Chapter 7
PKI and Cryptographic Applications

THE CISSP TOPICS COVERED IN THIS CHAPTER INCLUDE:

- ✓ **Domain 3:0 Security Architecture and Engineering**
  - 3.5 Assess and mitigate the vulnerabilities of security architectures, designs, and solution elements
    - 3.5.4 Cryptographic systems
  - 3.6 Select and determine cryptographic solutions
    - 3.6.1 Cryptographic life cycle (e.g., keys, algorithm selection)
    - 3.6.2 Cryptographic methods (e.g., asymmetric)
    - 3.6.3 Public key infrastructure (PKI) (e.g., quantum key distribution)
    - 3.6.4 Key management practices (e.g., rotation)
    - 3.6.5 Digital signatures and digital certificates (e.g., non-repudiation, integrity)
  - 3.7 Understand methods of cryptanalytic attacks
    - 3.7.1 Brute force
    - 3.7.2 Ciphertext only
    - 3.7.3 Known plaintext
    - 3.7.4 Frequency analysis
    - 3.7.5 Chosen ciphertext
    - 3.7.6 Implementation attacks
    - 3.7.7 Side-channel
    - 3.7.8 Fault injection
    - 3.7.9 Timing
    - 3.7.10 Man-in-the-middle (MITM)

In Chapter 6, “Cryptography and Symmetric Key Algorithms,” we introduced basic cryptography concepts and explored a variety of secret key cryptosystems. The symmetric cryptosystems discussed in that chapter offer fast, secure communication but introduce the substantial challenge of key exchange between previously unrelated parties.

This chapter explores the world of asymmetric (or public key) cryptography and the public key infrastructure (PKI) that supports secure communication between individuals who don't necessarily know each other prior to the communication. Asymmetric algorithms provide convenient key exchange mechanisms and are scalable to very large numbers of users, addressing the two most significant challenges for users of symmetric cryptosystems.

This chapter also explores several practical applications of asymmetric cryptography: securing portable devices, email, web communications, and networking. The chapter concludes with an examination of a variety of attacks malicious individuals might use to compromise weak cryptosystems.

Asymmetric Cryptography

The section “Modern Cryptography” in Chapter 6 introduced the basic principles behind both secret (symmetric) and public (asymmetric) key cryptosystems. You learned that symmetric key cryptosystems require that both communicating parties possess the same shared secret key, creating the problem of secure key distribution. You also learned that asymmetric cryptosystems avoid this hurdle by using pairs of public and private keys to facilitate secure communication without the overhead of complex key distribution systems.

In the following sections, we'll explore the concepts of public key cryptography in greater detail and look at four of the more common asymmetric cryptosystems in use today: Rivest–Shamir–Adleman (RSA), Diffie–Hellman, ElGamal, and elliptic curve cryptography (ECC). We'll also explore the emerging world of quantum cryptography.

Public and Private Keys

Recall from Chapter 6 that *public key cryptosystems* assign each user a pair of keys: a public key and a corresponding private key. As the names imply, public key cryptosystem users make their public keys freely available to anyone with whom they want to communicate. The mere possession of the public key by third parties does not introduce any weaknesses into the cryptosystem. The private key, on the other hand, is reserved for the sole use of the individual or entity who owns the key. Users should not normally share their private keys with any other cryptosystem user, outside of key escrow and recovery arrangements. Normal communication between public key cryptosystem users follows the process shown in Figure 7.1.

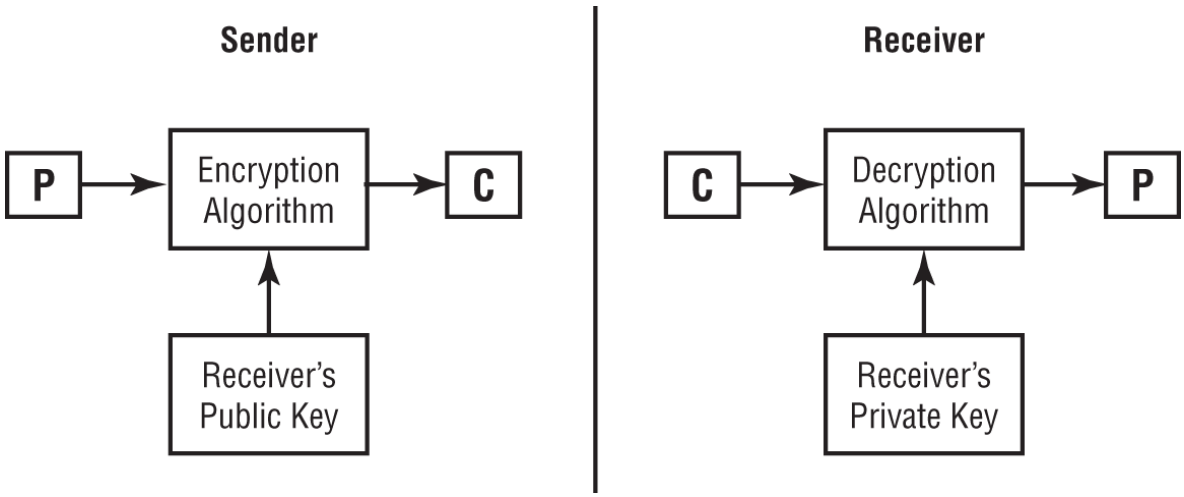

FIGURE 7.1 Asymmetric key cryptography

Notice that the process does not require the sharing of private keys. The sender encrypts the plaintext message (*P*) with the recipient's public key to create the ciphertext message (*C*). When the recipient opens the ciphertext message, they decrypt it using their private key to view the original plaintext message.

Once the sender encrypts the message with the recipient's public key, no user (including the sender) can decrypt that message without knowing the recipient's private key (the second half of the public-private key pair). This is the beauty of public key cryptography—public keys can be freely shared using insecure

communications and then used to create secure communications' channels between users previously unknown to each other.

You also learned in the previous chapter that public key cryptography entails a higher degree of computational complexity. Keys used within public key systems must be longer than those used in secret key systems to produce cryptosystems of equivalent strengths.

Because of the high computational requirements associated with public key cryptography, architects often prefer to use symmetric cryptography on anything other than short messages. Later in this chapter, you'll learn how hybrid cryptography combines the benefits of symmetric and asymmetric cryptography.

RSA

The most famous public key cryptosystem is named after its creators. In 1977, Ronald Rivest, Adi Shamir, and Leonard Adleman proposed the *RSA (Rivest-Shamir-Adleman) public key algorithm*, which remains a worldwide standard today. They patented their algorithm and formed a commercial venture known as RSA Security to develop mainstream implementations of their security technology. Today, the RSA algorithm has been released into the public domain and is widely used for secure communication.

The RSA algorithm depends on the computational difficulty inherent in factoring the product of large prime numbers. Each user of the cryptosystem generates a pair of public and private keys that are mathematically related using the algorithm described in the following steps:

1. Choose two large prime numbers (approximately 200 digits each), labeled p and q.
2. Compute the product of those two numbers: $n = p * q$.

3. Select a number, *e*, that satisfies the following two requirements:
    a. *e* is less than *n*.
    b. *e* and $(p - 1)(q - 1)$ are relatively prime—that is, the two numbers have no common factors other than 1.
4. Find a number, *d*, such that $ed = 1 \bmod ((p - 1)(q - 1))$.
5. Distribute *e* and *n* as the public key to all cryptosystem users. Keep *d* secret as the private key.

If Alice wants to send an encrypted message to Bob, she generates the ciphertext (*C*) from the plaintext (*P*) using the following formula (where *e* is Bob's public key and *n* is the product of *p* and *q* created during the key generation process):

$$C = P^e \bmod n$$

When Bob receives the message, he performs the following calculation to retrieve the plaintext message:

$$P = C^d \bmod n$$

Merkle–Hellman Knapsack

Another early asymmetric algorithm, the Merkle–Hellman Knapsack cryptosystem, was developed the year after RSA was publicized. Like RSA, it's based on the difficulty of performing factoring operations, but it relies on a component of set theory known as *super-increasing sequence* rather than on large prime numbers. Merkle–Hellman was proven ineffective when it was broken in 1984.

Importance of Key Length

The length of the cryptographic key is perhaps the most important security parameter that can be set at the discretion of the security administrator. It's important to understand the capabilities of your encryption algorithm and choose a key length that provides an appropriate level of protection. This judgment can be made by weighing the difficulty of defeating a given key length (measured in the amount of processing time required to defeat the cryptosystem) against the importance of the data.

Generally speaking, the more critical your data, the stronger the key you should use to protect that data. Timeliness of the data is also an important consideration. You must take into account the rapid growth of computing power—Moore's law suggests that computing power doubles approximately every two years. If it takes current computers one year of processing time to break your code, it will take only three months if the attempt is made with contemporary technology about four years down the road. If you expect that your data will still be sensitive at that time, you should choose a much longer cryptographic key that will remain secure well into the future.

Also, as attackers are now able to leverage cloud computing resources, they are able to more efficiently attack encrypted data. The cloud allows attackers to rent scalable computing power, including powerful graphic processing units (GPUs) on a per-hour basis, and offers significant discounts when using excess capacity during nonpeak hours. This brings powerful computing well within the reach of many attackers.

The strengths of various key lengths also vary greatly according to the cryptosystem you're using. The key lengths shown in the following table for three cryptosystems all provide equal protection because of differences in the way that the algorithms use the keying material:

Cryptosystem	Key length
Symmetric	128 bits
RSA	3,072 bits
Elliptic curve cryptography	256 bits

ElGamal

In Chapter 6, you learned how the Diffie–Hellman algorithm uses large integers and modular arithmetic to facilitate the secure exchange of secret keys over insecure communications channels. In 1985, Dr. Taher Elgamal published an article describing how the mathematical principles behind the Diffie–Hellman key exchange algorithm could be extended to support an entire public key cryptosystem used for encrypting and decrypting messages.

At the time of its release, one of the major advantages of ElGamal over the RSA algorithm was that it was released into the public domain. Elgamal did not obtain a patent on his extension of Diffie–Hellman, and it is freely available for use, unlike the then-patented RSA technology. (RSA released its algorithm into the public domain in 2000.)

However, ElGamal also has a major disadvantage—the algorithm doubles the size of any message that it encrypts. This presents a major hardship when encrypting large amounts of data that must be sent over a network.

Elliptic Curve Cryptography

The same year that Elgamal published his algorithm, two other mathematicians, Neal Koblitz from the University of Washington and Victor Miller from IBM, independently proposed the application of *elliptic curve cryptography (ECC)*.

The mathematical concepts behind elliptic curve cryptography are quite complex and well beyond the scope of this book. However, you should be generally familiar with the elliptic curve algorithm and its potential applications. If you are interested in learning the detailed mathematics behind elliptic curve cryptosystems, an excellent tutorial exists at www.certicom.com/content/certicom/en/ecc-tutorial.html.

Any elliptic curve can be defined by the following equation:

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

In this equation, *x*, *y*, *a*, and *b* are all real numbers. Each elliptic curve has a corresponding *elliptic curve group* made up of the points on the elliptic curve along with the point *O*, located at infinity. Two points within the same elliptic curve group (*P* and *Q*) can be added together with an elliptic curve addition algorithm. This operation is expressed, quite simply, as follows:

$$P + Q$$

This problem can be extended to involve multiplication by assuming that *Q* is a multiple of *P*, meaning the following:

$$Q = k.P$$

Computer scientists and mathematicians believe that it is extremely hard to find the integer *k*, even if *P* and *Q* are already known. This difficult problem, known as the elliptic curve discrete logarithm problem (ECDLP), forms the basis of elliptic curve cryptography. It is widely believed that this problem is harder to solve than both the prime factorization problem that the RSA cryptosystem is based on and the standard discrete logarithm problem (DLP) utilized by Diffie–Hellman and ElGamal. This is illustrated by the data shown in the table in the sidebar "Importance of Key Length," which noted that a 3,072-bit RSA key is cryptographically equivalent to a 256-bit elliptic curve cryptosystem key.

Diffie–Hellman Key Exchange

In Chapter 6, you learned how the Diffie–Hellman algorithm is an approach to key exchange that allows two individuals to generate a shared secret key over an insecure communications channel. With knowledge of asymmetric cryptography under your belt, we can now dive a little more into the details of how this algorithm actually works, as Diffie–Hellman key exchange is an example of public key cryptography.

The beauty of this algorithm lies in the ability of two users to generate a shared secret that they both know without ever actually transmitting that secret. Hence, they may use public key cryptography to generate a shared secret key that they then use to communicate with a symmetric encryption algorithm. This is one example of an approach known as *hybrid cryptography*, which we discuss in more detail later in this chapter.

The Diffie–Hellman algorithm works by using the mathematics of prime numbers, similar to the RSA algorithm. Imagine that Richard and Sue would like to communicate over a secure, encrypted connection but they are in different places and have no shared secret key. Richard or Sue could simply create such a key, but then they would have no way to share it with each other without exposing it to eavesdropping. So, instead, they use the Diffie–Hellman algorithm, following this process:

1. Richard and Sue agree on two large numbers: p (which is a prime number) and g (which is an integer), such that $1 < g < p$.
2. Richard chooses a large random integer r and performs the following calculation:

   $$R = g^r \bmod p$$

3. Sue chooses a large random integer s and performs the following calculation:

   $$S = g^s \bmod p$$

4. Richard sends R to Sue and Sue sends S to Richard.

5. Richard then performs this calculation:

$$K = S^r \bmod p$$

6. Sue then performs this calculation:

$$K = R^s \bmod p$$

At this point, Richard and Sue both have the same value, K, and can use this for secret key communication between the two parties.

It is important to note that Diffie–Hellman is not an encryption protocol in and of itself. It is technically a key exchange protocol. However, it is commonly used to create a shared secret key for use in Transport Layer Security (TLS), where it is referred to as either DHE (Diffie-Hellman Ephemeral) or EDH (Ephemeral Diffie-Hellman. We discuss this use of Diffie–Hellman later in this chapter.

The Diffie–Hellman key exchange algorithm relies on the use of large prime numbers. The ECDHE (Elliptic Curve Diffie-Hellman Ephemeral) key exchange algorithm is a variant of this approach that uses the elliptic curve problem to perform a similar shared secret key agreement process.

Quantum Cryptography

Quantum computing is an area of advanced theoretical research in computer science and physics. The theory behind them is that we can use principles of quantum mechanics to replace the binary 1 and 0 bits of digital computing with multidimensional quantum bits known as *qubits*.

Quantum computing remains an emerging field, and currently, quantum computers are confined to theoretical research. Nobody has yet developed a practical implementation of a useful quantum computer. That said, if quantum computers do come on the scene, they have the potential to revolutionize the world of

computer science by providing the technological foundation for the most powerful computers ever developed. Those computers would quickly upend many of the principles of modern cybersecurity.

The most significant impact of quantum computing on the world of cryptography resides in the potential that quantum computers may be able to solve problems that are not possible to solve on contemporary computers. This concept is known as *quantum supremacy* and, if achieved, may be able to easily solve the factorization problems upon which many classical asymmetric encryption algorithms rely. If this occurs, it could render popular algorithms such as RSA and Diffie–Hellman insecure.

However, quantum computers may also be used to create newer, more complex cryptographic algorithms. These *quantum cryptography* systems may be more resistant to quantum attacks and could usher in a new era of cryptography. Researchers have already developed lab implementations of *quantum key distribution (QKD)*, an approach to use quantum computing to create a shared secret key between two users, similar to the goal of the Diffie–Hellman algorithm. Like quantum cryptography in general, QKD has not yet reached the stage of practical use.

Post-Quantum Cryptography

The most practical implication of quantum computing today is that cybersecurity professionals should be aware of the length of time that their information will remain sensitive. It is possible that an attacker could retain stolen copies of encrypted data for an extended period of time and then use future developments in quantum computing to decrypt that data. If the data remains sensitive at that point, the organization may suffer injury. The most important point here for security professionals is that they must be thinking today about the security of their current data in a post-quantum world.

Also, it is quite possible that the first major practical applications of quantum computing to cryptanalytic attacks may occur in secret. An intelligence agency or other group discovering a practical means to break modern cryptography would benefit most if they kept that discovery secret and used it to their own advantage. It is even possible that such discoveries have already occurred in secret.

Hash Functions

Later in this chapter, you'll learn how cryptosystems implement digital signatures to provide proof that a message originated from a particular user of the cryptosystem and to ensure that the message was not modified while in transit between the two parties. Before you can completely understand that concept, we must first explain the concept of *hash functions*. We will explore the basics of hash functions and look at several common hash functions used in modern digital signature algorithms.

Hash functions have a very simple purpose—they take a potentially long message and generate a unique output value derived from the content of the message. This value is commonly referred to as the *message digest*. Message digests can be

generated by the sender of a message and transmitted to the recipient along with the full message for two reasons.

First, the recipient can use the same hash function to recompute the message digest from the full message. They can then compare the computed message digest to the transmitted one to ensure that the message sent by the originator is the same one received by the recipient. If the message digests do not match, that means the message was somehow modified while in transit. It is important to note that the messages must be *exactly* identical for the digests to match. If the messages have even a slight difference in spacing, punctuation, or content, the message digest values will be completely different. It is not possible to tell the degree of difference between two messages by comparing the digests. Even a slight difference will generate a totally different digest value.

Second, the message digest can be used to implement a digital signature algorithm. This concept is covered in the section "Digital Signatures," later in this chapter.

In most cases, a message digest is 128 bits or larger. However, a single-digit value can be used to perform the function of parity, a low-level or single-digit checksum value used to provide a single individual point of verification. In most cases, the longer the message digest, the more reliable its verification of integrity.

According to RSA Security, there are five basic requirements for a cryptographic hash function:

- The input can be of any length.
- The output has a fixed length.
- The hash function is relatively easy to compute for any input.
- The hash function is one-way (meaning that it is extremely hard to determine the input when provided with the output). One-way functions and their usefulness in cryptography are described in Chapter 6.
- The hash function is collision resistant (meaning that it is extremely hard to find two messages that produce the same

hash value).

The bottom line is that hash functions create a value that uniquely represents the data in the original message but cannot be reversed, or “de-hashed.” Access to the hashed value does not allow someone to determine what the original message contained. Access to the recipient's message, the original message's hashed value (message digest), and the hash function's identifier allows one to verify whether the message has been changed from the original. This is done by comparing the hashed value of the original message with the hashed value of the recipient's message. If the hashes match, the hash function was run on the same input data, so the input data has not changed.

In the following sections, we'll look at some common hashing algorithms:

- Secure Hash Algorithm (SHA)
- Message Digest 5 (MD5)
- RACE Integrity Primitives Evaluation Message Digest (RIPEMD).

Hash-Based Message Authentication Code (HMAC) is also discussed later in this chapter.

In addition to SHA, MD5, RIPEMD, and HMAC, you should recognize HAVAL. Hashing Algorithm with Variable Length (HAVAL) is a modification of MD5. HAVAL uses 1,024-bit blocks and produces hash values of 128, 160, 192, 224, and 256 bits.

SHA Family

The Secure Hash Algorithm (SHA) and its successors, SHA-1, SHA-2, and SHA-3, are government standard hash functions promoted by the National Institute of Standards and Technology

(NIST) and are specified in an official government publication—the Secure Hash Standard (SHS), also known as Federal Information Processing Standards (FIPS) 180-4.

SHA-1 takes an input of virtually any length (in reality, there is an upper bound of approximately 2,097,152 terabytes on the algorithm) and produces a 160-bit message digest. The SHA-1 algorithm processes a message in 512-bit blocks. Therefore, if the message length is not a multiple of 512, the SHA algorithm pads the message with additional data until the length reaches the next highest multiple of 512.

Cryptanalytic attacks demonstrated that there are weaknesses in the SHA-1 algorithm, and therefore, NIST deprecated SHA-1 and no longer recommends its use for any purpose, including digital signatures and digital certificates. Web browsers dropped support for SHA-1 in 2017.

As a replacement, NIST announced the SHA-2 standard, which has four major variants:

- SHA-256 produces a 256-bit message digest using a 512-bit block size.
- SHA-224 uses a truncated version of the SHA-256 hash that drops 32 bits to produce a 224-bit message digest using a 512-bit block size.
- SHA-512 produces a 512-bit message digest using a 1,024-bit block size.
- SHA-384 uses a truncated version of the SHA-512 hash that drops 128 bits to produce a 384-bit digest using a 1,024-bit block size.

Although it might seem trivial, you should take the time to memorize the size of the message digests produced by each one of the hash algorithms described in this chapter.

The cryptographic community generally considers the SHA-2 algorithms secure, but they theoretically suffer from the same weakness as the SHA-1 algorithm. In 2015, the federal government announced the release of the Keccak algorithm as the SHA-3 standard. The SHA-3 family was developed to serve as a drop-in replacement for the SHA-2 family of hash functions, offering the same variants and hash lengths using a different computational algorithm. SHA-3 provides the same level of security as SHA-2, but it is slower than SHA-2, so SHA-3 is not commonly used outside of some specialized cases where the algorithm is efficiently implemented in hardware.

MD5

The Message Digest 2 (MD2) hash algorithm was developed by Ronald Rivest (the same Rivest of Rivest, Shamir, and Adleman fame) in 1989 to provide a secure hash function for 8-bit processors. In 1990, Rivest enhanced his message digest algorithm to support 32-bit processors and increase the level of security with a version called MD4.

In 1992, Rivest released the next version of his message digest algorithm, which he called MD5. It also processes 512-bit blocks of the message, but it uses four distinct rounds of computation to produce a digest of the same length as the MD2 and MD4 algorithms (128 bits). MD5 has the same padding requirements as MD4—the message length must be 64 bits less than a multiple of 512 bits.

MD5 implements additional security features that reduce the speed of message digest production significantly. Unfortunately, cryptanalytic attacks demonstrated that the MD5 protocol is subject to collisions, preventing its use for ensuring message integrity. Specifically, Arjen Lenstra and others demonstrated in 2005 that it is possible to create two digital certificates from different public keys that have the same MD5 hash.

Some tools and systems still rely on MD5, so you may see it in use today, but it is now far better to rely on more secure hashing algorithms, such as SHA-2.

RIPEMD

The RACE Integrity Primitives Evaluation Message Digest (RIPEMD) series of hash functions is an alternative to the SHA family that is used in some applications, such as Bitcoin cryptocurrency implementations. The family contains a series of increasingly sophisticated functions:

- RIPEMD produced a 128-bit digest and contained some structural flaws that rendered it insecure.
- RIPEMD-128 replaced RIPEMD, also producing a 128-bit digest, but it is also no longer considered secure.
- RIPEMD-160 is the replacement for RIPEMD-128 that remains secure today and is the most commonly used of the RIPEMD variants. It produces a 160-bit hash value.

You may also see references to RIPEMD-256 and RIPEMD-320. These functions are actually based on RIPEMD-128 and RIPEMD-160, respectively. They do not add any security; they simply create longer hash values for cases where a longer value is needed. RIPEMD-256 has the same level of insecurity as RIPEMD-128, while RIPEMD-320 has the same level of security as RIPEMD-160. This leads to the unusual-sounding situation where RIPEMD-160 is secure, but RIPEMD-256 is not.

Comparison of Hash Function Value Lengths

Table 7.1 lists well-known hashing functions and their resultant hash value lengths in bits. Earmark this page for memorization.

TABLE 7.1 Hash algorithm memorization chart

Hash function name	Hash value length (bits)
HAVAL	128, 160, 192, 224, and 256 bits
HMAC	Variable
MD5	128
SHA-1	160
SHA2-224/SHA3-224	224
SHA2-256/SHA3-256	256
SHA2-384/SHA3-384	384
SHA2-512/SHA3-512	512
RIPEMD-128	128
RIPEMD-160	160
RIPEMD-256	256 (but with equivalent security to 128)
RIPEMD-320	320 (but with equivalent security to 160)

Digital Signatures

Once you have chosen a cryptographically sound hash function and a public key cryptographic algorithm, you can use them to implement a *digital signature* system. Digital signature infrastructures have two distinct goals:

- Digitally signed messages assure the recipient that the message truly came from the claimed sender. They enforce nonrepudiation (that is, they preclude the sender from later claiming that the message is a forgery).
- Digitally signed messages assure the recipient that the message was not altered while in transit between the sender and recipient. This protects against both malicious modification (a third party altering the meaning of the message) and unintentional modification (because of faults

in the communications process, such as electrical, optical, or radio frequency [RF] interference).

Digital signature algorithms rely on a combination of the two major concepts already covered in this chapter—public key cryptography and hashing functions.

If Alice wants to digitally sign a message she's sending to Bob, she performs the following actions:

1. Alice generates a message digest (i.e., hash) of the original plaintext message using one of the cryptographically sound hashing algorithms, such as SHA2-512.
2. Alice then encrypts only the message digest using her private key. This encrypted message digest is the digital signature.
3. Alice appends the signed message digest to the plaintext message.
4. Alice transmits the appended message to Bob.

When Bob receives the digitally signed message, he reverses the procedure, as follows:

1. Bob decrypts the digital signature using Alice's public key.
2. Bob uses the same hashing function to create a message digest of the full plaintext message received from Alice.
3. Bob then compares the decrypted message digest he received from Alice with the message digest he computed himself. If the two digests match, he can be assured that the message he received was sent by Alice. If they do not match, either the message was not sent by Alice or the message was modified while in transit.

Digital signatures are used for more than just messages. Software vendors often use digital signature technology to authenticate code distributions that you download from the Internet, such as applets and software patches.

Note that the digital signature process does not provide confidentiality in and of itself. It only ensures that the cryptographic goals of integrity, authentication, and nonrepudiation are met. Let's break that down. If the hash generated by the sender and the hash generated by the recipient match, then we know that the two hashed messages are identical and we have integrity. If the digital signature was verified with the public key of the sender, then we know that it was created using that sender's private key. That private key should be known only to the holder of the key (the sender, in this case), so the verification of the private key proves to the recipient that the digital signature came from the sender, providing origin authentication. The recipient (or anyone else) can then demonstrate that process to a third party, providing nonrepudiation.

If Alice also wanted to ensure the confidentiality of her message to Bob, she could add an additional step to the message creation process. After appending the digitally signed message digest to the plaintext message, Alice could encrypt the entire message with Bob's public key. Then, when Bob would receive the message, he would first decrypt it with his own private key before following the steps just outlined.

HMAC

The Hash-Based Message Authentication Code (HMAC), also known as Keyed-Hash Message Authentication Code, symmetric (secret) key algorithm implements a *partial* digital signature—it

guarantees the integrity of a message during transmission, but it does not provide for nonrepudiation.

Which Key Should I Use?

If you're new to public key cryptography, selecting the correct key for various applications can be quite confusing. Encryption, decryption, message signing, and signature verification all use the same algorithm with different key inputs. Here are a few simple rules to help keep these concepts straight in your mind:

- If you want to encrypt a confidential message, use the recipient's public key.
- If you want to decrypt a confidential message sent to you, use your private key.
- If you want to digitally sign a message you are sending to someone else, use your private key.
- If you want to verify the digital signature on a message sent by someone else, use the sender's public key.

These four rules are the core principles of public key cryptography and digital signatures. If you understand each of them, you're off to a great start.

HMAC can be combined with any standard hash function, such as MD5, SHA-2, or SHA-3, by using a shared secret key. Therefore, only communicating parties who know the secret key can generate or verify the *partial* digital signature. If the recipient decrypts the message digest but cannot successfully compare it to a message digest generated from the original plaintext message, that means the message was altered in transit.

Because HMAC relies on a shared secret key, it does not provide any nonrepudiation functionality (as previously mentioned). However, it operates in a more efficient manner than the digital signature standard (DSS) described in the following section and

may be suitable for applications in which symmetric key cryptography is appropriate. In short, HMAC represents a halfway point between the unencrypted use of a hash function and the computationally expensive digital signature algorithms (DSA) based on public key cryptography.

Digital Signature Standard

NIST specifies the digital signature algorithms acceptable for federal government use in FIPS 186-5, also known as the Digital Signature Standard (DSS). This document specifies that all federally approved digital signature algorithms must use the SHA-3 hashing functions.

DSS also specifies the encryption algorithms that can be used to support a digital signature infrastructure. There are three currently approved standard encryption algorithms in FIPS 186-5:

- The Rivest–Shamir–Adleman (RSA) digital signature algorithm, as specified in IETF RFC 8017.
- The Elliptic Curve Digital Signature Algorithm (ECDSA), as specified in FIPS 186-5. This algorithm provides inherent support for nonrepudiation.
- The Edwards Curve Digital Signature Algorithm (EdDSA), as specified in IETF RFC 8032.

Public Key Infrastructure

The major strength of public key encryption is its ability to facilitate communication between parties previously unknown to each other. This is made possible by the *public key infrastructure (PKI)* hierarchy of trust relationships. These trusts permit combining asymmetric cryptography with symmetric cryptography along with hashing and digital certificates, giving us hybrid cryptography.

In the following sections, you'll learn the basic components of the public key infrastructure and the cryptographic concepts that make secure global communications possible. You'll learn the

composition of a digital certificate, the role of certificate authorities, and the process used to generate and destroy digital certificates.

Certificates

Digital *certificates* provide communicating parties with the assurance that the people they are communicating with truly are who they claim to be. Digital certificates are essentially endorsed copies of an individual's public key. When users verify that a certificate was signed by a trusted certificate authority (CA), they know that the public key is legitimate.

Digital certificates contain specific identifying information, and their construction is governed by the International Telecommunications Union (ITU), an international standard—X.509. Certificates that conform to X.509 contain the following data:

- Version of X.509 to which the certificate conforms
- Serial number (from the certificate creator)
- Signature algorithm identifier (specifies the technique used by the certificate authority to digitally sign the contents of the certificate)
- Issuer name (identification of the certificate authority [CA] that issued the certificate)
- Validity period (specifies the dates and times—a starting date and time and an expiration date and time—during which the certificate is valid)
- Subject's name (contains the common name [CN] of the certificate as well as the distinguished name [DN] of the entity that owns the public key contained in the certificate)
- Subject's public key (the meat of the certificate—the actual public key the certificate owner used to set up secure communications)

Certificates may be issued for a variety of purposes. These include providing assurance for the public keys of:

- Computers/machines
- Individual users
- Email addresses
- Developers (code-signing certificates)

The subject of a certificate may include a wildcard in the certificate name, indicating that the certificate is good for subdomains as well. The wildcard is designated by an asterisk character. For example, a wildcard certificate issued to `*.example.org` would be valid for all of the following domains:

- `example.org`
- `www.example.org`
- `mail.example.org`
- `secure.example.org`

Wildcard certificates are only good for one level of subdomain. Therefore, the `*.example.org` certificate would not be valid for the `www.cissp.example.org` subdomain.

Certificate Authorities

Certificate authorities (CAs) are the glue that binds the public key infrastructure together. These neutral organizations offer notarization services for digital certificates. To obtain a digital certificate from a reputable CA, you must prove your identity to the satisfaction of the CA. The following list includes some of the major CAs who provide widely accepted digital certificates:

- IdenTrust

- AWS Certificate Manager (ACM)
- GlobalSign
- ComodoCA
- Certum
- GoDaddy
- DigiCert
- SECOM Trust Systems
- Entrust
- Actalis
- Trustwave

Nothing is preventing any organization from simply setting up shop as a certificate authority. However, the certificates issued by a CA are only as good as the trust placed in the CA that issued them. This is an important item to consider when receiving a digital certificate from a third party. If you don't recognize and trust the name of the CA that issued the certificate, you shouldn't place any trust in the certificate at all. PKI relies on a hierarchy of trust relationships. If you configure your browser to trust a CA, it will automatically trust all of the digital certificates issued by that CA. Browser developers preconfigure browsers to trust the major CAs to avoid placing this burden on users.

Let's Encrypt is a well-known CA because they offer free certificates in an effort to encourage the use of encryption. You can learn more about this free service at http://letsencrypt.org.

Registration authorities (RAs) assist CAs with the burden of verifying users' identities prior to the issuance of digital certificates. They do not directly issue certificates themselves, but they play an important role in the certification process, allowing CAs to remotely validate user identities.

Certificate authorities must carefully protect their own private keys to preserve their trust relationships. To do this, they often use an *offline CA* to protect their *root certificate*, the top-level certificate for their entire PKI. This offline CA is disconnected from networks and powered down until it is needed. The offline CA uses the root certificate to create subordinate *intermediate CAs* that serve as the *online CAs* used to issue certificates on a routine basis.

In the CA trust model, the use of a series of intermediate CAs is known as *certificate chaining*. To validate a certificate, the browser verifies the identity of the intermediate CA(s) first and then traces the path of trust back to a known root CA, verifying the identity of each link in the chain of trust.

Certificate authorities do not need to be third-party service providers. Many organizations operate internal CAs that provide *self-signed certificates* for use exclusively inside an organization. These certificates won't be trusted by the browsers of external users, but internal systems may be configured to trust the internal CA, saving the expense of purchasing certificates from a third-party CA.

Certificate Life Cycle

The technical concepts behind the public key infrastructure (PKI) are relatively simple. In the following sections, we'll cover the processes used by certificate authorities (CAs) to create, validate, and revoke client certificates.

Enrollment

When you want to obtain a digital certificate, you must first prove your identity to the CA in some manner; this process is called *enrollment*. As mentioned in the previous section, this sometimes involves physically appearing before an agent of the certificate authority with the appropriate identification documents. Some certificate authorities provide other means of verification, including the use of credit report data and identity verification by trusted community leaders.

Once you've satisfied the certificate authority regarding your identity, you provide them with your public key in the form of a *certificate signing request (CSR)*. The CA next creates an X.509 digital certificate containing your identifying information and a copy of your public key. The CA then digitally signs the certificate using the CA's private key and provides you with a copy of your signed digital certificate. You may then safely distribute this certificate to anyone with whom you want to communicate securely.

Certificate authorities issue different types of certificates depending on the level of identity verification that they perform. The simplest, and most common, certificates are *Domain Validation (DV) certificates*, where the CA simply verifies that the certificate subject has control of the domain name. *Extended Validation (EV) certificates* provide a higher level of assurance, and the CA takes steps to verify that the certificate owner is a legitimate business before issuing the certificate.

Verification

When you receive a digital certificate from someone with whom you want to communicate, you *verify* the certificate by checking the CA's digital signature using the CA's public key. You then must check the validity period of the certificate to ensure that the current date is after the starting date of the certificate and that the certificate has not yet expired. Finally, you must check and ensure that the certificate was not revoked using a *certificate revocation list (CRL)* or the *Online Certificate Status Protocol (OCSP)*. At this point, you may assume that the public key listed on the certificate is authentic, provided that it satisfies the following requirements:

- The digital signature of the CA is authentic.
- You trust the CA.
- The certificate is not listed on a CRL.
- The certificate actually contains the data you are trusting.

The last point is a subtle but extremely important item. Before you trust an identifying piece of information about someone, be sure that it is actually contained within the certificate. If a certificate contains the email address (billjones@foo.com) but not the individual's name, you can be certain only that the public key contained therein is associated with that email address. The CA is not making any assertions about the actual identity of the billjones@foo.com email account. However, if the certificate contains the name Bill Jones along with an address and telephone number, the CA is vouching for that information as well.

Digital certificate verification algorithms are built into a number of popular web browsing and email clients, so you won't often need to get involved in the particulars of the process. However, it's important to have a solid understanding of the technical details taking place behind the scenes to make appropriate security judgments for your organization. It's also the reason that, when purchasing a certificate, you choose a CA that is widely trusted. If a CA is not included in, or is later pulled from, the list of CAs trusted by a major browser, it will greatly limit the usefulness of your certificate.

In 2017, a significant security failure occurred in the digital certificate industry. Symantec, through a series of affiliated companies, issued several digital certificates that did not meet industry security standards. In response, Google announced that the Chrome browser would no longer trust Symantec certificates. As a result, Symantec wound up selling off its certificate-issuing business to DigiCert, which agreed to properly validate certificates prior to issuance. This demonstrates the importance of properly validating certificate requests. A series of seemingly small lapses in procedure can decimate a CA's business.

Certificate pinning approaches instruct browsers to attach (pin) a certificate to a host for an extended period of time. When sites use certificate pinning, the browser associates that site (domain or subdomain) with their public key. This allows users or administrators to notice and intervene if a certificate unexpectedly changes.

Revocation

Occasionally, a certificate authority needs to *revoke* a certificate. This might occur for one of the following reasons:

- The certificate was compromised (for example, the certificate owner accidentally gave away the private key).
- The certificate was erroneously issued (for example, the CA mistakenly issued a certificate without proper verification).
- The details of the certificate changed (for example, the subject's name changed).
- The security association changed (for example, the subject is no longer employed by the organization sponsoring the certificate).

The revocation request grace period is the maximum response time within which a CA will perform any requested revocation. This is defined in the *Certificate Practice Statement (CPS)*. The CPS states the practices a CA employs when issuing or managing certificates.

You can use three techniques to verify the authenticity of certificates and identify revoked certificates:

Certificate Revocation Lists Certificate revocation lists (CRLs) are maintained by the various certificate authorities and contain the serial numbers of certificates that have been issued by a CA and that have been revoked, along with the date and time the revocation went into effect. The major disadvantage to certificate revocation lists is that they must be downloaded and cross-referenced periodically, introducing a period of latency between the time a certificate is revoked and the time end users are notified of the revocation.

Online Certificate Status Protocol (OCSP) This protocol eliminates the latency inherent in the use of certificate

revocation lists by providing a means for real-time certificate verification. When a client receives a certificate, it sends an OCSP request to the CA's OCSP server. The server then responds with a status of good, revoked, or unknown. The browser uses this information to determine whether the certificate is valid.

Certificate Stapling The primary issue with OCSP is that it places a significant burden on the OCSP servers operated by certificate authorities. These servers must process requests from every single visitor to a website or other user of a digital certificate, verifying that the certificate is valid and not revoked.

Certificate stapling is an extension to the Online Certificate Status Protocol that relieves some of the burden placed on certificate authorities by the original protocol. In the absence of OCSP, when a user visits a website and initiates a secure connection, the web server sends its digital certificate to the user's browser. The user's browser would normally then be responsible for contacting an OCSP server to verify the certificate's validity.

In certificate stapling, the web server contacts the OCSP server itself and receives a signed and timestamped response from the OCSP server, which it then attaches, or staples, to the digital certificate. Then, when a user requests a secure web connection, the web server sends the digital certificate with the stapled OCSP response to the user's browser. The user's browser then verifies whether the certificate is authentic and also validates that the stapled OCSP response is genuine and recent. Because the CA signed the OCSP response, the user's browser knows that it is from the CA, and the timestamp provides the user's browser with assurance that the CA recently validated the certificate. From there, communication may continue as normal.

The time savings come when the next user visits the website. The web server can simply reuse the stapled certificate without recontacting the OCSP server. As long as the timestamp is recent enough, the user's browser will accept the stapled certificate without needing to contact the CA's OCSP server again. It's common to have stapled certificates with a validity period of 24 hours. That reduces the burden on an OCSP server from

handling one request per user over the course of a day, which could be millions of requests, to handling one request per certificate per day. That's a tremendous reduction.

Certificate Formats

Digital certificates are stored in files, and those files come in a variety of formats, both binary and text-based:

- The most common binary format is the Distinguished Encoding Rules (DER) format. DER certificates are normally stored in files with the `.der`, `.crt`, or `.cer` extension.
- The Privacy-Enhanced Mail (PEM) certificate format is an ASCII text version of the DER format. PEM certificates are normally stored in files with the `.pem` or `.crt` extension.

You may have picked up on the fact that the `.crt` file extension is used for both binary DER files and text PEM files. That's very confusing. You should remember that you can't tell whether a CRT certificate is binary or text without actually looking at the contents of the file.

- The Personal Information Exchange (PFX) format is commonly used by Windows systems. PFX certificates may be stored in binary form, using either `.pfx` or `.p12` file extensions.
- Windows systems also use P7B certificates, which are stored in ASCII text format using the `.p7b` file extension.

Table 7.2 provides a summary of certificate formats.

TABLE 7.2 Digital certificate formats

File format names	Format	File extension(s)
Distinguished Encoding Rules (DER)	Binary	`.der`, `.crt`, `.cer`
Privacy-Enhanced Mail (PEM)	Text	`.pem`, `.crt`
Personal Information Exchange (PFX)	Binary	`.pfx`, `.p12`
P7B	Text	`.p7b`

Asymmetric Key Management

When working within the public key infrastructure, you must comply with several best practice requirements to maintain the security of your communications.

First, choose your encryption system wisely. As you learned earlier, "security through obscurity" is not an appropriate approach. Choose an encryption system with an algorithm in the public domain that has been thoroughly vetted by industry experts. Be wary of systems that use a "black-box" approach and maintain that the secrecy of their algorithm is critical to the integrity of the cryptosystem.

You must also select your keys in an appropriate manner. Use a key length that balances your security requirements with performance considerations. Also, ensure that your key is truly random. Any patterns within the key increase the likelihood that an attacker will be able to break your encryption and degrade the security of your cryptosystem.

When using public key encryption, keep your private key secret. Do not, under any circumstances, allow anyone else to gain access to your private key. Remember, allowing someone access even once permanently compromises all communications that take place (past, present, or future) using that key and allows the third party to successfully impersonate you.

Retire keys when they've served a useful life. Many organizations have mandatory key rotation requirements to protect against undetected key compromise. If you don't have a formal policy that you must follow, select an appropriate interval based on the frequency with which you use your key. You might want to change your key pair every few months, if practical.

Back up your key. If you lose the file containing your private key because of data corruption, disaster, or other circumstances, you'll certainly want to have a backup available. You may want to either create your own backup or use a key escrow service that maintains the backup for you. In either case, ensure that the backup is handled in a secure manner. After all, it's just as important as your primary key file.

Hardware security modules (HSMs) also provide an effective way to manage encryption keys. These hardware devices store and manage encryption keys in a secure manner that prevents humans from ever needing to work directly with the keys. Many of them are also capable of improving the efficiency of cryptographic operations, in a process known as hardware acceleration. HSMs range in scope and complexity from very simple devices, such as the YubiKey, that store encrypted keys on a USB drive for personal use, to more complex enterprise products that reside in a data center. HSMs include tamper-resistance mechanisms to prevent someone who gains physical access to the device from accessing the cryptographic material it maintains. Cloud service providers, such as Amazon and Microsoft, also offer cloud-based HSMs that provide secure key management for infrastructure-as-a-service (IaaS) cloud computing resources.

Hybrid Cryptography

You've now learned about the two major categories of cryptographic systems: symmetric and asymmetric algorithms. You've also learned about the major advantages and disadvantages of each. Chief among these are the facts that symmetric algorithms are fast but introduce key distribution

challenges, and though asymmetric algorithms solve the key distribution problem, they are also computationally intensive and slow. If you're choosing between these approaches, you're forced to make a decision between convenience and speed.

Hybrid cryptography combines symmetric and asymmetric cryptography to achieve the shared secret key distribution benefits of asymmetric cryptosystems with the speed of symmetric algorithms. These approaches work by setting up an initial connection between two communicating entities using asymmetric cryptography. That connection is used for only one purpose: the exchange of a randomly generated shared secret key, known as an *ephemeral key*. The two parties then exchange whatever data they wish using the shared secret key with a symmetric algorithm. When the communication session ends, they discard the ephemeral key and then repeat the same process if they wish to communicate again later.

The beauty behind this approach is that it uses asymmetric cryptography for the shared secret key distribution, a task that requires the encryption of only a small amount of data. Then it switches to the faster symmetric algorithm for the vast majority of data exchanged.

Transport Layer Security (TLS) is the most well-known example of hybrid cryptography, and we discuss that approach later in this chapter.

Applied Cryptography

Up to this point, you've learned a great deal about the foundations of cryptography, the inner workings of various cryptographic algorithms, and the use of the public key infrastructure (PKI) to distribute identity credentials using digital certificates. You should now feel comfortable with the basics of cryptography and be prepared to move on to higher-level applications of this technology to solve everyday communications problems.

In the following sections, we'll examine the use of cryptography to secure data at rest, such as that stored on portable devices, as

well as data in transit, using techniques that include secure email, encrypted web communications, and networking.

Portable Devices

The now ubiquitous nature of laptop computers, smartphones, and tablets brings new risks to the world of computing. Those devices often contain highly sensitive information that, if lost or stolen, could cause serious harm to an organization and its customers, employees, and affiliates. For this reason, many organizations turn to encryption to protect the data on these devices in the event they are misplaced.

Current versions of popular operating systems now include disk encryption capabilities that make it easy to apply and manage encryption on portable devices. For example, Microsoft Windows includes the BitLocker and Encrypting File System (EFS) technologies, macOS includes FileVault encryption, and the VeraCrypt open source package allows the encryption of disks on Linux, Windows, and Mac systems.

Trusted Platform Module

Modern computers often include a specialized cryptographic component known as a Trusted Platform Module (TPM). The TPM is a chip that resides on the motherboard of the device. The TPM serves a number of purposes, including the storage and management of cryptographic keys used for full-disk encryption (FDE) solutions. The TPM provides the operating system with access to the keys only if the user successfully authenticates. This prevents someone from removing the drive from one device and inserting it into another device to access the drive's data.

A wide variety of commercial tools are available that provide added features and management capability. The major differentiators between these tools are how they protect keys stored in memory, whether they provide full-disk or volume-only

encryption, and whether they integrate with hardware-based Trusted Platform Modules (TPMs) to provide added security. Any effort to select encryption software should include an analysis of how well the alternatives compete on these characteristics.

Don't forget about smartphones when developing your portable device encryption policy. Most major smartphone and tablet platforms include enterprise-level functionality that supports encryption of data stored on the phone.

Email

We have mentioned several times that security should be cost-effective. When it comes to email, simplicity is the most cost-effective option, but sometimes cryptography functions provide specific security services that you can't avoid using. Since ensuring security is also cost-effective, here are some simple rules about encrypting email:

- If you need confidentiality when sending an email message, encrypt the message.
- If your message must maintain integrity, you must hash the message.
- If your message needs authentication, integrity, and/or nonrepudiation, you should digitally sign the message.
- If your message requires confidentiality, integrity, origin authentication, and nonrepudiation, you should encrypt and digitally sign the message.

It is always the responsibility of the sender to put proper mechanisms in place to ensure that the security (that is, confidentiality, integrity, authenticity, and nonrepudiation) of a message or transmission is maintained.

The coverage of email in this chapter focuses on the use of cryptography to provide secure communications between two parties. You'll find more coverage of email security topics in Chapter 12, “Secure Communications and Network Attacks.”

One of the most in-demand applications of cryptography is encrypting and signing email messages. Until recently, encrypted email required the use of complex, awkward software that in turn required manual intervention and complicated key exchange procedures. An increased emphasis on security in recent years resulted in the implementation of strong encryption technology in mainstream email packages. Next, we'll look at some of the secure email standards in widespread use today.

Pretty Good Privacy

Phil Zimmermann's Pretty Good Privacy (PGP) secure email system appeared on the computer security scene in 1991. It combines the CA hierarchy described earlier in this chapter with the “web of trust” concept—that is, you must become trusted by one or more PGP users to begin using the system. You then accept their judgment regarding the validity of additional users and, by extension, trust a multilevel “web” of users descending from your initial trust judgments.

PGP initially encountered a number of hurdles to widespread use. The most difficult obstruction was the U.S. government export regulations, which treated encryption technology as munitions and prohibited the distribution of strong encryption technology outside the United States. Fortunately, this restriction has since been repealed, and PGP may be freely distributed to most countries.

PGP is available in two versions: a commercial product called PGP Encryption Solutions, which is available through Broadcom's Symantec Enterprise Division, and an open source variant called

OpenPGP. These products allow for the use of modern encryption algorithms, hash functions, and signature standards within the PGP framework.

PGP messages are often sent in text-encoded format to facilitate compatibility with other email systems. Here is an example of how an encrypted message appears when sent using PGP:

```
-----BEGIN PGP MESSAGE-----

hQGMAyHB9q9kWbl7AQwAmgyZoaXC2Xvo3jrVIWains3/UvUImp3YEbcEml
LK+26o
TNGBSNi5jLi2A62e8TLGbPkJv5vN3JZH4F27ZvYIhqANwk2nTI1sE0bA2R
zlw6Pc
XCUooGhNY/rmmWTLvWNVRdSXZj2i28fk2gi2QJlrEwYLkKJdUxzKldSLht
+Bc+V2
NbvQrTzJ0LmRq9FKvZ4lz5v7Qj/f1GdKF/5HCTthUWxJMxxuSzCp46rFR6
sKAQXG
tHdi2IzrroyQLR23HO6KuleisGf1X2wzfWENlXMUNGNLxPi2YNvo3MaFMM
w3o1dF
Zj28ptpCH8eGOVIAa05ZNnCk2a6alqTf9aKH8932uCS/AcYG3xqVcRCz7q
yaLqD5
NFg4GXq10KD8Jo1VP/HncOx7/39MGRDuzJqFieQzsVo0uCwVB2zJYC0SeJ
yMHkyD
TaAxz4HMQxzm8FubreTfisXKuUfPbYAuT855kc2iBKTGo9Cz1WjhQo6mve
I6hvu0
qYUaX5sGgfbD4bzCMFJj0nUBUdMni0jqHJ2XuZerEd8m0DioUOBRJybLlo
htRkik
Gzra/+WGE1ckQmzch5LDPdIEZphvV+5/DbhHdhxN7QMWe6ZkaADAZRgu77
tkQK6c
QvrBPZdk22uS0vzdwzJzzvybspzq1HkjD+aWR9CpSZ9mukZPXew=
=7NWG
-----END PGP MESSAGE-----
```

Similarly, digitally signed messages contain the text of the message followed by a PGP signature. Here is an example:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA256

I am enjoying studying cybersecurity.
-----BEGIN PGP SIGNATURE-----

iQGzBAEBCAAdFiEE75kumjjPhsn37slI+Cb2Pddh6OYFAmAF4FMACgkQ+C
b2Pddh
6Oba4gv8D4ybEtYidHdlfDYfbF+wYAz8JZ0Mw//f41iwkBG6BO6RtKtNPV
202Ngb
3Uxqjody48ndmDM4q60x3EMy+97ZXNoZL7fY5vv2viDa1so4BqevtRKYe6
sfjxMg
```

```
XImhPVxUknWhJUlUopQvsetBe51nqiqhpVONx/GRDXR9gdmGO89gD7XSCy
0vHhEW
AuoBVNBjbXqmxWdBPdrGcA9zFhdvxzmc6iI4zYe2mQxk1Nt1K6PRXNGjJL
IxqchL
sD7rLVYG1I7+CLGYreJH0siW0Xltbr96qT++1u4tMo1ng1UraoB21zTPVc
HA0pJu
DLrlXB0GFxVbDHpttOhYDPFZPk4NpzztDuAeNCA5/Oi3JJMjzBRrRuoIH7
abmePX
qc0Bl1/DAbbiYd5uX01i8ejIveLoeb4OZfLZH/j+bJZT5762Wx0DwkVtm8
smk6nl
+whpAZb5MV6SaS1xEcsRpU+w/O61OPteZ6eIHkU9pDu0yXM6IdtfRpqEw3
LKVN/M
zblGsAq4
=GXp+
-----END PGP SIGNATURE-----
```

The preceding example sends the message in plaintext with a PGP signature appended to the bottom. If you add encryption to protect the confidentiality of the message, the encryption is applied after the message is digitally signed, producing output that appears similar to any other encrypted message. For example, here is that same digitally signed message with encryption added:

```
-----BEGIN PGP MESSAGE-----

owEBBwL4/ZANAwAIAfgm9j3XYejmAaxAYgh0ZXN0LnR4dGAF4N1JIGFtIG
Vuam95
aW5nIG15IHByZXBhcmF0aW9uIGZvciB0aGUgQ0lTU1AgZXhhbS4KCokBsw
QAAQgA
HRYhBO+ZLpo4z4bJ9+7JSPgm9j3XYejmBQJgBeDdAAoJEPgm9j3XYejmLf
oL/RRW
oDUl+AeZGffqwnYiJH2gB+Tn+pLjnXAhdf/YV4OsWEsjqKBvItctgcQuSO
FJzuO+
jNgoCAFryi6RrwJ6dTh3F50QJYyJYlgIXCbkyVlaV6hXCZWPT40Bk/pI+H
X9A6l4
J272xabjFf63/HiIEUJDHg/9u8FXKVvBImV3NuMMjJEqx9RcivwvpPn6YL
JJ1MWy
zlUhu3sUIGDWNlArJ4SdskfY32hWAvHkgOAY8JSYmG6L6SVhvbRgv3d+rO
OlutqK
4bVIO+fKMvxycnluPuwmVH99I1Ge8p1ciOMYCVg0dBEP/DeoFlQ4tvKMCP
JG0w0E
ZgLgKyKQpjmNU9BheGvIfzRt1dKYeMx7lGZPlu7rr1Fk0oX/yMiaePWy5N
YE2O5I
D6op9EcJImcMn8wmPM9YTZbmcfcumSpaG1i0EzzAT5eMXn3BoDij12JJrk
CCbhYy
34u2CFR4WycGIIoFHV4RgKqu5TTuV+SCc//vgBaN20Qh9p7gRaNfOxHspt
o6fA==
```

```
=oTCB
-----END PGP MESSAGE-----
```

As you can see, it is not possible to tell that this message is digitally signed until after it is decrypted.

Many commercial providers also offer PGP-based email services as web-based cloud email offerings, mobile device applications, or webmail plug-ins. These services appeal to administrators and end users because they remove the complexity of configuring and maintaining encryption certificates and provide users with a managed secure email service. Some products in this category include Proton Mail, StartMail, Mailvelope, SafeGmail Chrome extension, and Hushmail.

S/MIME

The Secure/Multipurpose Internet Mail Extensions (S/MIME) protocol has emerged as a de facto standard for encrypted email. S/MIME uses the RSA encryption algorithm and has received the backing of major industry players, including RSA Security. S/MIME has already been incorporated in a large number of commercial products, including these:

- Microsoft Outlook and Microsoft 365
- Apple's iCloud Mail
- Google Workspace Enterprise Plus edition

S/MIME relies on the use of X.509 certificates for exchanging cryptographic keys. The public keys contained in these certificates are used for digital signatures and for the exchange of symmetric keys used for longer communications sessions. Users who receive a message signed with S/MIME will be able to verify that message by using the sender's digital certificate. Users who wish to use S/MIME for confidentiality or want to create their own digitally signed messages must obtain their own certificates.

Despite strong industry support for the S/MIME standard, technical limitations have prevented its widespread adoption. Although major desktop mail applications support S/MIME

email, mainstream web-based email systems do not support it out of the box (the use of browser extensions is required).

Web Applications

Encryption is widely used to protect web transactions. This is mainly because of the strong movement toward ecommerce and the desire of both ecommerce vendors and consumers to securely exchange financial information (such as credit card information) over the web. We'll look at the two technologies that are responsible for the small lock icon within web browsers—Secure Sockets Layer (SSL) and Transport Layer Security (TLS).

Secure Sockets Layer (SSL)

SSL was originally developed by Netscape to provide client/server encryption for web traffic sent using the Hypertext Transfer Protocol Secure (HTTPS) over port 443. Over the years, security researchers discovered a number of critical flaws in the SSL protocol that render it insecure for use today. However, SSL serves as the technical foundation for its successor, Transport Layer Security (TLS), which remains widely used today.

Even though TLS has been in existence for more than a decade, many people still mistakenly call it SSL. When you hear people use the term *SSL*, that's a red flag that you should further investigate to ensure that they're really using the modern, secure TLS and not the outdated SSL.

Transport Layer Security (TLS)

TLS relies on the exchange of server digital certificates to negotiate encryption/decryption parameters between the browser and the web server. TLS's goal is to create a secure communications channel that remains open for an entire web browsing session. It depends on a combination of symmetric and

asymmetric cryptography. The following steps are involved in a TLS 1.3 connection:

1. When a user accesses a website, their browser and the web server negotiate a cipher suite that is supported by both.
2. The browser retrieves the web server's digital certificate and extracts the server's public key from it.
3. The browser creates a random symmetric key (known as the ephemeral key), uses the web server's public key to encrypt the ephemeral key, and sends the encrypted ephemeral key to the web server.
4. The web server decrypts the ephemeral key using its own private key, and the two systems exchange all future messages using the ephemeral key.

This approach allows TLS to leverage the advanced functionality of asymmetric cryptography while encrypting and decrypting the vast majority of the data exchanged using the faster symmetric algorithm.

When TLS was first proposed as a replacement for SSL, not all browsers supported the more modern approach. To ease the transition, early versions of TLS supported downgrading communications to SSL v3.0 when both parties did not support TLS. However, in 2011, TLS v1.2 dropped this backward compatibility.

In 2014, an attack known as the Padding Oracle On Downgraded Legacy Encryption (POODLE) demonstrated a significant flaw in the SSL 3.0 fallback mechanism of TLS. In an effort to remediate this vulnerability, many organizations completely dropped SSL support and now rely solely on TLS security.

The original version of TLS, TLS 1.0, was simply an enhancement to the SSL 3.0 standard. TLS 1.1, developed in 2006 as an upgrade to TLS 1.0, also contains known security vulnerabilities. TLS 1.2, released in 2008, is now considered the minimum secure option. TLS 1.3, released in 2018, is secure and adds performance improvements. As of 2024, NIST requires that U.S. federal

agencies support TLS 1.3 and recommends the same for all other organizations.

It's important to understand that TLS is not an encryption algorithm itself. It is a protocol within which encryption algorithms may function. Therefore, it isn't sufficient to verify that a system is using a secure version of TLS. Security professionals must also ensure that the algorithms being used with TLS are secure as well.

Each system supporting TLS provides a listing of the cipher suites that it supports. These are combinations of encryption algorithms that it is willing to use together, and these lists are used by two systems to identify a secure option that both systems support. In TLS 1.3, a cipher suite consists of two components:

- The bulk encryption algorithm that will be used for symmetric encryption. For example, a server might support multiple versions of AES and 3DES.
- The hash algorithm that will be used to create message digests. For example, a server might support different versions of the SHA algorithm.

TLS 1.3 cipher suites are usually expressed in long strings that combine both of these elements. For example, the cipher suite:

```
TLS AES_256_CBC_SHA384
```

means that the server supports TLS using AES CBC mode with a 256-bit key for bulk encryption. Hashing will take place using the SHA-384 algorithm.

TLS 1.3 uses variants of the Diffie-Hellman key exchange algorithm. The systems participating in TLS 1.3 communication automatically determine the version of Diffie-Hellman to use and, therefore, the key exchange algorithm is not included in the cipher suite negotiation, as it was in earlier versions of TLS.

Tor and the Dark Web

Tor, formerly known as The Onion Router, provides a mechanism for anonymously routing traffic across the Internet using encryption and a set of relay nodes. It relies on a technology known as *perfect forward secrecy (PFS)*, where layers of encryption prevent nodes in the relay chain from reading anything other than the specific information they need to accept and forward the traffic. By using perfect forward secrecy in combination with a set of three or more relay nodes, Tor allows for both anonymous browsing of the standard Internet, as well as the hosting of completely anonymous sites on the dark web.

Steganography and Watermarking

Steganography is the art of using cryptographic techniques to embed secret messages within another message. Steganographic algorithms work by making alterations to the least significant bits of the many bits that make up image files. The changes are so minor that there is no appreciable effect on the viewed image. This technique allows communicating parties to conceal messages in plain sight—for example, they might embed a secret message within an illustration on an otherwise innocent web page.

It is also possible to embed messages inside larger excerpts of text. This approach is known as a concealment cipher.

Steganographers often embed their secret messages within images or WAV files because these files are often so large that the secret message would easily be missed by even the most observant inspector. Steganography techniques are often used for illegal activities, such as espionage and child pornography.

Steganography can also be used for legitimate purposes, however. Adding digital watermarks to documents to protect intellectual property is accomplished by means of steganography. The hidden

information is known only to the file's creator. If someone later creates an unauthorized copy of the content, the watermark can be used to detect the copy and (if uniquely watermarked files are provided to each original recipient) trace the offending copy back to the source.

Steganography commonly works by modifying the least significant bit (LSB) of a pixel value in its binary representation. For example, in the RGB color model, each pixel is described by using three decimal numbers, each ranging from 0 to 255. The first number represents the degree of red color in a pixel, the second represents green, and the third represents blue. If a pixel has the blue value of 64 (binary value of 1000000), changing the LSB to 1 would result in the binary value of 1000001 or the decimal equivalent of 65. This is an imperceptible change but does allow the encoding of a bit of steganographic data.

Steganography is an extremely simple technology to use, with free tools openly available on the Internet. Figure 7.2 shows the entire interface of one such tool, iSteg. It simply requires that you specify a text file containing your secret message and an image file that you wish to use to hide the message. Figure 7.3 shows an example of a picture with an embedded secret message; the message is impossible to detect with the human eye because the text file was added into the message by modifying only the least significant bits of the file. Those do not survive the printing process, and in fact, even if you examined the original full-color, high-resolution digital images, you would not be able to detect the difference.

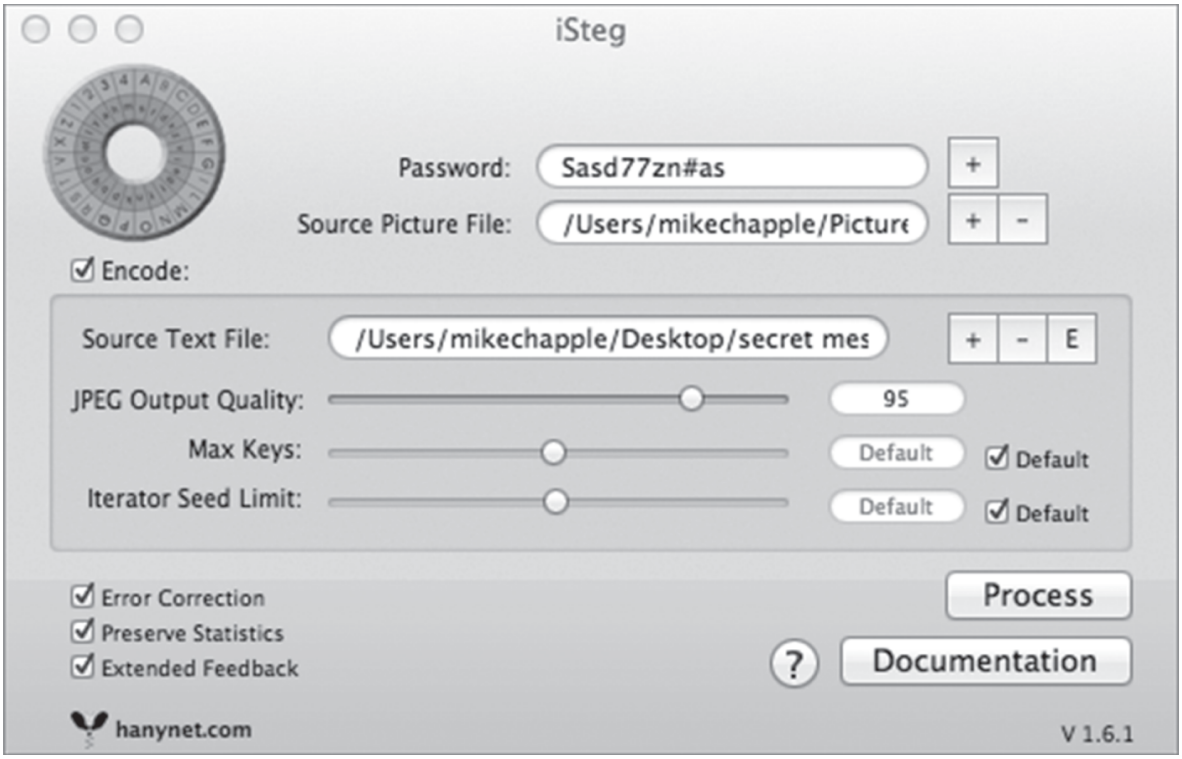

FIGURE 7.2 Steganography tool

FIGURE 7.3 Image with embedded message

Networking

The final application of cryptography we'll explore in this chapter is the use of cryptographic algorithms to provide secure networking services. In the following sections, we'll take a brief look at methods used to secure communications circuits.

Circuit Encryption

Security administrators use two types of encryption techniques to protect data traveling over networks:

- *Link encryption* protects entire communications circuits by creating a secure tunnel between two points using either a hardware solution or a software solution that encrypts all traffic entering one end of the tunnel and decrypts all traffic leaving the other end of the tunnel. For example, a company with two offices connected via a data circuit might use link encryption to protect against attackers monitoring at a point in between the two offices.
- *End-to-end encryption* protects communications between two parties (for example, a client and a server) and is performed independently of link encryption. An example of end-to-end encryption would be the use of TLS to protect communications between a user and a web server.

The critical difference between link and end-to-end encryption is that in link encryption, all the data, including the header, trailer, address, and routing data, is also encrypted. Therefore, each packet has to be decrypted at each hop so that it can be properly routed to the next hop and then reencrypted before it can be sent along its way, which slows the routing. End-to-end encryption does not encrypt the header, trailer, address, and routing data, so it moves faster from point to point but is more susceptible to sniffers and eavesdroppers.

When encryption happens at the higher OSI layers, it is usually end-to-end encryption, and if encryption is done at the lower layers of the OSI model, it is usually link encryption.

Secure Shell (SSH) is a good example of an end-to-end encryption technique. This suite of programs provides encrypted alternatives to common Internet applications such as the File Transfer Protocol (FTP), Telnet, and rlogin. There are two versions of SSH. SSH-1 is now considered insecure. SSH-2 drops support for some insecure algorithms and adds several security enhancements, including support for the Diffie–Hellman key exchange protocol and the ability to run multiple sessions over a single SSH connection (channel multiplexing). SSH-2 adds support for secure file transfer (SFTP). SSH-2 provides added protection against on-path attacks, eavesdropping, and IP/DNS spoofing.

IPSec

Various security architectures are in use today, each one designed to address security issues in different environments. One such architecture that supports secure communications is the Internet Protocol Security (IPSec) standard. IPSec is a standard architecture set forth by the Internet Engineering Task Force (IETF) for setting up a secure channel to exchange information between two entities.

The IP Security (IPSec) protocol provides a complete infrastructure for secured network communications. IPSec has gained widespread acceptance and is now offered in several commercial operating systems out of the box. IPSec relies on security associations, and there are two main components:

- The Authentication Header (AH) provides assurances of message integrity. AH also provides authentication and access control and prevents replay attacks.
- The Encapsulating Security Payload (ESP) provides confidentiality and integrity of packet contents. It provides encryption and limited authentication and prevents replay attacks.

ESP also provides some limited authentication, but not to the degree of the AH. Though ESP is sometimes used without AH, it's rare to see AH used without ESP.

IPSec provides for two discrete modes of operation. When IPSec is used in *transport mode* for end-to-end encryption, only the packet payload is encrypted. This mode is designed for peer-to-peer communication. When it's used in *tunnel mode*, the entire packet, including the header, is encrypted. This mode is designed for link encryption.

At runtime, you set up an IPSec session by creating a *security association (SA)*. The SA represents the communication session and records any configuration and status information about the connection. The SA represents a simplex connection. If you want a two-way channel, you need two SAs, one for each direction. Also, if you want to support a bidirectional channel using both AH and ESP, you will need to set up four SAs.

Some of IPSec's greatest strengths come from being able to filter or manage communications on a per-SA basis so that clients or gateways between which security associations exist can be rigorously managed in terms of what kinds of protocols or services can use an IPSec connection. Also, without a valid security association defined, pairs of users or gateways cannot establish IPSec links.

Further details of the IPSec algorithm are provided in Chapter 11, “Secure Network Architecture and Components.”

Emerging Applications

Cryptography plays a central role in many emerging areas of cybersecurity and technology. Let's look at a few of these concepts: the blockchain, lightweight cryptography, and homomorphic encryption.

Blockchain

The *blockchain* is, in its simplest description, a distributed and immutable public ledger. This means that it can store records in a way that distributes those records among many different systems located around the world and do so in a manner that prevents anyone from tampering with those records. The blockchain creates a data store that nobody can tamper with or destroy.

The first major application of the blockchain is *cryptocurrency*. The blockchain was originally invented as a foundational technology for Bitcoin, allowing the tracking of Bitcoin transactions without the use of a centralized authority. In this manner, the blockchain allows the existence of a currency that has no central regulator. Authority for Bitcoin transactions is distributed among all participants in the Bitcoin blockchain.

Although cryptocurrency is the blockchain application that has received the most attention, there are many other uses for a distributed immutable ledger—so much so that new applications of blockchain technology seem to be appearing every day. For example, property ownership records could benefit tremendously from a blockchain application. This approach would place those records in a transparent, public repository that is protected against intentional or accidental damage. Blockchain technology might also be used to track supply chains, providing grocery consumers, for example, with confidence that their produce came from reputable sources and allowing regulators to easily track down the origin of recalled produce.

Lightweight Cryptography

There are many specialized use cases for cryptography that you may encounter during your career where computing power and energy might be limited.

Some devices operate at extremely low power levels and put a premium on conserving energy. For example, imagine sending a satellite into space with a limited power source. Thousands of hours of engineering go into getting as much life as possible out of that power source. Similar cases happen here on Earth, where

remote sensors must transmit information using solar power, a small battery, or other equipment.

Smartcards are another example of a low-power environment. They must be able to securely communicate with smartcard readers but only using the energy either stored on the card or transferred to it by a magnetic field.

In these cases, cryptographers often design specialized hardware that is purpose-built to implement lightweight cryptographic algorithms with as little power expenditure as possible. You won't need to know the details of how these algorithms work, but you should be familiar with the concept that specialized hardware can minimize power consumption.

Another specialized use for cryptography is in cases where you need very low latency. That simply means that the encryption and decryption should not take a long time. Encrypting network links is a common example where low-latency cryptography is desirable. The data is moving quickly across a network, and the encryption should be done as quickly as possible to avoid becoming a bottleneck.

Specialized encryption hardware also fulfills many low-latency requirements. For example, a dedicated VPN hardware device may contain cryptographic hardware that implements encryption and decryption operations in highly efficient form to maximize speed.

High resiliency requirements exist when it is extremely important that data be preserved and not accidentally destroyed during an encryption operation. In cases where resiliency is extremely important, the easiest way to address the issue is for the sender of data to retain a copy until the recipient confirms the successful receipt and decryption of the data.

Homomorphic Encryption

Privacy concerns also introduce some specialized use cases for encryption. In particular, we sometimes have applications where we want to protect the privacy of individuals but still want to perform calculations on their data. *Homomorphic encryption*

technology allows this, encrypting data in a way that preserves the ability to perform computation on that data. When you encrypt data with a homomorphic algorithm and then perform computation on that data, you get a result that, when decrypted, matches the result you would have received if you had performed the computation on the plaintext data in the first place.

Cryptographic Attacks

As with any security mechanism, malicious individuals have found a number of attacks to defeat cryptosystems. It's important that you understand the threats posed by various cryptographic attacks to minimize the risks posed to your systems:

Brute-Force Attack Brute-force attacks are quite straightforward. Such an attack attempts every possible valid combination for a key or password. They involve using massive amounts of processing power to methodically guess the key used to secure cryptographic communications.

Analytic Attack This is an algebraic manipulation that attempts to reduce the complexity of the algorithm. Analytic attacks focus on the logic of the algorithm itself.

Implementation Attack This is a type of attack that exploits weaknesses in the implementation of a cryptography system. It focuses on exploiting the hardware or the software code, not just errors and flaws but the methodology employed to program the encryption system.

Statistical Attack A statistical attack exploits statistical weaknesses in a cryptosystem, such as the inability to produce truly random numbers. Statistical attacks may be attempted against a database. Also, a vulnerability in the hardware or operating system hosting the cryptography application may be exploited.

Fault Injection Attack In these attacks, the attacker attempts to compromise the integrity of a cryptographic device by causing some type of external fault. For example, they might use high-

voltage electricity, high or low temperature, or other factors to cause a malfunction that undermines the security of the device.

Side-Channel Attack Computer systems generate characteristic footprints of activity, such as changes in processor utilization, electricity consumption, or electromagnetic radiation. Side-channel attacks seek to use this information to monitor system activity and retrieve information that is actively being encrypted.

Timing Attack Timing attacks are an example of a side-channel attack where the attacker measures precisely how long cryptographic operations take to complete, gaining information about the cryptographic process that may be used to undermine its security.

For a nonflawed protocol, the average amount of time required to discover the key through a brute-force attack is directly proportional to the length of the key. A brute-force attack will always be successful given enough time. Every additional bit of key length doubles the time to perform a brute-force attack because the number of potential keys doubles.

There are two modifications that attackers can make to enhance the effectiveness of a brute-force attack:

- Rainbow tables provide precomputed values for cryptographic hashes. These are commonly used for cracking passwords stored on a system in hashed form.
- Specialized, scalable computing hardware designed specifically for the conduct of brute-force attacks may greatly increase the efficiency of this approach.

Salting Saves Passwords

Salt might be hazardous to your health, but it can save your password. To help combat the use of brute-force attacks, including those aided by dictionaries and rainbow tables, cryptographers make use of a technology known as *cryptographic salt*.

The cryptographic salt is a random value that is added to the end of the password before the operating system hashes the password. The salt is then stored in the password file along with the hash. When the operating system wishes to compare a user's proffered password to the password file, it first retrieves the salt and appends it to the password. It feeds the concatenated value to the hash function and compares the resulting hash with the one stored in the password file.

Specialized password hashing functions, such as PBKDF2 (Password-Based Key Derivation Function 2), bcrypt, and scrypt, allow for the creation of hashes using salts and also incorporate a technique known as *key stretching* that makes it more computationally difficult to perform a single password guess.

The use of salting, especially when combined with key stretching, dramatically increases the difficulty of brute-force attacks. Anyone attempting to build a rainbow table must build a separate table for each possible value of the cryptographic salt.

Frequency Analysis and the Ciphertext-Only Attack In many cases, the only information you have at your disposal is the encrypted ciphertext message, a scenario known as the *ciphertext-only attack*. In this case, one technique that proves helpful against simple ciphers is frequency analysis—counting the number of times each letter appears in the ciphertext. Using your knowledge that the letters *E*, *T*, *A*, *O*, *I*,

N are the most common in the English language, you can then test several hypotheses:

- If these letters are also the most common in the ciphertext, the cipher was likely a transposition cipher, which rearranged the characters of the plaintext without altering them.
- If other letters are the most common in the ciphertext, the cipher is probably some form of substitution cipher that replaced the plaintext characters.

This is a simple overview of frequency analysis, and many sophisticated variations on this technique can be used against polyalphabetic ciphers and other sophisticated cryptosystems.

Known Plaintext Attack In the known plaintext attack, the attacker has a copy of the encrypted message along with the plaintext message used to generate the ciphertext (the copy). This knowledge greatly assists the attacker in breaking weaker codes. For example, imagine the ease with which you could break the Caesar cipher described in Chapter 6 if you had both a plaintext copy and a ciphertext copy of the same message.

Chosen Plaintext Attack In this attack, the attacker obtains the ciphertexts corresponding to a set of plaintexts of their own choosing. This allows the attacker to attempt to derive the key used and thus decrypt other messages encrypted with that key. This can be difficult, but it is not impossible. Advanced methods such as differential cryptanalysis are types of chosen plaintext attacks.

Chosen Ciphertext Attack In a chosen ciphertext attack, the attacker has access to the algorithm. They have the ability to decrypt chosen portions of the ciphertext message and use the decrypted portion of the message to discover the key.

Meet-in-the-Middle Attack Attackers might use a meet-in-the-middle attack to defeat encryption algorithms that use

two rounds of encryption. This attack is the reason that Double DES (2DES) was quickly discarded as a viable enhancement to the DES encryption (it was replaced by Triple DES, or 3DES).

In the meet-in-the-middle attack, the attacker uses a known plaintext message. The plaintext is then encrypted using every possible key (k1), and the equivalent ciphertext is decrypted using all possible keys (k2). When a match is found, the corresponding pair (k1, k2) represents both portions of the double encryption. This type of attack generally takes only double the time necessary to break a single round of encryption (or 2^n rather than the anticipated $2^n * 2^n$), offering minimal added protection.

Man-in-the-Middle Attack In the man-in-the-middle (MITM) attack, a malicious individual sits between two communicating parties and intercepts all communications (including the setup of the cryptographic session). The attacker responds to the originator's initialization requests and sets up a secure session with the originator. The attacker then establishes a second secure session with the intended recipient using a different key and posing as the originator. The attacker can then "sit in the middle" of the communication and read all traffic as it passes between the two parties.

Be careful not to confuse the meet-in-the-middle attack with the man-in-the-middle attack. They may have similar names, but they are quite different.

Birthday Attack The birthday attack, also known as a *collision attack* or *reverse hash matching* (see the discussion of brute-force and dictionary attacks in Chapter 14, "Controlling and Monitoring Access"), seeks to find flaws in the one-to-one nature of hashing functions. In this attack, the malicious individual seeks to replace the content of a

digitally signed communication with a different message that produces the same message digest, thereby maintaining the validity of the original digital signature.

Don't forget that social engineering techniques can also be used in cryptanalysis. If you're able to obtain a decryption key by simply asking the sender for it, that's much easier than attempting to crack the cryptosystem.

Replay Attack The replay attack is used against cryptographic algorithms that don't incorporate temporal protections. In this attack, the malicious individual intercepts an encrypted message between two parties (often a request for authentication) and then later "replays" the captured message to open a new session. This attack can be defeated by incorporating a timestamp and expiration period into each message, using a challenge-response mechanism, and encrypting authentication sessions with ephemeral session keys.

Many other attacks make use of cryptographic techniques as well. For example, Chapter 14 describes the use of cryptographic techniques in pass-the-hash and Kerberos exploitation, and Chapter 21, "Malicious Code and Application Attacks," describes the use of cryptography in ransomware attacks.

Summary

Asymmetric key cryptography, or public key encryption, provides an extremely flexible infrastructure, facilitating simple, secure communication between parties that do not necessarily know each other prior to initiating the communication. It also provides

the framework for the digital signing of messages to ensure nonrepudiation and message integrity.

This chapter explored public key encryption, which provides a scalable cryptographic architecture for use by large numbers of users. We also described some popular cryptographic algorithms, and the use of link encryption and end-to-end encryption. We introduced you to the public key infrastructure, which uses certificate authorities (CAs) to generate digital certificates containing the public keys of system users and digital signatures, which rely on a combination of public key cryptography and hashing functions. You also learned how to use the PKI to obtain integrity and nonrepudiation through the use of digital signatures. You learned how to ensure consistent security throughout the cryptographic life cycle by adopting key management practices and other mechanisms.

We also looked at some of the common applications of cryptographic technology in solving everyday problems. You learned how cryptography can be used to secure email (using PGP and S/MIME), web communications (using TLS), and both peer-to-peer and gateway-to-gateway networking (using IPSec).

Finally, we covered some of the more common attacks used by malicious individuals attempting to interfere with or intercept encrypted communications between two parties. Such attacks include cryptanalytic, replay, brute-force, known plaintext, chosen plaintext, chosen ciphertext, on-path, man-in-the-middle, and birthday attacks. It's important for you to understand these attacks in order to provide adequate security against them.

Study Essentials

Understand the key types used in asymmetric cryptography. Public keys are freely shared among communicating parties, whereas private keys are kept secret. To encrypt a message, use the recipient's public key. To decrypt a message, use your own private key. To sign a message, use your own private key. To validate a signature, use the sender's public key.

Be familiar with the three major public key cryptosystems. RSA is the most famous public key cryptosystem; it was developed by Rivest, Shamir, and Adleman in 1977. It depends on the difficulty of factoring the product of prime numbers. ElGamal is an extension of the Diffie–Hellman key exchange algorithm that depends on modular arithmetic. Elliptic curve cryptography depends on the elliptic curve discrete logarithm problem and provides more security than other algorithms when both are used with keys of the same length.

Know the fundamental requirements of a hash function. Good hash functions have five requirements. They must allow input of any length, provide fixed-length output, make it relatively easy to compute the hash function for any input, provide one-way functionality, and be collision-resistant.

Be familiar with the major hashing algorithms. The Secure Hash Algorithm SHA-3 is the government standard message digest function. SHA-2 supports variable-length message digests, ranging up to 512 bits. SHA-3 improves upon the security of SHA-2 and supports the same hash lengths.

Know how cryptographic salts improve the security of password hashing. When straightforward hashing is used to store passwords in a password file, attackers may use rainbow tables of precomputed values to identify commonly used passwords. Adding salts to the passwords before hashing them reduces the effectiveness of rainbow table attacks. Common password hashing algorithms that use key stretching to further increase the difficulty of attack include PBKDF2, bcrypt, and scrypt.

Understand how digital signatures are generated and verified. To digitally sign a message, first use a hashing function to generate a message digest; then encrypt the digest with your private key. To verify the digital signature on a message, decrypt the signature with the sender's public key and then compare the original message digest to one you generate yourself. If they match, the message is authentic.

Understand the public key infrastructure (PKI). In the public key infrastructure, certificate authorities (CAs) generate digital certificates containing the public keys of system users. Users then distribute these certificates to people with whom they want to communicate. Certificate recipients verify a certificate using the CA's public key.

Know the common applications of cryptography to secure email. The emerging standard for encrypted messages is the S/MIME protocol. Another popular email security tool is Phil Zimmermann's Pretty Good Privacy (PGP). Most users of email encryption rely on having this technology built into their email client or their web-based email service.

Know the common applications of cryptography to secure web activity. The de facto standard for secure web traffic is the use of HTTP over Transport Layer Security (TLS). This approach relies on hybrid cryptography using asymmetric cryptography to exchange an ephemeral session key, which is then used to carry on symmetric cryptography for the remainder of the session.

Know the common applications of cryptography to secure networking. The IPSec protocol standard provides a common framework for encrypting network traffic and is built into a number of common operating systems. In IPSec transport mode, packet contents are encrypted for peer-to-peer communication. In tunnel mode, the entire packet, including header information, is encrypted for gateway-to-gateway communications.

Be able to describe IPSec. IPSec is a security architecture framework that supports secure communication over IP. IPSec establishes a secure channel in either transport mode or tunnel mode. It can be used to establish direct communication between computers or to set up a VPN between networks. IPSec uses two protocols: Authentication Header (AH) and Encapsulating Security Payload (ESP).

Be able to explain common cryptographic attacks. Ciphertext-only attacks require access only to the ciphertext of a

message. One example of a ciphertext-only attack is the brute-force attack, which attempts to randomly find the correct cryptographic key. Frequency analysis, another ciphertext-only attack, counts characters in the ciphertext to reverse substitution ciphers. Known plaintext, chosen ciphertext, and chosen plaintext attacks require the attacker to have some extra information in addition to the ciphertext. The on-path attack fools both parties into communicating with the attacker instead of directly with each other. The birthday attack is an attempt to find collisions in hash functions. The replay attack is an attempt to reuse authentication requests.

Written Lab

1. Explain the process Bob should use if he wants to send a confidential message to Alice using asymmetric cryptography.
2. Explain the process Alice would use to decrypt the message Bob sent in question 1.
3. Explain the process Bob should use to digitally sign a message to Alice.
4. Explain the process Alice should use to verify the digital signature on the message from Bob in question 3.

Review Questions

1. Brian computes the digest of a single sentence of text using a SHA-2 hash function. He then changes a single character of the sentence and computes the hash value again. Which one of the following statements is true about the new hash value?

    A. The new hash value will be one character different from the old hash value.

    B. The new hash value will share at least 50 percent of the characters of the old hash value.