it is otherwise equivalent to the term "trusted execution environment."

So, how do TPMs, HSMs, and TEEs differ from each other? A TPM is usually a system on a chip (SoC) soldered onto the motherboard to provide limited cryptographic functions. An HSM is a big TPM that plugs into a computer system to provide these functions at a much larger scale. A TEE can perform the functions of a TPM, but, unlike both the TPM and HSM, it is specifically designed to run trusted applications that may have nothing to do with cryptography.

Trusted execution starts with a secure boot, in which the firmware verifies the integrity of the trusted OS bootloader before executing it. In fact, every executable and driver in the TEE is verified to the hardware root of trust and restricted to its own assigned resources. Only specific applications that have undergone rigorous security assessments at the hands of trusted parties are deployed in the TEE by the device manufacturer. This enables trusted applications such as cryptography, identity, and payment systems to enjoy high levels of protection that would otherwise be impossible to attain.

EXAM TIP TEEs (and, by extension, secure enclaves) do not implement hardware roots of trust because they are implemented in software. However, TEEs typically rely on an underlying root of trust provided by a TPM on the device.