Not as easy as you may think.

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

1. Choose two random large prime numbers, p and q.
2. Generate the product of these numbers: n = pq. n is used as the modulus.
3. Choose a random integer e (the public key) that is greater than 1 but less than (p – 1)(q – 1). Make sure that e and (p – 1)(q – 1) are relatively prime.

5. The public key = (n, e).
6. The private key = (n, d).
7. The original prime numbers p and q are discarded securely.

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

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

c = me mod n

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

m = cd mod n

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

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

One-Way Functions A one-way function is a mathematical function that is easier to compute in one direction than in the opposite direction. An analogy of this is when you

4. Compute the corresponding private key, d, such that de – 1 is a multiple of (p – 1)(q – 1).

drop a glass on the floor. Although dropping a glass on the floor is easy, putting all the pieces back together again to reconstruct the original glass is next to impossible. This concept is similar to how a one-way function is used in cryptography, which is what the RSA algorithm, and all other asymmetric algorithms, are based upon.
The easy direction of computation in the one-way function that is used in the RSA algorithm is the process of multiplying two large prime numbers. If I asked you to multiply two prime numbers, say 79 and 73, it would take you just a few seconds to punch that into a calculator and come up with the product (5,767). Easy. Now, suppose I asked you to find out which two numbers, when multiplied together, produce the value 5,767. This is called factoring and, when the factors involved are large prime numbers, it turns out to be a really hard problem. This difficulty in factoring the product of large prime numbers is what provides security for RSA key pairs.
As explained earlier in this chapter, work factor is the amount of time and resources it would take for someone to break an encryption method. In asymmetric algorithms, the work factor relates to the difference in time and effort that carrying out a one-way function in the easy direction takes compared to carrying out a one-way function in the hard direction. In most cases, the larger the key size, the longer it would take for the adversary to carry out the one-way function in the hard direction (decrypt a message).
The crux of this section is that all asymmetric algorithms provide security by using mathematical equations that are easy to perform in one direction and next to impossible to perform in the other direction. The “hard” direction is based on a “hard” mathematical problem. RSA’s hard mathematical problem requires factoring large numbers into their original prime numbers.

Elliptic Curve Cryptography
The one-way function in RSA has survived cryptanalysis for over four decades but eventually will be cracked simply because we keep building computers that are faster. Sooner

or later, computers will be able to factor the products of ever-larger prime numbers in reasonable times, at which point we would need to either ditch RSA or figure out how to use larger keys. Anticipating this eventuality, cryptographers found an even better trapdoor in elliptic curves. An elliptic curve, such as the one shown in Figure 8-12, is the set of points that satisfies a specific mathematical equation such as this one:

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

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

Figure 8-12
Elliptic curve

Q

P

R=P+Q

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

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

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

Quantum Cryptography

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

Quantum key distribution (QKD) is a system that generates and securely distributes encryption keys of any length between two parties. Though we could, in principle, use anything that obeys the principles of quantum mechanics, photons (the tiny particles that make up light) are the most convenient particles to use for QKD. It turns out photons are polarized or spin in ways that can be described as vertical, horizontal,

diagonal left
(–45o), and diagonal right (45o). If we put a polarized filter in front of a detector, any
photon that makes it to that detector will have the polarization of its filter. Two types
of filters are commonly used in QKD. The first is rectilinear and allows vertically and
horizontally polarized photons through. The other is a (you guessed it) diagonal filter,
which allows both diagonally left and diagonally right polarized photons through. It
is important to note that the only way to measure the polarization on a photon is to
essentially destroy it: either it is blocked by the filter if the polarizations are different or
it is absorbed by the sensor if it makes it through.

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

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

They now have a shared secret key through this process, which is known as key distillation.

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

Figure 8-13
Key distillation
between Alice

and Bob

Alice’s bit

0

1

1

0

1

0

0

1

Alice’s basis

+

+

×

+

×

×

×

+

+

×

×

×

+

×

+

+

Alice’s polarization
Bob’s filter
Bob’s measurement
Shared secret key

0

1

0

1

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

PART III

Alice and Bob encrypt with it. Since the quantum state of photons is destroyed when they are filtered or measured, she would have to follow the same process as Bob intends to and then generate a new photon stream to forward to Bob. The catch is that Eve (just like Bob) will get 50 percent of the measurements wrong, but (unlike Bob) now has to guess what random basis was used and send these guesses to Bob. When Alice and Bob compare polarizations, they’ll note a much higher error rate than normal and be able to infer that someone was eavesdropping.
If you’re still awake and paying attention, you may be wondering, “Why use the polarization filters in the first place? Why not just capture the photon and see how it’s spinning?” The answer gets complicated in a hurry, but the short version is that polarization is a random quantum state until you pass the photon through the filter and

force the photon to “decide” between the two polarizations. Eve cannot just re-create the photon’s quantum state like she would do with conventional data. Keep in mind that quantum mechanics are pretty weird but lead to unconditional security of the shared key.
Now that we have a basic idea of how QKD works, let’s think back to our discussion of the only perfect and unbreakable cryptosystem: the one-time pad. You may recall that it has five major requirements that largely make it impractical. We list these here and show how QKD addresses each of them rather nicely:

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

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

Asymmetric and Symmetric Algorithms Used Together
Asymmetric and symmetric cryptosystems are used together very frequently. In this hybrid approach, the two technologies are used in a complementary manner, with each performing a different function. A symmetric algorithm creates keys used for encrypting bulk

data, and an asymmetric algorithm creates keys used for automated key distribution. Each algorithm has its pros and cons, so using them together can be the best of both worlds.
When a symmetric key is used for bulk data encryption, this key is used to encrypt the message you want to send. When your friend gets the message you encrypted, you want him to be able to decrypt it, so you need to send him the necessary symmetric key to use to decrypt the message. You do not want this key to travel unprotected, because if the message were intercepted and the key were not protected, an eavesdropper could intercept the message that contains the necessary key to decrypt your message and read your information.
If the symmetric key needed to decrypt your message is not protected, there is no use in encrypting the message in the first place. So you should use an asymmetric algorithm to encrypt the symmetric key, as depicted in Figure 8-14. Why use the symmetric key on the message and the asymmetric key on the symmetric key? As stated earlier, the asymmetric algorithm takes longer because the math is more complex. Because your message is most likely going to be longer than the length of the key, you use the faster algorithm (symmetric) on the message and the slower algorithm (asymmetric) on the key.
How does this actually work? Let's say Bill is sending Paul a message that Bill wants only Paul to be able to read. Bill encrypts his message with a secret key, so now Bill has ciphertext and a symmetric key. The key needs to be protected, so Bill encrypts the symmetric key with an asymmetric key. Remember that asymmetric algorithms use private and public keys, so Bill will encrypt the symmetric key with Paul's public key. Now Bill has ciphertext from the message and ciphertext from the symmetric key. Why did Bill encrypt the symmetric key with Paul's public key instead of his own private key? Because if Bill

Message and key will be sent to receiver.

Receiver decrypts
and retrieves the
symmetric key,
then uses this
symmetric key to

decrypt the
message.

Symmetric key
encrypted with an
asymmetric key
Message encrypted
with symmetric key

PART III

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

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

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

Symmetric
key
Decrypts with Paul's
private key

Paul reads Bill's
message.

Symmetric
key
Encrypted with Paul's
public key

Message
Message

Decrypts with
symmetric key

Encrypted with the
symmetric key

Bill

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

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

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

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

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

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

Now check your answers:

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

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

Session Keys

1.
2.
3.

Session key
Encrypted with
Tanya’s public
key

Tanya
4.

Lance

5.

Session key

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

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

A session key is a single-use symmetric key that is used to encrypt messages between two
users during a communication session. A session key is no different from the symmetric
key described in the previous section, but it is only good for one communication session
between users.
If Tanya has a symmetric key she uses to always encrypt messages between Lance and herself, then this symmetric key would not be regenerated or changed. They would

use the same key every time they communicated using encryption. However, using the same key repeatedly increases the chances of the key being captured and the secure communication being compromised. If, on the other hand, a new symmetric key were generated each time Lance and Tanya wanted to communicate, as shown in Figure 8-15, it would be used only during their one dialogue and then destroyed. If they wanted to communicate an hour later, a new session key would be created and shared.

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

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

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

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

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

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

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

Integrity

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

Hashing Functions

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

Various Hashing Algorithms
As stated earlier, the goal of using a one-way hash function is to provide a fingerprint of the message. If two different messages produce the same hash value, it would be easier for an attacker to break that security mechanism because patterns would be revealed.
A strong one-hash function should not provide the same hash value for two or more different messages. If a hashing algorithm takes steps to ensure it does not create the same hash value for two or more messages, it is said to be collision free.

A one-way hash is a function that takes a variable-length string (a message) and produces a fixed-length value called a hash value. For example, if Kevin wants to send a message to Maureen and he wants to ensure the message does not get altered in an unauthorized fashion while it is being transmitted, he would calculate a hash value for the message and append it to the message itself. When Maureen receives the message, she performs the same hashing function Kevin used and then compares her result with the hash value sent with the message. If the two values are the same, Maureen can be sure the message was not altered during transmission. If the two values are different, Maureen knows the message was altered, either intentionally or unintentionally, and she discards the message.
The hashing algorithm is not a secret—it is publicly known. The secrecy of the oneway hashing function is its “one-wayness.” The function is run in only one direction, not the other direction. This is different from the one-way function used in public key cryptography, in which security is provided based on the fact that, without knowing a trapdoor, it is very hard to perform the one-way function backward on a message and come up with readable plaintext. However, one-way hash functions are never used in reverse; they create a hash value and call it a day. The receiver does not attempt to reverse the process at the other end, but instead runs the same hashing function one way and compares the two results.

Algorithm

Description

Message Digest 5 (MD5) algorithm

Produces a 128-bit hash value. More complex than MD4.

Secure Hash Algorithm (SHA)

Produces a 160-bit hash value. Used with Digital Signature Algorithm (DSA).

SHA-1, SHA-256, SHA-384, SHA-512

Updated versions of SHA. SHA-1 produces a 160-bit hash value, SHA-256 creates a 256-bit value, and so on.

Table 8-2 Various Hashing Algorithms Available

Strong cryptographic hash functions have the following characteristics:

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

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

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

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

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

Attacks Against One-Way Hash Functions

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

How many people must be in the same room for the chance to be greater than even that another person has the same birthday as you?
Answer: 253

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

Why do we care? The birthday paradox can apply to cryptography as well. Since any random set of 23 people most likely (at least a 50 percent chance) includes two people who share a birthday, by extension, if a hashing algorithm generates a message digest of 60 bits, there is a high likelihood that an adversary can find a collision using only 230 inputs.

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

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

How Would a Birthday Attack Take Place?

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

One month after Sue and Joe get married, Sue carries out some devious activity behind Joe’s back. She makes a copy of the message digest value without anyone knowing. Then she makes a new contract that states that if Joe and Sue get a divorce, Sue owns both her

How many people must be in the same room for the chance to be greater than even that at least two people share the same birthday?
Answer: 23

own original belongings and Joe’s original belongings. Sue hashes this new contract and compares the new message digest value with the message digest value that correlates with the contract. They don’t match. So Sue tweaks her contract ever so slightly and creates another message digest value and compares them. She continues to tweak her contract until she forces a collision, meaning her contract creates the same message digest value as the original contract. Sue then changes out the original contract with her new contract and quickly divorces Joe. When Sue goes to collect Joe’s belongings and he objects, she shows him that no modification could have taken place on the original document because it still hashes out to the same message digest. Sue then moves to an island.

Hash algorithms usually use message digest sizes (the value of n) that are large enough to make collisions difficult to accomplish, but they are still possible. An algorithm that has 256-bit output, like SHA-256, may require approximately 2128 computations to break. This means there is a less than 1 in 2128 chance that someone could carry out a successful birthday attack.

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

Message Integrity Verification

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

Message Digest

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

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

Message

3
Hashing
function
10110
00101
10110
00101

2

10110
00101

?
=

10110
00101

4

5
Sender

Receiver

Figure 8-16 Verifying message integrity with a message digest

Message Authentication Code

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

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

Figure 8-17 shows the following steps to use an HMAC:

1. The sender writes a message.
2. The sender concatenates a shared secret key with the message and puts them

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

Message
digest

Secret key Message

1

3

10110
00101
Message
authentication
code (MAC)

10110
00101

2

10110
00101

?
=

10110
00101

4

5
Sender

Receiver

Figure 8-17 Verifying message integrity with a message digest

Now, when we say that the message is concatenated with a symmetric key, we don't mean a symmetric key is used to encrypt the message. The message is not encrypted in an HMAC function, so there is no confidentiality being provided. Think about throwing a message in a bowl and then throwing a symmetric key in the same bowl. If you dump the contents of the bowl into a hashing algorithm, the result will be a MAC value.

Digital Signatures

A MAC can ensure that a message has not been altered, but it cannot ensure that it comes from the entity that claims to be its source. This is because MACs use symmetric keys, which are shared. If there was an insider threat who had access to this shared key, that person could modify messages in a way that could not be easily detected. If we wanted to protect against this threat, we would want to ensure the integrity verification mechanism is tied to a specific individual, which is where public key encryption comes in handy.

A digital signature is a hash value that has been encrypted with the sender's private key. Since this hash can be decrypted by anyone who has the corresponding public key, it verifies that the message comes from the claimed sender and that it hasn't been altered.

The act of signing means encrypting the message's hash value with a private key, as shown in Figure 8-18.

Continuing our example from the previous section, if Cheryl wants to ensure that the message she sends to Scott is not modified and she wants him to be sure it came only from her, she can digitally sign the message. This means that a one-way hashing function would be run on the message, and then Cheryl would encrypt that hash value with her private key.

Message

Calculated hash
value of ABC
Encrypted with
the private key

Message

ABC

Message

ABC

Calculates hash value
on received message =
ABC

Decrypts sent hash
value with public key =
ABC

Figure 8-18

User reads message.

Both the calculated
and sent hash
values are the
same, thus the
message was not
modified during
transmission.

Creating a digital signature for a message

When Scott receives the message, he performs the hashing function on the message and comes up with his own hash value. Then he decrypts the sent hash value (digital signature) with Cheryl's public key. He then compares the two values, and if they are the same, he can be sure the message was not altered during transmission. He is also sure the message came from Cheryl because the value was encrypted with her private key.
The hashing function ensures the integrity of the message, and the signing of the hash value provides authentication and nonrepudiation. The act of signing just means the value was encrypted with a private key.
Because digital signatures are so important in proving who sent which messages, the U.S. federal government decided to establish standards pertaining to their functions and acceptable use. In 1991, NIST proposed a federal standard called the Digital Signature Standard (DSS). It was developed for federal departments and agencies, but most vendors also designed their products to meet these specifications. The federal government requires its departments to use DSA, RSA, or the elliptic curve digital signature algorithm (ECDSA) and SHA-256. SHA creates a 256-bit message digest output, which is then inputted into one of the three mentioned digital signature algorithms. SHA is

used to
ensure the integrity of the message, and the other algorithms are used to
digitally sign

PART III

Encrypted
message
digest

the message. This is an example of how two different algorithms are combined to provide
the right combination of security services.
RSA and DSA are the best-known and most widely used digital signature algorithms.
DSA was developed by the NSA. Unlike RSA, DSA can be used only for digital
signatures, and DSA is slower than RSA in signature verification. RSA can be used for
digital signatures, encryption, and secure distribution of symmetric keys.

Digests, HMACs, and Digital Signatures—Oh My!
MACs and hashing processes can be confusing. The following table simplifies the
differences between them.
Security Service
Provided

Function

Steps

Hash

1. Sender puts a message through a hashing
algorithm and generates a message digest (MD)
value.
2. Sender sends message and MD value to receiver.
3. Receiver runs just the message through
the same hashing algorithm and creates an
independent MD value.
4. Receiver compares both MD values. If they are
the same, the message was not modified.

Integrity; not
confidentiality or
authentication.
Can detect only
unintentional
modifications.

HMAC

1. Sender concatenates a message and secret key
and puts the result through a hashing algorithm.
This creates a MAC value.
2. Sender appends the MAC value to the message
and sends it to the receiver.
3. The receiver takes just the message and
concatenates it with her own symmetric key.
This results in an independent MAC value.
4. The receiver compares the two MAC values.
If they are identical, the receiver knows the
message was not modified.

Integrity and
data origin
authentication;
confidentiality is not
provided.

Digital
signature

1. The sender computes the hash of the message
and encrypts it with her private key.
2. The sender appends the encrypted message
digest to the message and sends it to the
receiver.
3. The receiver computes the hash of the received
message.
4. The receiver decrypts the received message
digest using the sender's public key.
5. The receiver compares the two digests. If they
are identical, the receiver knows the message
was not modified and knows from which system
it came.

Integrity, sender
authentication, and
nonrepudiation;
confidentiality is not
provided.

Public Key Infrastructure

Digital Certificates
Recall that, in asymmetric key cryptography, we keep a private key secret and widely
share its corresponding public key. This allows anyone to send us an encrypted
message that only we (or whoever is holding the private key) can decrypt. Now, suppose
you receive a message from your boss asking you to send some sensitive

information encrypted with her public key, which she attaches to the message. How can you be sure it really is her? After all, anybody could generate a key pair and send you a public key claiming to be hers.

A digital certificate is the mechanism used to associate a public key with a collection of components in a manner that is sufficient to uniquely identify the claimed owner. The most commonly used standard for digital certificates is the International Telecommunications Union's X.509, which dictates the different fields used in the certificate and the valid values that can populate those fields. The certificate includes the serial number, version number, identity information, algorithm information, lifetime dates, and the signature of the issuing authority, as shown in Figure 8-19.

Note that the certificate specifies the subject, which is the owner of the certificate and holder of the corresponding private key, as well as an issuer, which is the entity that is certifying that the subject is who they claim to be. The issuer attaches a digital signature to the certificate to prove that it was issued by that entity and hasn't been altered by others. There is nothing keeping anyone from issuing a self-signed certificate, in which the subject and issuer can be one and the same. While this might be allowed, it should be very suspicious when dealing with external entities. For example, if your bank presents to you a self-signed certificate, you should not trust it at all. Instead, we need a reputable third party to verify subjects' identities and issue their certificates.

PART III

Now that you understand the main approaches to modern cryptography, let's see how they come together to provide an infrastructure that can help us protect our organizations in practical ways. A public key infrastructure (PKI) consists of programs, data formats, procedures, communication protocols, security policies, and cryptosystems working in a comprehensive manner to enable a wide range of dispersed people to communicate in a secure and predictable fashion. In other words, a PKI establishes and maintains a high level of trust within an environment. It can provide confidentiality, integrity, nonrepudiation, authentication, and even authorization. As we will see shortly, it is a hybrid system of symmetric and asymmetric cryptosystems, which were

discussed in earlier sections.
There is a difference between public key cryptography and PKI. Public key cryptography is another name for asymmetric algorithms, while PKI (as the name states) is an infrastructure that is partly built on public key cryptography. The central concept in PKI is the digital certificate, but it also requires certificate authorities, registration authorities, and effective key management.

Serial
Signature
number

Version

Identifies
the version
of the
certificate

Issuer

Validity

Algorithm ID
used to
sign the
certificate

Subject
public
key info

Issuer
unique ID

Subject
unique ID

Public key
of owner

Name of
certificate
issuer

Validity
dates

Extensions

Optional
extensions
ID of
subject

Name of
owner

Unique
number
for the
certificate

Figure 8-19

Subject

ID of
issuing CA

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.

Certificate
authority

Dave and Kevin trust each other indirectly.

Kevin trusts the CA.

Dave trusts the CA.

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.

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
Directory

Figure 8-20
CA and user
relationships

John requests
Diane’s public key.

CA

Diane’s public
key is sent.

Diane validates
John’s public key.

Session key is encrypted
with Diane’s public key.

John

Diane

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.

• 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.

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,

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.

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.

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

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.

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.

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.

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.

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.

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

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.

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 bestknown 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.

1. Login
2. List files

LSASS Memory:

1. Auth

Username: user1
NTLM: a3d747dhdh…

User1 (local admin)

Remote
login

Username: IT_Admin
NTLM: b7ars7r9773…

DC
2. Auth

File server
IT_Admin (domain admin)

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

LSASS Memory:
1. Auth
Username: user1
NTLM: a3d747dhdh…
User1 (local admin)

DC

Username: IT_Admin
NTLM: b7ars7r9773…
Remote
login

2. Auth

File server
IT_Admin (domain admin)

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

Apparent connection

1. TLS Hello

4. Cert: g00dsite.com

Figure 8-23

A web-based man-in-the-middle attack

2. TLS Hello

3. Cert: goodsite.com

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

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,

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).

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