

Chapter 10

Understanding Cryptography and PKI

CompTIA Security+ objectives covered in this chapter:

- 1.2 Compare and contrast types of attacks.**
 - Cryptographic attacks (Downgrade, Weak implementations)
- 1.6 Explain the impact associated with types of vulnerabilities.**
 - Weak cipher suites and implementations, Improper certificate and key management
- 2.2 Given a scenario, use appropriate software tools to assess the security posture of an organization.**
 - Steganography tools
- 2.3 Given a scenario, troubleshoot common security issues.**
 - Certificate issues
- 2.6 Given a scenario, implement secure protocols.**
 - Protocols (S/MIME)
- 6.1 Compare and contrast basic concepts of cryptography.**
 - Symmetric algorithms, Modes of operation, Asymmetric algorithms, Hashing, Salt, IV, nonce, Elliptic curve, Weak/deprecated algorithms, Key exchange, Digital signatures, Diffusion, Confusion, Steganography, Obfuscation, Stream vs. block, Key strength, Session keys, Ephemeral key, Secret algorithm, Data-in-transit, Data-at-rest, Data-in-use, Random/ pseudo-random number generation, Key stretching, Implementation vs. algorithm selection (Crypto service provider, Crypto modules), Perfect forward secrecy, Security through obscurity, Common use cases (Low power devices, Low latency, High resiliency)
- 6.2 Explain cryptography algorithms and their basic characteristics.**
 - Symmetric algorithms (AES, DES, 3DES, RC4, Blowfish/Twofish), Cipher modes (CBC, GCM, ECB, CTM,

Stream vs. block), Asymmetric algorithms (RSA, DSA, Diffie-Hellman [Groups, DHE, ECDHE], Elliptic curve, PGP/GPG)

- Hashing algorithms (MD5, SHA, HMAC, RIPEMD)
- Key stretching algorithms (BCRYPT, PBKDF2)
- Obfuscation (XOR, ROT13, Substitution ciphers)

6.4 Given a scenario, implement public key infrastructure.

- Components (CA, Intermediate CA, CRL, OCSP, CSR, Certificate, Public key, Private key, Object identifiers [OID]), Concepts (Online vs. offline CA, Stapling, Pinning, Trust model, Key escrow, Certificate chaining)
- Types of certificates (Wildcard, SAN, Code signing, Self-signed, Machine/computer, Email, User, Root, Domain validation, Extended validation)
- Certificate formats (DER, PEM, PFX, CER, P12, P7B)

**

Although cryptography and Public Key Infrastructure (PKI) are only 12 percent of the exam, you might find that many of these topics aren't as familiar to you as other topics, and you might have to spend more than 12 percent of your study time here. When tackling these topics, don't lose sight of the basics. The first section in this chapter, "Introducing Cryptography Concepts," outlines and summarizes these basics. Other sections dig into the details of hashing, encryption, and Public Key Infrastructure (PKI) components.

Introducing Cryptography Concepts

Cryptography has several important concepts that you need to grasp for the CompTIA Security+ exam, but the topics are often new to many information technology (IT) professionals. Two core topics are integrity and confidentiality, introduced in Chapter 1, "Mastering Security Basics."

As an introduction, the following points identify the important core cryptography concepts. Remember, this is only an overview. If these bullets don't make sense to you now, they should after you complete this chapter:

- **Integrity** provides assurances that data has not been modified.

Hashing ensures that data has retained integrity.

- A **hash** is a number derived from performing a calculation on data, such as a message, patch, or file.

- Hashing creates a fixed-size string of bits or hexadecimal characters, which cannot be reversed to re-create the original data.
- Common hashing algorithms include MD5 and Secure Hash Algorithm (SHA).
- **Confidentiality** ensures that data is only viewable by authorized users. Encryption protects the confidentiality of data.
 - **Encryption** scrambles, or ciphers, data to make it unreadable if intercepted. Encryption normally includes an algorithm and a key.
 - Symmetric encryption uses the same key to encrypt and decrypt data.
 - Asymmetric encryption uses two keys (public and private) created as a matched pair.
 - Asymmetric encryption requires a Public Key Infrastructure (PKI) to issue certificates.
 - Anything encrypted with the public key can only be decrypted with the matching private key.
 - Anything encrypted with the private key can only be decrypted with the matching public key.
- Stream ciphers encrypt data 1 bit at a time. Block ciphers encrypt data in blocks.
- Steganography provides a level of confidentiality by hiding data within other files. For example, it's possible to embed data within the white space of a picture file.
- A **digital signature** provides authentication, non-repudiation, and integrity.
 - **Authentication** validates an identity.
 - **Non-repudiation** prevents a party from denying an action.
 - Users sign emails with a digital signature, which is a hash of an email message encrypted with the sender's private key.
 - Only the sender's public key can decrypt the hash, providing verification it was encrypted with the sender's private key.

Providing Integrity with Hashing

You can verify integrity with hashing. Hashing is an algorithm performed on data such as a file or message to produce a number called a hash (sometimes called a checksum). The hash is used to verify that data is not modified, tampered with, or corrupted. In other words, you can verify the data has maintained integrity.

A key point about a hash is that no matter how many times you execute the hashing algorithm against the data, the hash will always be the same if the data is the same.

Hashes are created at least twice so that they can be compared. For example, imagine a software company is releasing a patch for an application that customers can download. They can calculate the hash of the patch and post both a link to the patch file and the hash on the company site. They might list it as:

- **Patch file.** Patch_v2_3.zip
- **SHA-1 checksum.**

d4723ac6f72daea2c7793ac113863c5082644229

The Secure Hash Algorithm 1 (SHA-1) checksum is the calculated hash displayed in hexadecimal. Customers can download the file and then calculate the hash on the downloaded file. If the calculated hash is the same as the hash posted on the web site, it verifies the file has retained integrity. In other words, the file has not changed.

Remember this

Hashing verifies integrity for data such as email, downloaded files, and files stored on a disk. A hash is a number created with a hashing algorithm, and is sometimes listed as a checksum.

MD5

Message Digest 5 (**MD5**) is a common hashing algorithm that produces a 128-bit hash. Hashes are commonly shown in hexadecimal format instead of a stream of 1s and 0s. For example, an MD5 hash is displayed as 32 hexadecimal characters instead of 128 bits. Hexadecimal characters are composed of 4 bits and use the numbers 0 through 9 and the

characters a through f.

MD5 has been in use since 1992. Experts discovered significant vulnerabilities in MD5 in 2004 and later years. As processing power of computers increased, it became easier and easier to exploit these vulnerabilities. Security experts now consider it cracked and discourage its use.

However, it is still widely used to verify the integrity of files. This includes email, files stored on disks, files downloaded from the Internet, executable files, and more. The “Hashing Files” section shows how you can manually calculate hashes.

SHA

Secure Hash Algorithm (**SHA**) is another hashing algorithm. There are several variations of SHA grouped into four families—SHA-0, SHA-1, SHA-2, and SHA-3:

- SHA-0 is not used.
- SHA-1 is an updated version that creates 160-bit hashes. This is similar to the MD5 hash except that it creates 160-bit hashes instead of 128-bit hashes.
- SHA-2 improved SHA-1 to overcome potential weaknesses. It includes four versions. SHA-256 creates 256-bit hashes and SHA-512 creates 512-bit hashes. SHA-224 (224-bit hashes) and SHA-384 (384-bit hashes) create truncated versions of SHA-256 and SHA-512, respectively.
- SHA-3 (previously known as Keccak) is an alternative to SHA-2. The U.S. National Security Agency (NSA) created SHA-1 and SHA-2. SHA-3 was created outside of the NSA and was selected in a non-NSA public competition. It can create hashes of the same size as SHA-2 (224 bits, 256 bits, 384 bits, and 512 bits).

Just as MD5 is used to verify the integrity of files, SHA also verifies file integrity. As an example, it’s rare for executable files to be modified. However, some malware modifies executable files by adding malicious code into the file. Rootkits will often modify system-level files.

Some host-based intrusion detection systems (HIDSs) and antivirus software capture hashes of files on a system when they first scan it and include valid hashes of system files in signature definition files. When they

scan a system again, they can capture hashes of executable and system files and compare them with known good hashes. If the hashes are different for an executable or system file, it indicates the file has been modified, and it may have been modified by malware.

HMAC

Another method used to provide integrity is with a Hash-based Message Authentication Code (***HMAC***). An HMAC is a fixed-length string of bits similar to other hashing algorithms such as MD5 and SHA-1 (known as HMAC-MD5 and HMAC-SHA1). However, HMAC also uses a shared secret key to add some randomness to the result and only the sender and receiver know the secret key.

For example, imagine that one server is sending a message to another server using HMAC- MD5. It starts by first creating a hash of a message with MD5 and then uses a secret key to complete another calculation on the hash. The server then sends the message and the HMAC-MD5 hash to the second server. The second server performs the same calculations and compares the received HMAC-MD5 hash with its result. Just as with any other hash comparison, if the two hashes are the same, the message retained integrity, but if the hashes are different, the message lost integrity.

The HMAC provides both integrity and authenticity of messages. The MD5 portion of the hash provides integrity just as MD5 does. However, because only the server and receiver know the secret key, if the receiver can calculate the same HMAC-MD5 hash as the sender, it knows that the sender used the same key. If an attacker was trying to impersonate the sender, the message wouldn't pass this authenticity check because the attacker wouldn't have the secret key. Internet Protocol security (IPsec) and Transport Layer Security (TLS) often use a version of HMAC such as HMAC-MD5 and HMAC-SHA1.

Remember this

Two popular hashing algorithms used to verify integrity are MD5 and SHA. HMAC verifies both the integrity and authenticity of a message with the use of a shared secret. Other protocols such as IPsec and TLS use HMAC-MD5 and HMAC-SHA1.

RIPEMD

RACE Integrity Primitives Evaluation Message Digest (**RIPEMD**) is another hash function used for integrity, though it isn't as widely used as MD5, SHA, and HMAC. Different versions create different size hashes. RIPEMD-160 creates 160-bit, fixed-size hashes. Other versions create hash sizes of 128 bits, 256 bits, and 320 bits.

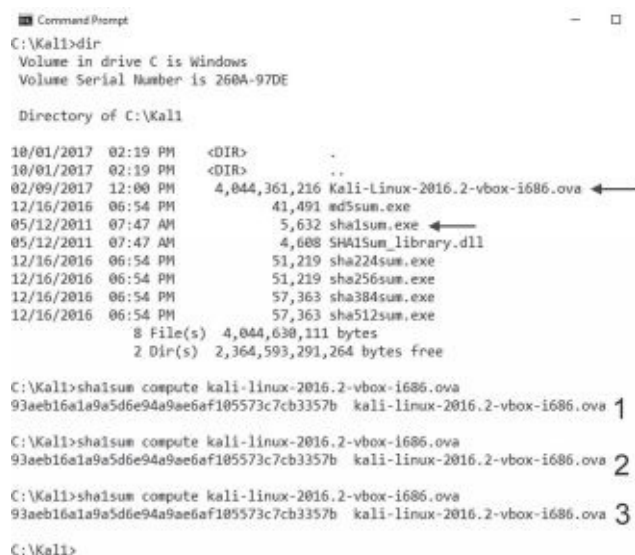
Hashing Files

Many applications calculate and compare hashes automatically without any user intervention. For example, digital signatures (described later) use hashes within email, and email applications automatically create and compare the hashes.

Additionally, there are several applications you can use to manually calculate hashes. As an example, *sha1sum.exe* is a free program anyone can use to create hashes of files. A Google search on “download sha1sum” will show several locations. It runs the SHA-1 hashing algorithm against a file to create the hash.

If you downloaded a Kali Linux distribution image from the lab mentioned in Chapter 1, you would have seen hashes of the different images posted on the web site. The web site indicates that the SHA-1 hash for the file I downloaded is:

93AEB16A1A9A5D6E94A9AE6AF105573C7CB3357B.



```
Command Prompt
C:\Kali>dir
Volume in drive C is Windows
Volume Serial Number is 260A-97DE

Directory of C:\Kali

10/01/2017  02:19 PM  <DIR>          .
10/01/2017  02:19 PM  <DIR>          ..
02/09/2017  12:00 PM         4,044,361,216 Kali-Linux-2016.2-vbox-i686.ova
12/16/2016  06:54 PM         41,491 md5sum.exe
05/12/2011  07:47 AM           5,632 sha1sum.exe
05/12/2011  07:47 AM           4,608 SHA1sum_library.dll
12/16/2016  06:54 PM           51,219 sha224sum.exe
12/16/2016  06:54 PM           51,219 sha256sum.exe
12/16/2016  06:54 PM           57,363 sha384sum.exe
12/16/2016  06:54 PM           57,363 sha512sum.exe
               8 File(s)  4,044,630,111 bytes
               2 Dir(s)  2,364,593,291,264 bytes free

C:\Kali>sha1sum compute kali-linux-2016.2-vbox-i686.ova
93aeb16a1a9a5d6e94a9ae6af105573c7cb3357b  kali-linux-2016.2-vbox-i686.ova 1
C:\Kali>sha1sum compute kali-linux-2016.2-vbox-i686.ova
93aeb16a1a9a5d6e94a9ae6af105573c7cb3357b  kali-linux-2016.2-vbox-i686.ova 2
C:\Kali>sha1sum compute kali-linux-2016.2-vbox-i686.ova
93aeb16a1a9a5d6e94a9ae6af105573c7cb3357b  kali-linux-2016.2-vbox-i686.ova 3
C:\Kali>
```

By running the sha1sum

command against the file, I can calculate the hash of the file I downloaded. I first used the `dir` command to list the files in the directory. I then ran `sha1sum` against the Kali Linux file three times. Each time, `sha1sum` created the same hash `93aeb16a1a9a5d6e94a9ae6af105573c7cb3357b`, as shown in Figure 10.1

Figure 10.1: Calculating a hash with sha1sum

Figure 10.1 demonstrates two important points:

- **The hash will always be the same no matter how many times you calculate it.**

In the figure, I ran `sha1sum` three times, but it would give me the same result if I ran it 3,000 times.

- **Hashing verifies the file has retained integrity.**

Because the calculated hash is the same as the hash posted on the download site, it verifies the file has not lost integrity.

In contrast, if `sha1sum` created a different hash than the one posted on the web site, I'd know that the file lost integrity. I wouldn't necessarily know *why* the file lost integrity. An attacker might have infected it with malware, or it might have lost a bit or two during the transfer. However, I would know that the integrity of the file was lost and the file should not be trusted.

It's worth stressing that hashes are one-way functions. In other words,

you can calculate a hash on a file or a message, but you can't use the hash to reproduce the original data. The hashing algorithms always create a fixed-size bit string regardless of the size of the original data. The hash doesn't give you a clue about the size of the file, the type of the file, or anything else.

As an example, the SHA-1 hash from the message "I will pass the Security+ exam" is: `765591c4611be5e03bea41882ffdaa159352cf49`. However, you can't look at the hash and identify the message, or even know that it is a hash of a six-word message. Similarly, you can't look at the hash shown in Figure 10.1 and know that it was calculated from a 3.8 GB executable file.

If you want to work with hashes yourself, check out the hashing and checksum labs in the online resources for this book at <http://gcgapremium.com/501labs/>.

Hashing Passwords

Passwords are often stored as hashes. When a user creates a new password, the system calculates the hash for the password and then stores the hash. Later, when the user authenticates by entering a username and password, the system calculates the hash of the entered password, and then compares it with the stored hash. If the hashes are the same, it indicates that the user entered the correct password.

Remember this

Hashing is a one-way function that creates a string of characters. You cannot reverse the hash to re-create the original file. Passwords are often stored as hashes instead of storing the actual password. Additionally, applications often salt passwords with extra characters before hashing them.

Key Stretching

As mentioned in Chapter 7, "Protecting Against Advanced Attacks," many password attacks attempt to discover a password by calculating a hash on a guessed password, and then comparing it with the stored hash of the password. Using complex passwords goes a long way toward preventing

these types of attacks but doesn't prevent them all.

Key stretching (sometimes called key strengthening) is a technique used to increase the strength of stored passwords and can help thwart brute force and rainbow table attacks. Key stretching techniques **salt** the passwords with additional random bits to make them even more complex. Two common key stretching techniques are bcrypt and Password-Based Key Derivation Function 2 (PBKDF2).

Bcrypt is based on the Blowfish block cipher and is used on many Unix and Linux distributions to protect the passwords stored in the shadow password file. Bcrypt salts the password by adding additional random bits before encrypting it with Blowfish. Bcrypt can go through this process multiple times to further protect against attempts to discover the password. The result is a 60-character string.

As an example, if your password is IL0ve\$ecurity, an application can encrypt it with bcrypt and a random salt. It might look like this, which the application stores in a database:

\$2b\$12\$HXIKtJr93DH59BzzKQhehOI9pGjRA/03ENcFRby1jH7nXwt17

Later, when a user authenticates with a username and password, the application runs bcrypt on the supplied password and compares it with the stored bcrypt-encrypted password. If the bcrypt result of the supplied password is the same as the stored bcrypt result, the user is authenticated.

As an added measure, it's possible to add some pepper to the salt to further randomize the bcrypt string. In this context, the pepper is another set of random bits stored elsewhere.

PBKDF2 uses salts of at least 64 bits and uses a pseudo-random function such as HMAC to protect passwords. Many algorithms such as Wi-Fi Protected Access II (WPA2), Apple's iOS mobile operating system, and Cisco operating systems use PBKDF2 to increase the security of passwords. Some applications send the password through the PBKDF2 process as many as 1,000,000 times to create the hash. The size of the resulting hash varies with PBKDF2 depending on how it is implemented. Bit sizes of 128 bits, 256 bits, and 512 bits are most common.

Some security experts believe that PBKDF2 is more susceptible to brute force attacks than bcrypt. A public group created the Password Hashing Competition (PHC). They received and evaluated 24 different hashing algorithms as alternatives. In July 2015, the PHC selected Argon2 as the winner of the competition and recommended it be used instead of legacy

algorithms such as PBKDF2.

Remember this

Bcrypt and PBKDF2 are key stretching techniques that help prevent brute force and rainbow table attacks. Both salt the password with additional random bits.

Hashing Messages

Hashing provides integrity for messages. It provides assurance to someone receiving a message that the message has not been modified. Imagine that Lisa is sending a message to Bart, as shown in Figure 10.2. The message is “The price is \$75.” This message is not secret, so there is no need to encrypt it. However, we do want to provide integrity, so this explanation is focused only on hashing.

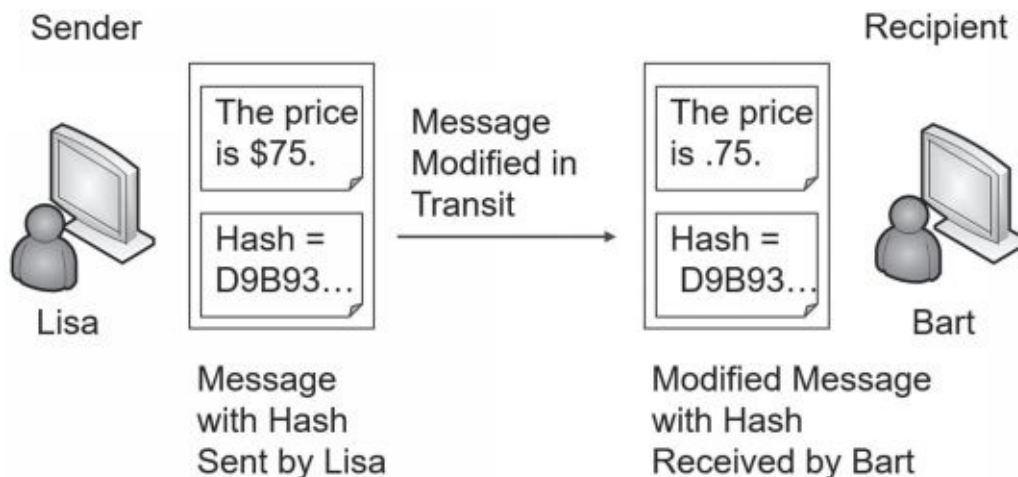


Figure 10.2: Simplified hash process

An application on Lisa’s computer calculates the MD5 hash as:
D9B93C99B62646ABD06C887039053F56.

In the figure, I’ve shortened the full hash down to just the first five characters of “D9B93.” Lisa then sends both the message and the hash to Bart.

In this example, something modified the message before it reaches Bart. When Bart receives the message and the original hash, the message is now “The price is .75.” Note that the message is modified in transit, but the hash is *not* modified.

A program on Bart's computer calculates the MD5 hash on the received message as `564294439E1617F5628A3E3EB75643FE`. It then compares the received hash with the calculated hash:

- Hash created on Lisa's computer, and received by Bart's computer:
`D9B93C99B62646ABD06C887039053F56`
- Hash created on Bart's computer:
`564294439E1617F5628A3E3EB75643FE`

Clearly, the hashes are different, so you know the message lost integrity. The program on Bart's computer would report the discrepancy. Bart doesn't know what caused the problem. It could have been a malicious attacker changing the message, or it could have been a technical problem. However, Bart does know the received message isn't the same as the sent message and he shouldn't trust it.

Using HMAC

You might have noticed a problem in the explanation of the hashed message. If an attacker can change the message, why can't the attacker change the hash, too? In other words, if Hacker Harry changed the message to "The price is .75," he could also calculate the hash on the modified message and replace the original hash with the modified hash. Here's the result:

- Hash created on Lisa's computer:
`D9B93C99B62646ABD06C887039053F56`
- Modified hash inserted by attacker after modifying the message:
`564294439E1617F5628A3E3EB75643FE`
- Hash created for modified message on Bart's computer:
`564294439E1617F5628A3E3EB75643FE`

The calculated hash on the modified message would be the same as the received hash. This erroneously indicates that the message maintained integrity. HMAC helps solve this problem.

With HMAC, both Lisa and Bart's computers would know the same secret key and use it to create an HMAC-MD5 hash instead of just an MD5 hash. Figure 10.3 shows the result.

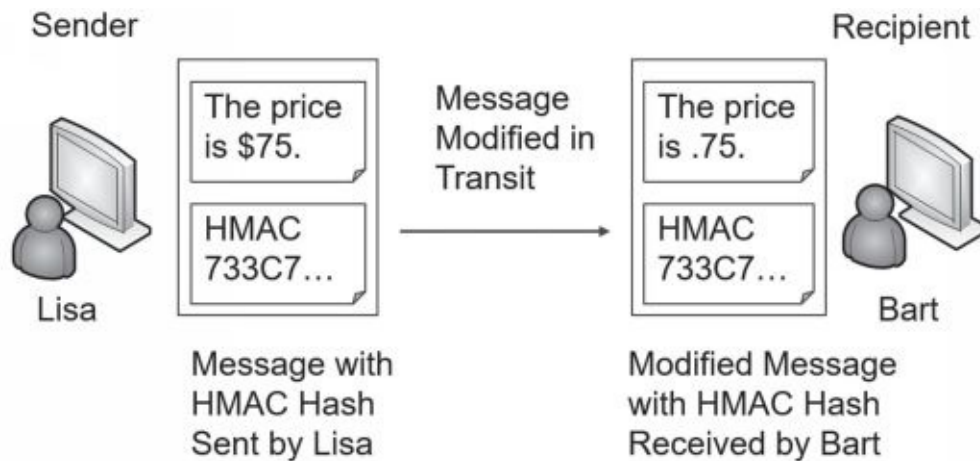


Figure 10.3: Using HMAC

Lisa is still sending the same message.

The MD5 hash is *D9B93C99B62646ABD06C887039053F56*. However, after applying the HMAC secret key, the HMAC-MD5 hash is *733C70A54A13744D5C2C9C4BA3B15034*. For brevity, I shortened this to only the first five characters (733C7) in the figure.

An attacker can modify the message in transit just as before. However, the attacker doesn't know the secret key, so he can't calculate the HMAC hash.

Bart's computer calculates the HMAC-MD5 hash on the received message using the shared secret key. It then compares the calculated hash with the hash received from Lisa:

- HMAC-MD5 hash created on Lisa's computer:
733C70A54A13744D5C2C9C4BA3B15034
- HMAC-MD5 hash created on Bart's computer:
1B4FF0F6C04434BF97F1E3DDD4B6C137

Again, you can see that the hashes are different and the message has lost integrity. If the messages weren't modified, the HMAC-MD5 hash would be the same.

Table 10.1 summarizes the important hashing protocols covered on the CompTIA Security+ exam.

Algorithm	Type	Comments
MD5	Hashing - Integrity	Creates 128-bit hashes
SHA-1	Hashing - Integrity	Creates 160-bit hashes
SHA-2	Hashing - Integrity	Creates 224-, 256-, 384-, or 512-bit hashes
SHA-3	Hashing - Integrity	Creates 224-, 256-, 384-, or 512-bit hashes
HMAC-MD5	Integrity/Authenticity	Creates 128-bit hashes
HMAC-SHA1	Integrity/Authenticity	Creates 160-bit hashes

Table 10.1: Hashing protocols

Remember this

If you can recognize the hashing algorithms such as MD5, SHA, and HMAC, it will help you answer many exam questions. For example, if a question asks what you would use to encrypt data and it lists hashing algorithms, you can quickly eliminate them because hashing algorithms don't encrypt data.

Providing Confidentiality with Encryption

Encryption provides confidentiality and prevents unauthorized disclosure of data. Encrypted data is in a ciphertext format that is unreadable. Attackers can't read encrypted traffic sent over a network or encrypted data stored on a system. In contrast, if data is sent in cleartext, an attacker can capture and read the data using a protocol analyzer.

Data-at-rest refers to any data stored on media and it's common to encrypt sensitive data. For example, it's possible to encrypt individual fields in a database (such as the fields holding customer credit card data), individual files, folders, or a full disk.

Data-in-transit refers to any data sent over a network and it's common to encrypt sensitive data-in-transit. For example, e-commerce web sites commonly use Hypertext Transfer Protocol Secure (HTTPS) sessions to encrypt transactions that include credit card data. If attackers intercept the transmissions, they only see ciphertext.

Data-in-use refers to data being used by a computer. Because the

computer needs to process the data, it is not encrypted while in use. If the data is encrypted, an application will decrypt it and store it in memory while in use. If the application changes the data, it will encrypt it again before saving it. Additionally, applications usually take extra steps to purge memory of sensitive data after processing it.

The two primary encryption methods are symmetric and asymmetric. Symmetric encryption encrypts and decrypts data with the same key. Asymmetric encryption encrypts and decrypts data using a matched key pair of a public key and a private key.

These encryption methods include two elements:

- **Algorithm.** The algorithm performs mathematical calculations on data. The algorithm is always the same.
- **Key.** The key is a number that provides variability for the encryption. It is either kept private and/or changed frequently.

Remember this

Encryption provides confidentiality and helps ensure that data is viewable only by authorized users. This applies to any data-at-rest (such as data stored in a database) or data- in-transit being sent over a network.

Encryption Terms

There are several terms within cryptography that are important to grasp. Understanding these terms makes it much easier to understand many of the more advanced concepts:

- **Random and pseudo-random numbers.** Many encryption schemes require a random or pseudo-random number as an input. If you're able to pick a number completely by chance, it is a random number. However, computers don't do things by chance, so it's often difficult to get a true random number. Instead, computers often use techniques to obtain pseudo-random numbers. A pseudo-random number appears to be random, but it is created by a deterministic algorithm. In other words, given the same input, a pseudo-random number generator will produce the same output.
- **IV.** An initialization vector (IV) provides a starting value for a

cryptographic algorithm. It is a fixed-size random or pseudo-random number that helps create random encryption keys. Ideally, the IV should be large enough so that the algorithm doesn't reuse the same IV and re-create the same encryption keys.

- **Nonce.** A *nonce* is a number used once. For example, an IV should be large enough so that it is only used once. Many cryptographic algorithms use a random or pseudo-random nonce as a seed or a starting number.
- **XOR.** XOR is a logical operation used in some encryption schemes. *XOR* operations compare two inputs. If the two inputs are the same, it outputs True (or a binary 1). If the two inputs are different, it outputs False (or a binary 0).
- **Confusion.** In the context of encryption, *confusion* means that the ciphertext is significantly different than the plaintext. As an example, when encrypting the words "I passed!" with Advanced Encryption Standard (AES), it results in the following text:
nkm6olLdchB049LbrCpL5Q.
- **Diffusion.** Effective *diffusion* techniques ensure that small changes in the plaintext result in large changes in the ciphertext. Just changing a single character in the plaintext results in completely different ciphertext. As an example, when encrypting "I passed." with AES, it results in the following text:
k/ljn+j6WBxAIKHN6IONBA. Compare this with the ciphertext from "I passed!" and you can see how changing only one character in the plaintext completely changed the ciphertext, indicating that AES is using strong diffusion techniques.
- **Secret algorithm.** A secret algorithm is one that is kept private. Security experts discourage this practice because it prevents a review of the algorithm by experts and mathematicians. Vigorous reviews can discover any flaws or potential weaknesses. In contrast, most algorithms are published and known.
- **Weak/deprecated algorithms.** A weak algorithm can be cracked, allowing an attacker to easily convert ciphertext back to plaintext. When flaws are discovered in algorithms, experts and authorities recommend deprecating the weak algorithm. As an example, web sites commonly used Secure Sockets Layer (SSL) to encrypt HTTPS sessions on the Internet. However, experts discovered

flaws in SSL and most authorities have deprecated the use of SSL and recommend the use of TLS instead.

- **High resiliency.** A common use case for encryption algorithms is to provide high resiliency. Within cryptography, high resiliency refers to the security of an encryption key even if an attacker discovers part of the key. Ideally, keys are not susceptible to leakage, preventing attackers from gaining information on any part of a key. However, there are many situations that can cause leakage. A strong algorithm implements high- resiliency techniques that ensure this leakage does not compromise the encryption key.

Remember this

Random numbers are picked by chance. Pseudo-random numbers appear to be random but are created by deterministic algorithms, meaning that given the same input, a pseudo-random number generator will create the same output. In cryptology, confusion indicates that the ciphertext is significantly different than the plaintext. Diffusion cryptographic techniques ensure that small changes in the plaintext result in significant changes in the ciphertext.

Block Versus Stream Ciphers

Most symmetric algorithms use either a block cipher or a stream cipher. They are both symmetric, so they both use the same key to encrypt or decrypt data. However, they divide data in different ways.

A **block cipher** encrypts data in specific-sized blocks, such as 64-bit blocks or 128-bit blocks. The block cipher divides large files or messages into these blocks and then encrypts each individual block separately. A **stream cipher** encrypts data as a stream of bits or bytes rather than dividing it into blocks.

In general, stream ciphers are more efficient than block ciphers when the size of the data is unknown or sent in a continuous stream, such as when streaming audio and video over a network. Block ciphers are more efficient when the size of the data is known, such as when encrypting a file or a specific-sized database field.

An important principle when using a stream cipher is that encryption keys should never be reused. If a key is reused, it is easier to crack the encryption.

Remember this

Stream ciphers encrypt data a single bit, or a single byte, at a time in a stream. Block ciphers encrypt data in a specific-sized block such as 64-bit or 128-bit blocks. Stream ciphers are more efficient than block ciphers when encrypting data in a continuous stream.

Cipher Modes

Block ciphers can use a variety of different modes of operation. It's important to have a basic understanding of these modes when choosing cipher suites.

The Electronic Codebook (**ECB**) mode of operation is the simplest cipher mode mentioned in this section. Algorithms that use ECB divide the plaintext into blocks and then encrypt each block using the same key. This represents a significant weakness. If any of the plaintext blocks are the same, the resulting ciphertext is the same, making it much easier to crack. ECB is not recommended for use in any cryptographic protocols today.

Cipher Block Chaining (**CBC**) mode is used by some symmetric block ciphers. It uses an IV for randomization when encrypting the first block. It then combines each subsequent block with the previous block using an XOR operation. Because encryption of each block is dependent on the encryption of all previous blocks, CBC sometimes suffers from pipeline delays, making it less efficient than some other modes.

Counter (**CTM**) mode effectively converts a block cipher into a stream cipher. It combines an IV with a counter and uses the result to encrypt each plaintext block. Each block uses the same IV, but CTM combines it with the counter value, resulting in a different encryption key for each block. Multiprocessor systems can encrypt or decrypt multiple blocks at the same time, allowing the algorithm to be quicker on multiprocessor or multicore systems. CTM is widely used and respected as a secure mode of operation.

It's worthwhile noting that the CompTIA objectives list CTM and include CTM in the acronym list as Counter-Mode. However, it's much more common to see it listed as CTR or CM.

Galois/Counter Mode (**GCM**) is a mode of operation used by many block ciphers. It combines the Counter mode of operation with the Galois mode of authentication. Note that it doesn't authenticate users or systems, but instead provides data authenticity (integrity) and confidentiality. In addition to encrypting the data for confidentiality, it includes hashing techniques for integrity. It is widely used due to its efficiency and performance, allowing systems to quickly encrypt and decrypt data.

Remember this

The Electronic Codebook (ECB) mode of operation is deprecated and should not be used. Cipher Block Chaining (CBC) mode combines each block with the previous block when encrypting data and sometimes suffers from pipeline delays. Counter (CTM) mode combines an IV with a counter to encrypt each block. Galois/Counter Mode (GCM) combines Counter mode with hashing techniques for integrity.

Symmetric Encryption

Symmetric encryption uses the same key to encrypt and decrypt data. In other words, if you encrypt data with a key of three, you decrypt it with the same key of three. Symmetric encryption is also called secret-key encryption or session-key encryption.

As a simple example, when I was a child, a friend and I used to pass encoded messages back and forth to each other. Our algorithm was:

- **Encryption algorithm.** Move X spaces forward to encrypt.
- **Decryption algorithm.** Move X spaces backward to decrypt.

On the way to school, we would identify the key (X) we would use that day. For example, we may have used the key of three one day. If I wanted to encrypt a message, I would move each character three spaces forward, and he would decrypt the message by moving three spaces backward.

Imagine the message "PASS" needs to be sent:

- Three characters past "P" is "S"—Start at P (Q, R, S)
- Three characters past "A" is "D"—Start at A (B, C, D)
- Three characters past "S" is "V"—Start at S (T, U, V)
- Three characters past "S" is "V"—Start at S (T, U, V)

The encrypted message is SDVV. My friend decrypted it by moving backward three spaces and learned that “PASS” was the original message.

We were using a simple substitution cipher. A **substitution cipher** replaces **plaintext** with **ciphertext** using a fixed system. In the example, “PASS” is the plaintext, “SDVV” is the ciphertext, and the fixed system is three letters.

The **ROT13** (short for rotate 13 places) cipher uses the same substitution algorithm, but always uses a key of 13. To encrypt a message, you would rotate each letter 13 spaces. To decrypt a message, you would rotate each letter 13 spaces. However, because ROT13 uses both the same algorithm and the same key, it doesn’t provide true encryption but instead just obfuscates the data.

Obfuscation methods attempt to make something unclear or difficult to understand. This is sometimes referred to as security through obscurity. However, this is rarely a reliable method of maintaining security.

Rotating letters with a different key shows how symmetric encryption uses the same key for encryption and decryption. If I encrypted the message with a key of three, my friend wouldn’t be able to decrypt it with anything but a key of three. It also helps to demonstrate an algorithm and a key, though it is admittedly simple.

Sophisticated symmetric encryption techniques use the same components of an algorithm and a key. However, the algorithms and keys are much more complex. For example, the Advanced Encryption Standard (AES) symmetric algorithm typically uses 128-bit keys, but can use keys with 256 bits.

Imagine two servers sending encrypted traffic back and forth to each other using AES symmetric encryption. They both use the same AES algorithm and the same key for this data. The data is encrypted on one server with AES and a key, sent over the wire or other transmission medium, and the same key is used to decrypt it on the other server. Similarly, if a database includes encrypted data, the key used to encrypt data is the same key used to decrypt data.

However, symmetric encryption doesn’t use the same key to encrypt and decrypt all data. For example, my friend and I used a different key each day. On the way to school, we decided on a key to use for that day. The next day, we picked a different key. If someone cracked our code yesterday, they couldn’t easily crack our code today.

Symmetric encryption algorithms change keys much more often than once a day. For example, imagine an algorithm uses a key of 123 to encrypt a project file. It could then use a key of 456 to encrypt a spreadsheet file. The key of 123 can only decrypt the project file and the key of 456 can only decrypt the spreadsheet file.

On the other hand, if symmetric encryption always used the same key of 123, it would add vulnerabilities. First, when keys are reused, the encryption is easier to crack. Second, once the key is cracked, all data encrypted with this key is compromised. If attackers discover the key of 123, not only would they have access to the project file, but they would also have access to the spreadsheet file and any other data encrypted with this same key.

As a more realistic example, Chapter 4, “Securing Your Network,” describes how Remote Authentication Dial-In User Service (RADIUS) encrypts password packets. RADIUS uses shared keys for symmetric encryption. When users authenticate, RADIUS servers and clients use the shared key to encrypt and decrypt data exchanged in a challenge/response session. Without the shared key, clients are unable to decrypt the data and respond appropriately.

Comparing Symmetric Encryption to a Door Key (Sidebar)

Occasionally, security professionals compare symmetric keys to a house key and this analogy helps some people understand symmetric encryption a little better. For example, imagine Marge moves into a new home. She’ll receive a single key that she can use to lock and unlock her home. Of course, Marge can’t use this key to unlock her neighbor’s home.

Later, Marge marries Homer, and Homer moves into Marge’s home. Marge can create a copy of her house key and give it to Homer. Homer can now use that copy of the key to lock and unlock the house. By sharing copies of the same key, it doesn’t matter whether Marge or Homer is the one who locks the door; they can both unlock it.

Similarly, symmetric encryption uses a single key to encrypt

and decrypt data. If a copy of the symmetric key is shared, others who have the key can also encrypt and decrypt data.

Remember this

Symmetric encryption uses the same key to encrypt and decrypt data. For example, when transmitting encrypted data, symmetric encryption algorithms use the same key to encrypt and decrypt data at both ends of the transmission media.

RADIUS uses symmetric encryption.

AES

The Advanced Encryption Standard (**AES**) is a strong symmetric block cipher that encrypts data in 128-bit blocks. The National Institute of Standards and Technology (NIST) adopted AES from the Rijndael encryption algorithm after a lengthy evaluation of several different algorithms. NIST is a U.S. agency that develops and promotes standards. They spent about five years conducting a review of 15 different symmetric algorithms and identified AES as the best of the 15.

AES can use key sizes of 128 bits, 192 bits, or 256 bits, and it's sometimes referred to as AES-128, AES-192, or AES-256 to identify how many bits are used in the key. When more bits are used, it makes it more difficult to discover the key and decrypt the data. AES-128 provides strong protection, but AES-256 provides stronger protection.

In general, the size of the key for any encryption directly corresponds to the key strength.

Longer keys for a specific algorithm result in stronger key strength.

Because of its strengths, AES has been adopted in a wide assortment of applications. For example, many applications that encrypt data on USB drives use AES. Some of the strengths of AES are:

- **Fast.** AES uses elegant mathematical formulas and only requires one pass to encrypt and decrypt data. In contrast, 3DES (mentioned later in this chapter) requires multiple passes to encrypt and decrypt data.
- **Efficient.** AES is less resource intensive than other encryption algorithms such as 3DES. AES encrypts and decrypts quickly even when ciphering data on small devices, such as USB flash drives.

- **Strong.** AES provides strong encryption of data, providing a high level of confidentiality.

DES

Data Encryption Standard (**DES**) is a symmetric block cipher that was widely used for many years, dating back to the 1970s. It encrypts data in 64-bit blocks. However, it uses a relatively small key of only 56 bits and can be broken with brute force attacks. In the '70s, the technology required to break 56-bit encryption wasn't easily available, but with the advances in computer technology, a 56-bit key is now considered trivial. DES is not recommended for use today.

3DES

3DES (pronounced as “Triple DES”) is a symmetric block cipher designed as an improvement over the known weaknesses of DES. In basic terms, it encrypts data using the DES algorithm in three separate passes and uses multiple keys. Just as DES encrypts data in 64-bit blocks, 3DES also encrypts data in 64-bit blocks.

Although 3DES is a strong algorithm, it isn't used as often as AES today. AES is much less resource intensive. However, if hardware doesn't support AES, 3DES is a suitable alternative. 3DES uses key sizes of 56 bits, 112 bits, or 168 bits.

Remember this

AES is a strong symmetric block cipher that encrypts data in 128-bit blocks. AES uses 128-bit, 192-bit, or 256-bit keys. DES and 3DES are block ciphers that encrypt data in 64-bit blocks. 3DES was originally designed as a replacement for DES, but NIST selected AES as the current standard. However, 3DES is still used in some applications, such as when legacy hardware doesn't support AES.

RC4

Ron Rivest invented several versions of RC, which are sometimes referred to as Ron's Code or Rivest Cipher. The most commonly used version is **RC4** (also called ARC4), which is a symmetric stream cipher and it can use between 40 and 2,048 bits.

RC4 has enjoyed a long life as a strong cipher. For many years, it has been the recommended encryption mechanism in SSL and TLS, when encrypting HTTPS connections on the Internet.

However, experts have speculated since 2013 that agencies such as the U.S. National Security Agency (NSA) can break RC4, even when implemented correctly such as in TLS. Because of this, companies such as Microsoft recommend disabling RC4 and using AES instead. Even though AES is a block cipher and RC4 is a stream cipher, TLS can implement either one.

Blowfish and Twofish

Blowfish is a strong symmetric block cipher that is still widely used today. It encrypts data in 64-bit blocks and supports key sizes between 32 and 448 bits. Bruce Schneier (a widely respected voice in IT security) designed Blowfish as a general-purpose algorithm to replace DES.

Interestingly, Blowfish is actually faster than AES in some instances. This is especially true when comparing Blowfish with AES-256. Part of the reason is that Blowfish encrypts data in smaller 64-bit blocks, whereas AES encrypts data in 128-bit blocks.

Twofish is related to Blowfish, but it encrypts data in 128-bit blocks and it supports 128-, 192-, or 256-bit keys. It was one of the finalist algorithms evaluated by NIST for AES. However, NIST selected Rijndael as AES instead.

Remember this

RC4 is a strong symmetric stream cipher, but most experts recommend using AES instead today. Blowfish is a 64-bit block cipher and Twofish is a 128-bit block cipher. Although NIST recommends AES as the standard, Blowfish is faster than AES-256.

Symmetric Encryption Summary

Algorithm	Type	Method	Key Size
AES	Symmetric encryption	128-bit block cipher	128-, 192-, or 256-bit key
3DES	Symmetric encryption	64-bit block cipher	56-, 112-, or 168-bit key
Blowfish	Symmetric encryption	64-bit block cipher	32- to 448-bit key
Twofish	Symmetric encryption	128-bit block cipher	128-, 192-, or 256-bit key
RC4*	Symmetric encryption	Stream cipher	40- to 2,048-bit key
DES*	Symmetric encryption	64-bit block cipher	56-bit key

Table 10.2 summarizes the important symmetric algorithms and their basic characteristics. The items marked with an asterisk (RC4 and DES) are no longer recommended for use, but are still included in the CompTIA Security+ objectives.

Table 10.2: Symmetric encryption protocols

Remember this

If you can recognize the symmetric algorithms such as AES, DES, 3DES, Blowfish, and Twofish, it will help you answer many exam questions. For example, if a question asks what you would use to hash data and it lists encryption algorithms, you can quickly eliminate them because encryption algorithms don't hash data. You should also know the size of the blocks and the size of the keys listed in Table 10.2.

Asymmetric Encryption

Asymmetric encryption uses two keys in a matched pair to encrypt and decrypt data—a public key and a private key. There are several important points to remember with these keys:

- If the **public key** encrypts information, only the matching private key can decrypt the same information.
- If the **private key** encrypts information, only the matching public key can decrypt the same information.
- Private keys are always kept private and never shared.
- Public keys are freely shared by embedding them in a shared certificate.

Remember this

Only a private key can decrypt information encrypted with a matching public key. Only a public key can decrypt information

encrypted with a matching private key. A key element of several asymmetric encryption methods is that they require a certificate and a PKI.

Although asymmetric encryption is very strong, it is also very resource intensive. It takes a significant amount of processing power to encrypt and decrypt data, especially when compared with symmetric encryption. Most cryptographic protocols that use asymmetric encryption only use it for key exchange. Key exchange is any cryptographic method used to share cryptographic keys between two entities. In this context, asymmetric encryption uses key exchange to share a symmetric key. The cryptographic protocol then uses the symmetric encryption to encrypt and decrypt data because symmetric encryption is much more efficient.

Some of the more advanced topics related to asymmetric encryption become harder to understand if you don't understand the relationship of matched public and private key pairs. However, because you can't actually see these keys, the concepts are hard to grasp for some people. The Rayburn box demonstrates how you can use physical keys for the same purposes as these public and private keys.

The Rayburn Box

I often talk about the Rayburn box in the classroom to help people understand the usage of public and private keys. A Rayburn box is a lockbox that allows people to securely transfer items over long distances. It has two keys. One key can lock the box, but can't unlock it. The other key can unlock the box, but can't lock it.

Both keys are matched to one box and won't work with other boxes:

- Only one copy of one key exists—think of it as the private key.
- Multiple copies of the other key exist, and copies are freely made and distributed—think of these as public keys.

The box comes in two different versions. In one version, it's used to send secrets in a confidential manner to prevent unauthorized disclosure. In the other version, it's used to send messages with authentication, so you know the sender sent the message and that the message wasn't modified in transit.

The Rayburn Box Used to Send Secrets

Imagine that I wanted you to send some proprietary information and a working model of an invention to me. Obviously, we wouldn't want anyone else to be able to access the information or the working model. I could send you the empty open box with a copy of the key used to lock it. You place everything in the box and then lock it with the public key I've sent with the box.

This key can't unlock the box, so even if other people had copies of the public key that I sent to you, they couldn't use it to unlock the box. When I receive the box from you, I can unlock it with the only key that will unlock it—my private key.

This is similar to how public and private keys are used to send encrypted data over the Internet to ensure confidentiality. The public key encrypts information. Information encrypted with a public key can only be decrypted with the matching private key. Many copies of the public key are available, but only one private key exists, and the private key always stays private. The “Encrypting HTTPS Traffic with TLS” section later in this chapter shows this process in more depth.

The Rayburn Box Used for Authentication

With a little rekeying of the box, I can use it to send messages while giving assurances to recipients that I sent the message. In this context, the message isn't secret and doesn't need to be protected. Instead, it's important that you know I sent the message.

When used this way, the private key will lock the Rayburn box, but it cannot unlock the box. Instead, only a matching public key can unlock it. Multiple copies of the public key exist and anyone with a public key can unlock the box. However, after unlocking the box with a matching public key, it isn't possible to lock it with the public key.

Imagine that you and I are allies in a battle. I want to give you a message of “SY0-501,” which is a code telling you to launch a specific attack at a specific time. We don't care if someone reads this message because it's a code. However, we need you to have assurances that I sent the message.

I write the message, place it in the box, and lock it with my private key. When you receive it, you can unlock it with the matching public key. Because the public key opens it, you know this is my box and it was locked with my private key—you know I sent the message.

If an enemy spy intercepted the box and opened it with the public key,

the spy wouldn't be able to lock it again using the public key, so you'd receive an open box. However, the spy could replace the message with something else. An open box with a message inside it doesn't prove I sent it. The only way you know that I sent it is if you receive a locked box that you can unlock with the matching public key.

This is similar to how digital signatures use public and private keys. The "Signing Email with Digital Signatures" section later in this chapter explains digital signatures in more depth. In short, I can send you a message digitally signed with my private key. If you can decrypt the digital signature with my matching public key, you know it was encrypted, or signed, with my private key. Because only one copy of the private key exists, and I'm the only person who can access it, you know I sent the message.

The Rayburn Box Demystified

Before you try to find a Rayburn box, let me clear something up. The Rayburn box is just a figment of my imagination. Rayburn is my middle name.

I haven't discovered a real-world example of how public/private keys work, so I've created the Rayburn box as a metaphor to help people visualize how public/private keys work. Feel free to build one if you want.

Certificates

A key element of asymmetric encryption is a certificate. A **certificate** is a digital document that typically includes the public key and information on the owner of the certificate. Certificate Authorities (CAs, explored in greater depth later in this chapter) issue and manage certificates. Certificates are used for a variety of purposes beyond just asymmetric encryption, including authentication and digital signatures.

Figure 10.4 shows a sample certificate with the public key selected. Users and applications share the certificate file to share the public key. They do not share the private key.



Figure 10.4: Certificate with public key selected

There is much more information in the certificate than just the public key, but not all of it is visible in the figure. Common elements within a certificate include:

- **Serial number.** The serial number uniquely identifies the certificate. The CA uses this serial number to validate a certificate. If the CA revokes the certificate, it publishes this serial number in a certificate revocation list (CRL).
- **Issuer.** This identifies the CA that issued the certificate.
- **Validity dates.** Certificates include “Valid From” and “Valid To” dates. This ensures a certificate expires at some point.
- **Subject.** This identifies the owner of the certificate. In the figure, it identifies the subject as Google, Inc and indicates this is a wildcard certificate used for all web sites with the *google.com* root domain name.
- **Public key.** RSA asymmetric encryption uses the public key in combination with the matching private key.
- **Usage.** Some certificates are only for encryption or authentication, whereas other certificates support multiple usages.

If you want to view a certificate, check out the View a Certificate Lab in the online labs for this book at <http://gcapremium.com/501labs/>.

Remember this

Certificates are an important part of asymmetric encryption. Certificates include public keys along with details on the owner of the certificate and on the CA that issued the certificate. Certificate owners share their public key by sharing a copy of their certificate.

RSA

Ron Rivest, Adi Shamir, and Leonard Adleman developed RSA in 1977 and the **RSA** acronym uses their last names (Rivest, Shamir, Adleman). It is an asymmetric encryption method using both a public key and a private key in a matched pair, and it is widely used on the Internet and elsewhere due to its strong security.

As an example, email applications often use RSA to privately share a symmetric key between two systems. The application uses the recipient's public key to encrypt a symmetric key, and the recipient's private key decrypts it. The "Protecting Email" section later in this chapter discusses this process in more detail.

The RSA algorithm uses the mathematical properties of prime numbers to generate secure public and private keys. Specifically, RSA relies on the fact that it is difficult to factor the product of two large prime numbers. The math is complex and intriguing to mathematicians, but you don't have to understand the math to understand that RSA is secure if sufficient key sizes are used.

What is a sufficient key size? RSA laboratories recommend a key size of 2,048 bits to protect data through the year 2030. If data needs to be protected beyond 2030, they recommend a key size of 3,072 bits.

Remember this

RSA is widely used to protect data such as email and other data transmitted over the Internet. It uses both a public key and a private key in a matched pair.

Static Versus Ephemeral Keys

The two primary categories of asymmetric keys are static and ephemeral. In general, a static key is semipermanent and stays the same over a long period of time. In contrast, an **ephemeral key** has a very short lifetime

and is re-created for each session.

RSA uses static keys. A certificate includes an embedded public key matched to a private key and this key pair is valid for the lifetime of a certificate, such as a year. Certificates have expiration dates and systems continue to use these keys until the certificate expires. A benefit of static keys is that a CA can validate them as discussed in the “Certificate Issues” section later in this chapter. An ephemeral key pair includes a private ephemeral key and a public ephemeral key.

However, systems use these key pairs for a single session and then discard them. Some versions of Diffie-Hellman (discussed later in this section) use static keys and some versions use ephemeral keys.

Perfect forward secrecy is an important characteristic that ephemeral keys comply with in asymmetric encryption. Perfect forward secrecy indicates that a cryptographic system generates random public keys for each session and it doesn't use a deterministic algorithm to do so. In other words, given the same input, the algorithm will create a different public key. This helps ensure that systems do not reuse keys. The result is that the compromise of a long-term key does not compromise any past keys.

Elliptic Curve Cryptography

Elliptic curve cryptography (ECC) doesn't take as much processing power as other cryptographic methods. Because of this, ECC is often considered with common use cases of low-power devices. For example, ECC is often used with small wireless devices because it doesn't take much processing power to achieve the desired security. It uses mathematical equations to formulate an elliptical curve. It then graphs points on the curve to create keys. This is mathematically easier and requires less processing power, while also being more difficult to crack.

The U.S. NSA previously endorsed the use of ECC for digital signatures and Diffie-Hellman key agreements. However, they announced in late 2015 their intent to move away from its use. Since then, they have deprecated the use of various ECC versions for government agencies.

Diffie-Hellman

Diffie-Hellman (DH) is a key exchange algorithm used to privately

share a symmetric key between two parties. Once the two parties know the symmetric key, they use symmetric encryption to encrypt the data.

Whitfield Diffie and Martin Hellman first published the Diffie-Hellman scheme in 1976. Interestingly, Malcolm J. Williamson secretly created a similar algorithm while working in a British intelligence agency. It is widely believed that the work of these three provided the basis for public-key cryptography.

Diffie-Hellman methods support both static keys and ephemeral keys. RSA is based on the Diffie-Hellman key exchange concepts using static keys. Two Diffie-Hellman methods that use ephemeral keys are:

- **DHE.** Diffie-Hellman Ephemeral (DHE) uses ephemeral keys, generating different keys for each session. Some documents list this as Ephemeral Diffie-Hellman (EDH).
- **ECDHE.** Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) uses ephemeral keys generated using ECC. Another version, Elliptic Curve Diffie-Hellman (ECDH), uses static keys.

When Diffie-Hellman is used, the two parties negotiate the strongest group that both parties support. There are currently more than 25 DH (Diffie Hellman) groups in use and they are defined as DH Group 1, DH Group 2, and so on. Higher group numbers indicate the group is more secure. For example, DH Group 1 uses 768 bits in the key exchange process and DH Group 15 uses 3,072 bits.

Remember this

Diffie-Hellman is a secure method of sharing symmetric encryption keys over a public network. Elliptic curve cryptography is commonly used with small wireless devices. ECDHE is a version of Diffie-Hellman that uses elliptic curve cryptography to generate encryption keys.

Steganography

Steganography hides data inside other data, or, as some people have said, it hides data in plain sight. The goal is to hide the data in such a way that no one suspects there is a hidden message. It doesn't actually encrypt the data, so it can't be classified as either symmetric or asymmetric. However, it can effectively hide information using obfuscation, so it is

included with encryption topics. As mentioned in Chapter 1, obfuscation methods attempt to make something unclear or difficult to understand.

There are a variety of steganography tools available that make the process easier. Additionally, these tools attempt to resist detection by forensic methods. For example, Kali Linux includes Steghide and StegoSuite, two tools you can use to embed data into graphic files.

Some common examples of steganography are:

- **Hide data by manipulating bits.** It's possible to manipulate some bits within an image or sound file to embed a message. One method of embedding data in large files is modifying the least significant bit in some bytes. By modifying the least significant bit in some of the individual bytes of a JPEG file, it embeds a message, but the changes are so small that they are difficult to detect. However, if people know the file includes a message, they can easily retrieve it.
- **Hide data in the white space of a file.** Many files have unused space (called white space) at the end of file clusters. Imagine a small 6 KB file stored in two 4 KB clusters. It has an extra 2 KB of unused space and it's possible to fill this white space with a message. For example, you can embed a message into the white space of a GIF or JPEG file without altering the file size.

Security professionals use steganalysis techniques to detect steganography, and the most common method is with hashing. If a single bit of a file is modified, the hashing algorithm creates a different hash. By regularly taking the hashes of different files and comparing them with previous hashes, it's easy to detect when a file has been modified.

If you want to see how to embed a text file in an image file, check out the Steganography Lab mentioned in Chapter 1 at

<http://gcgapremium.com/501labs>.

Remember this

Steganography hides messages or other data within a file. For example, you can hide messages within the white space of a JPEG or GIF file. Security professionals use hashing to detect changes in files that may indicate the use of steganography.

Using Cryptographic Protocols

With a basic understanding of hashing, symmetric encryption, and asymmetric encryption, it's easier to grasp how cryptography is used. Many applications use a combination of these methods, and it's important to understand how they're intertwined.

When describing public and private keys earlier, it was stressed that one key encrypts and the other key decrypts. A common question is “which one encrypts and which one decrypts?” The answer depends on what you're trying to accomplish. The following sections describe the details, but as an overview, these are the important points related to these keys:

- Email digital signatures
 - The *sender's private key* encrypts (or signs).
 - The *sender's public key* decrypts.
- Email encryption
 - The *recipient's public key* encrypts.
 - The *recipient's private key* decrypts.
- Web site encryption
 - The *web site's public key* encrypts.
 - The *web site's private key* decrypts.
 - The *symmetric key* encrypts data in the web site session.

Email and web site encryption commonly use a combination of both asymmetric and symmetric encryption. They use asymmetric encryption for key exchange, privately sharing a symmetric key. Symmetric encryption encrypts the data.

Remember this

Knowing which key encrypts and which key decrypts will help you answer many questions on the exam. For example, just by knowing that a private key is encrypting, you know that it is being used for a digital signature.

Protecting Email

Cryptography provides two primary security methods you can use with email: digital signatures and encryption. These are separate processes, but you can digitally sign and encrypt the same email.

Signing Email with Digital Signatures

Digital signatures are similar in concept to handwritten signatures on printed documents that identify individuals, but they provide more security benefits. The digital signature algorithm (**DSA**) uses an encrypted hash of a message. The hash is encrypted with the sender's private key. If the recipient of a digitally signed email can decrypt the hash, it provides the following three security benefits:

- **Authentication.** This identifies the sender of the email. Email recipients have assurances the email actually came from who it appears to be coming from. For example, if an executive digitally signs an email, recipients know it came from the executive and not from an attacker impersonating the executive.
- **Non-repudiation.** The sender cannot later deny sending the message. This is sometimes required with online transactions. For example, imagine Homer sends an order to sell stocks using a digitally signed email. If the stocks increase after his sale completes, he can't deny the transaction.
- **Integrity.** This provides assurances that the message has not been modified or corrupted. Recipients know that the message they received is the same as the sent message.

Digital signatures are much easier to grasp if you understand some other cryptography concepts discussed in this chapter. As a short review, these concepts are:

- **Hashing.** Digital signatures start by creating a hash of the message. A hash is simply a number created by executing a hashing algorithm on the message.
- **Certificates.** Digital signatures need certificates, and certificates include the sender's public key.
- **Public/private keys.** In a digital signature, the sender uses the sender's private key to encrypt the hash of the message. The recipient uses the sender's public key to decrypt the hash of the message. The public key is often distributed in an *S/MIME.p7s* formatted file.

Figure 10.5 shows an overview of this process. In the figure, Lisa is sending a message to Bart with a digital signature. Note that the message "I passed" is not secret. If it was, Lisa would encrypt it, which is a separate

process. The focus in this explanation is only the digital signature.

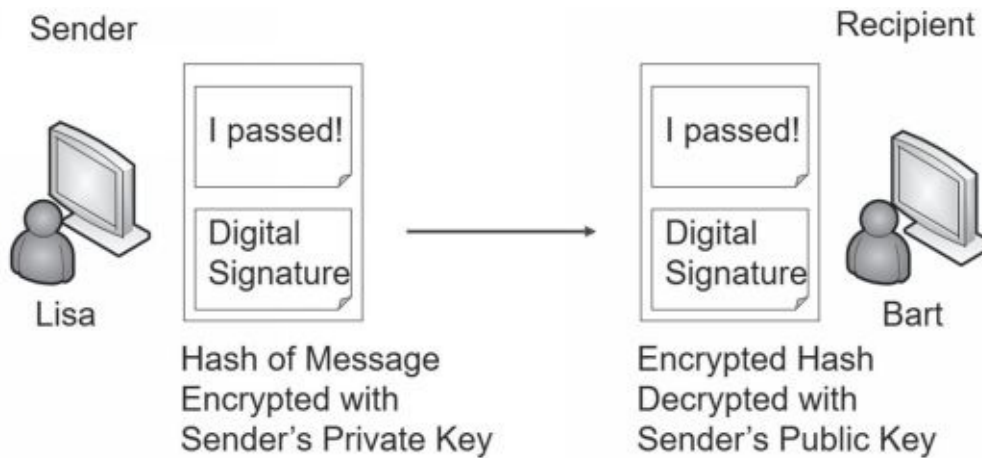


Figure 10.5: Digital signature process

Lisa creates her message in an email program, such as Microsoft Outlook. Once Microsoft Outlook is configured, all she does is click a button to digitally sign the message. Here is what happens when she clicks the button:

1. The application hashes the message.
2. The application retrieves Lisa's private key and encrypts the hash using this private key.
3. The application sends both the encrypted hash digital signature (the digital signature) and the unencrypted message to Bart.

When Bart's system receives the message, it verifies the digital signature using the following steps:

1. Bart's system retrieves Lisa's public key, which is in Lisa's public certificate. In some situations, Lisa may have sent Bart a copy of her certificate with her public key. In domain environments, Bart's system can automatically retrieve Lisa's certificate from a network location.
2. The email application on Bart's system decrypts the encrypted hash with Lisa's public key.
3. The application calculates the hash on the received message.
4. The application compares the decrypted hash with the calculated hash.

If the calculated hash of the received message is the same as the encrypted hash of the digital signature, it validates several important checks:

- **Authentication.** Lisa sent the message. The public key can only decrypt something encrypted with the private key, and only Lisa has the private key. If the decryption succeeded, Lisa's private key must have encrypted the hash. On the other hand, if another key was used to encrypt the hash, Lisa's public key could not decrypt it. In this case, Bart will see an error indicating a problem with the digital signature.
- **Non-repudiation.** Lisa cannot later deny sending the message. Only Lisa has her private key and if the public key decrypted the hash, the hash must have been encrypted with her private key. Non-repudiation is valuable in online transactions.
- **Integrity.** Because the hash of the sent message matches the hash of the received message, the message has maintained integrity. It hasn't been modified.

At this point, you might be thinking, if we do all of this, why not just encrypt the message, too? The answer is resources. It doesn't take much processing power to encrypt 256 bits in a SHA- 256 hash. In contrast, it would take quite a bit of processing power to encrypt a lengthy email and its attachments. However, if you need to ensure confidentiality of the email, you can encrypt it.

Remember this

A digital signature is an encrypted hash of a message. The sender's private key encrypts the hash of the message to create the digital signature. The recipient decrypts the hash with the sender's public key. If successful, it provides authentication, non-repudiation, and integrity. Authentication identifies the sender. Integrity verifies the message has not been modified. Non-repudiation prevents senders from later denying they sent an email.

Encrypting Email

There are times when you want to ensure that email messages are only readable by authorized users. You can encrypt email and just as any other time encryption is used, encryption provides confidentiality.

Encrypting Email with Only Asymmetric Encryption

Imagine that Lisa wants to send an encrypted message to Bart. The following steps provide a simplified explanation of the process if only asymmetric encryption is used:

1. Lisa retrieves a copy of Bart's certificate that contains his public key.
2. Lisa encrypts the email with Bart's public key.
3. Lisa sends the encrypted email to Bart.
4. Bart decrypts the email with his private key.

This works because Bart is the only person who has access to his private key. If attackers intercepted the email, they couldn't decrypt it without Bart's private key. It's important to remember that when you're encrypting email contents, the recipient's public key encrypts and the recipient's private key decrypts. The sender's keys are not involved in this process. In contrast, a digital signature only uses the sender's keys but not the recipient's keys.

In most cases, the public key doesn't actually encrypt the message, but instead encrypts a symmetric key used to encrypt the email. The recipient then uses the private key to decrypt the symmetric key, and then uses the symmetric key to decrypt the email.

Remember this

The recipient's public key encrypts when encrypting an email message and the recipient uses the recipient's private key to decrypt an encrypted email message.

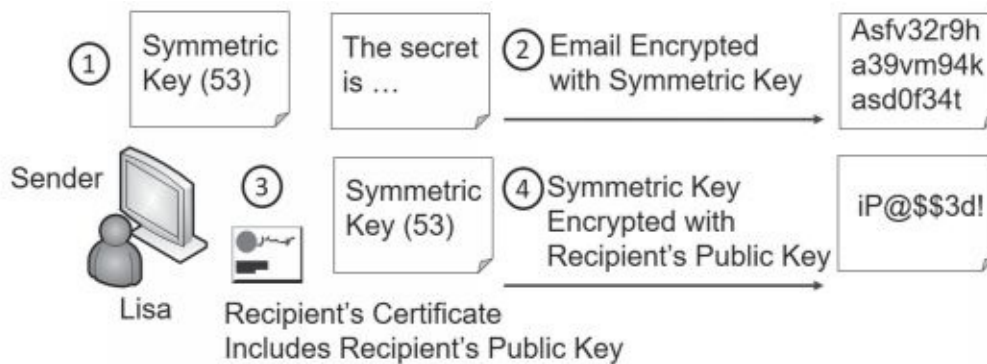
Encrypting Email with Asymmetric and Symmetric Encryption

The previous description provides a simplistic explanation of email encryption used by some email applications. However, most email applications combine both asymmetric and symmetric encryption. You may remember from earlier in this chapter that asymmetric encryption is slow and inefficient, but symmetric encryption is very quick.

Instead of using only symmetric encryption, most email applications use asymmetric encryption to privately share a session key. They then use symmetric encryption to encrypt the data. For example, imagine that Lisa is sending Bart an encrypted message. Figure 10.6 shows the process of

encrypting the message and the symmetric key. Figure 10.7 shows the process of sending the encrypted message and encrypted session key, and identifies how the recipient can decrypt the data:

1. Lisa identifies a symmetric key to encrypt her email. For this example, assume it's a simplistic symmetric key of 53, though a symmetric algorithm like AES would use 128-bit or larger keys.
2. Lisa encrypts the email contents with the symmetric key of 53.
3. Lisa retrieves a copy of Bart's certificate that contains his public key.
4. She uses Bart's public key to encrypt the symmetric key of 53.
5. Lisa sends the encrypted email and the encrypted symmetric key to Bart.
6. Bart decrypts the symmetric key with his private key.
7. He then decrypts the email with the decrypted symmetric key.



Figure

10.6: Encrypting email

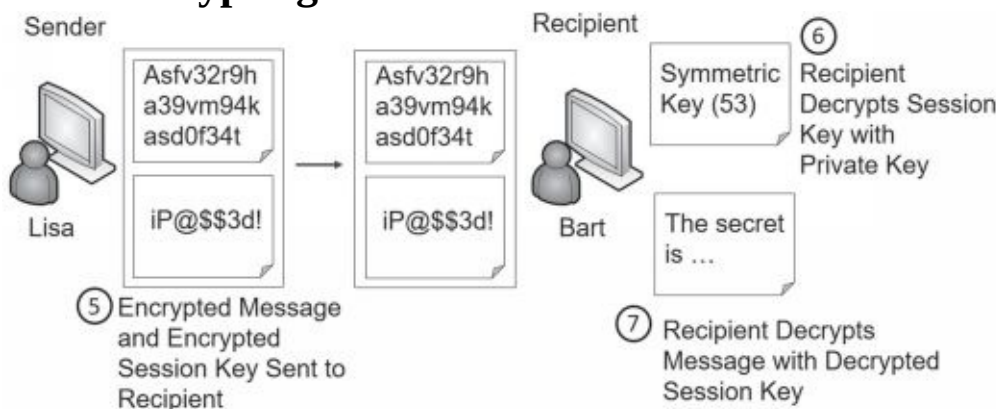


Figure 10.7: Decrypting email

Unauthorized users who intercept the email sent by Lisa won't be able to read it because it's encrypted with the symmetric key. Additionally, they

can't read the symmetric key because it's encrypted with Bart's public key, and only Bart's private key can decrypt it.

S/MIME

Secure/Multipurpose Internet Mail Extensions(**S/MIME**) is one of the most popular standards used to digitally sign and encrypt email. Most email applications that support encryption and digital signatures use S/MIME standards.

S/MIME uses RSA for asymmetric encryption and AES for symmetric encryption. It can encrypt email at rest (stored on a drive) and in transit (data sent over the network). Because S/MIME uses RSA for asymmetric encryption, it requires a PKI to distribute and manage certificates.

PGP/GPG

Pretty Good Privacy (PGP) is a method used to secure email communication. It can encrypt, decrypt, and digitally sign email. Phillip Zimmerman designed PGP in 1991, and it has gone through many changes and improvements over the years. Symantec Corporation purchased it in June 2010.

OpenPGP is a PGP-based standard created to avoid any conflict with existing licensing. In other words, users have no obligation to pay licensing fees to use it. Some versions of PGP follow S/MIME standards. Other versions follow OpenPGP standards. GNU Privacy Guard (GPG) is free software that is based on the OpenPGP standard.

Each of the PGP versions uses the RSA algorithm and public and private keys for encryption and decryption. Just like S/MIME, PGP uses both asymmetric and symmetric encryption.

HTTPS Transport Encryption

Transport encryption methods encrypt data-in-transit to ensure transmitted data remains confidential. This includes data transmitted over the Internet and on internal networks. As an example, Chapter 3, "Exploring Network Technologies and Tools," discusses the use of Secure Shell (SSH) to encrypt traffic, such as Secure File Transfer Protocol (SFTP). This section focuses on transport encryption methods used with HTTPS.

SSL Versus TLS

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are encryption protocols that have been commonly used to encrypt data-in-transit. For example, it is common to encrypt HTTPS with either SSL or TLS to ensure confidentiality of data transmitted over the Internet. They can also be used to encrypt other transmissions such as File Transfer Protocol Secure (FTPS).

Netscape created SSL for its web browser and updated it to version SSL 3.0. This was before organizations such as the Internet Engineering Task Force (IETF) created and maintained standards. Netscape's success waned and there wasn't a standardization process to update SSL, even though all web browsers were using it. The IETF created TLS to standardize improvements with SSL.

Both SSL and TLS provide certificate-based authentication and they encrypt data with a combination of both symmetric and asymmetric encryption during a session. They use asymmetric encryption for the key exchange (to privately share a session key) and symmetric encryption to encrypt data displayed on the web page and transmitted during the session. The next section shows this process.

Transport Layer Security (TLS) is a replacement for SSL and is widely used in many different applications. The IETF has updated and published several TLS documents specifying the standard. TLS 1.0 is based on SSL 3.0 and is referred to as SSL 3.1. Similarly, each update to TLS indicates it is an update to SSL. For example, TLS 1.1 is called SSL 3.2 and TLS 1.2 is called SSL 3.3.

As mentioned in Chapter 3, SSL has been deprecated by most organizations in favor of TLS. However, many people commonly refer to TLS as SSL/TLS as if they are the same. Even the CompTIA Security+ objectives list many topics as SSL or SSL/TLS. In this context, you can consider any reference to SSL on the exam as a reference to TLS.

It's important to remember that TLS and SSL require certificates. Certificate Authorities (CAs) issue and manage certificates, so a CA is required to support TLS and SSL. These CAs can be internal or external third-party CAs.

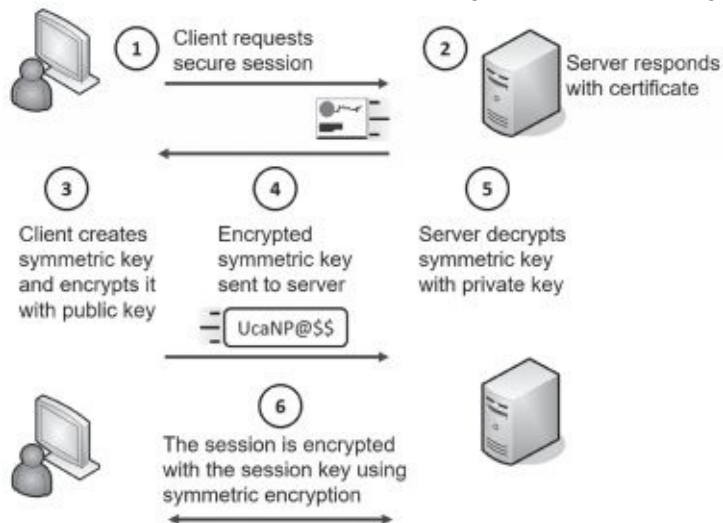
Remember this

TLS is the replacement for SSL. Both TLS and SSL require

certificates issued by Certificate Authorities (CAs). TLS encrypts HTTPS traffic, but it can also encrypt other traffic

Encrypting HTTPS Traffic with TLS

HTTP Secure (HTTPS) is commonly used on the Internet to secure web traffic. HTTPS commonly uses TLS to encrypt the traffic, with both asymmetric and symmetric encryption. If you're able to grasp the basics of how HTTPS combines both asymmetric and symmetric



encryption, you'll have what you need to know for most protocols that use both encryptions.

Because asymmetric encryption isn't efficient to encrypt large amounts of data, symmetric encryption is used to encrypt the session data. However, both the client and the server must know what this symmetric key is before they can use it. They can't whisper it to each other over the Internet. That's like an actor on TV using a loud whisper, or stage whisper, to share a secret. Millions of TV viewers can also hear the secret.

Instead, HTTPS uses asymmetric encryption to transmit a symmetric key using a secure key exchange method. It then uses the symmetric key with symmetric encryption to encrypt all the data in the HTTPS session.

Figure 10.8 and the following steps show the overall process of establishing and using an HTTPS session. As you read these steps, try to keep these two important concepts in mind:

- TLS uses *asymmetric* encryption to securely share the symmetric key.
- TLS uses *symmetric* encryption to encrypt the session data.

Figure 10.8: Simplified TLS handshake process used with HTTPS

1. The client begins the process by requesting an HTTPS session. This could be by entering an HTTPS address in the URL or by clicking on an HTTPS link.
2. The server responds by sending the server's certificate. The certificate includes the server's public key. The matching private key is on the server and only accessible by the server.
3. The client creates a symmetric key and encrypts it with the server's public key. As an example, imagine that the symmetric key is 53 (though it would be much more complex in a live session). The client encrypts the symmetric key of 53 using the web server's public key creating ciphertext of UcaNP@\$\$\$. This symmetric key will be used to encrypt data in the HTTPS session, so it is sometimes called a session key.
4. The client sends the encrypted session key (UcaNP@\$\$\$) to the web server. Only the server's private key can decrypt this. If attackers intercept the encrypted key, they won't be able to decrypt it because they don't have access to the server's private key.
5. The server receives the encrypted session key and decrypts it with the server's private key. At this point, both the client and the server know the session key.
6. All the session data is encrypted with this symmetric key using symmetric encryption.

The amazing thing to me is that this happens so quickly. If a web server takes as long as five seconds, many of us wonder why it's taking so long. However, a lot is happening to establish this session.

Cipher Suites

Cipher suites are a combination of cryptographic algorithms that provide several layers of security for TLS and SSL, though most organizations have deprecated the use of SSL. When two systems connect, they identify a cipher suite that is acceptable to both systems and then use the protocols within that suite. The protocols within the suite provide three primary cryptographic solutions. They are:

- **Encryption.** Encryption provides confidentiality of data. TLS uses

asymmetric cryptography to privately exchange a symmetric key and then encrypts the data with a symmetric algorithm. TLS supports several types of symmetric encryption, including 3DES and AES.

- **Authentication.** TLS uses certificates for authentication. Clients can verify the authenticity of the certificate by querying the CA that issued the certificate.
- **Integrity.** TLS uses a message authentication code (MAC) for integrity. For example, it can use HMAC-MD5 or HMAC-SHA256.

There are over 200 named cipher suites, and systems identify them with a cipher identifier as a string of hexadecimal characters and a coded name.

Here are two examples:

- **0x00C031.** TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
- **0x00003C.** TLS_RSA_WITH_AES_128_CBC_SHA256

If you're familiar with the acronyms, you can get a good idea of what each cipher suite is using. Here are some notes for clarification:

- **Protocol.** Both are using TLS.
- **Key exchange method.** The first one is using ECDH and the second one is using RSA.
- **Authentication.** Both are using RSA, though, they shortened the code in the second one. Instead of listing RSA twice (for both the key exchange method and authentication), it is only listed once.
- **Encryption.** Both are using 128-bit AES, though in different modes of operation. Galois/ Counter Mode (GCM) and Cipher Block Chaining (CBC) are the two modes identified here. The next section discusses cipher modes in more depth.
- **Integrity.** Both are using the SHA-256 hashing algorithm.

Some cipher suites are very old and include encryption algorithms such as DES. Clearly, they shouldn't be used and are disabled by default in most systems today. When necessary, administrators configure systems and applications to disable older specific cipher suites.

When two systems connect, they negotiate to identify which cipher suite they use. Each system passes a prioritized list of cipher suites it is willing to use. They then choose the cipher suite that is highest on their lists and common to both lists.

Implementation Versus Algorithm Selection

As you read about different algorithms and different modes of operation, you might have been wondering how this is relevant. It really depends on your job. Two terms are relevant here:

- **Crypto module.** A *crypto module* is a set of hardware, software, and/or firmware that implements cryptographic functions. This includes algorithms for encryption and hashing, key generation, and authentication techniques such as a digital signature.
- **Crypto service providers.** A *crypto service provider* is a software library of cryptographic standards and algorithms. These libraries are typically distributed within crypto modules.

As an example, software developers access software libraries when implementing cryptographic functions in their programs. If they need to encrypt data, they don't start from nothing. Instead, they access a crypto service provider, or cryptographic service provider, which is a library of different implementations of cryptographic standards and algorithms.

Crypto service providers are typically distributed within crypto modules. As an example, Python is a popular application used by web developers to create dynamic web sites. Python includes a rich assortment of crypto modules developers can use. The Python Cryptography Toolkit includes a library of both hashing functions and encryption algorithms. Developers simply follow the syntax defined within the library to implement these hashing functions and encryption algorithms.

Developers should know about the different cryptographic algorithms, along with the different modes of operation. As an example, although it's possible to select DES for encryption, this would be a mistake because DES has been deprecated. Similarly, it's possible for a developer to implement an algorithm using the ECB mode of operation. However, ECB has known weaknesses, so this would also be a mistake.

Administrators implement algorithms via cipher suites on servers. Their responsibility is to ensure that deprecated and weak cipher suites are disabled on servers. If administrators leave weak or deprecated algorithms functioning on servers, it makes the servers susceptible to attacks such as downgrade attacks.

Downgrade Attacks on Weak Implementations

A *downgrade attack* is a type of attack that forces a system to downgrade its security. The attacker then exploits the lesser security control.

It is most often associated with cryptographic attacks due to weak implementations of cipher suites.

The common example is with Transport Layer Security (TLS) and Secure Sockets Layer (SSL). SSL has known vulnerabilities and has been replaced with TLS in most implementations. However, many servers have both SSL and TLS installed. If a client is unable to use TLS, the server will downgrade its security and use SSL.

Attackers exploit this vulnerability by configuring their systems so that they cannot use TLS. When they communicate with the server, the server downgrades security to use SSL instead of TLS. This allows attackers to launch SSL-based attacks such as the well-known Padding Oracle On Downgraded Legacy Encryption (POODLE) attack.

One way to ensure that SSL isn't used on a site is to modify the server's protocol list and ensure that SSL is disabled. Typically, a web site server will have the following five options: SSL 2, SSL 3, TLS 1.0, TLS 1.1, and TLS 1.2. You can prevent SSL-based downgrade attacks by disabling SSL 2 and SSL 3 on the web site server.

Similarly, cipher suites with known vulnerabilities should be disabled. If weak cipher suites are enabled on a server, it increases the vulnerabilities.

Remember this

Administrators should disable weak cipher suites and weak protocols on servers. When a server has both strong and weak cipher suites, attackers can launch downgrade attacks bypassing the strong cipher suite and exploiting the weak cipher suite.

Exploring PKI Components

A **Public Key Infrastructure (PKI)** is a group of technologies used to request, create, manage, store, distribute, and revoke digital certificates. Asymmetric encryption depends on the use of certificates for a variety of purposes, such as protecting email and protecting Internet traffic with SSL and TLS. For example, HTTPS sessions protect Internet credit card transactions, and these transactions depend on a PKI.

A primary benefit of a PKI is that it allows two people or entities to communicate securely without knowing each other previously. In other

words, it allows them to communicate securely through an insecure public medium such as the Internet.

For example, you can establish a secure session with Amazon.com even if you've never done so before. Amazon purchased a certificate from Symantec. As shown in the “Encrypting HTTPS Traffic with TLS” section previously, the certificate provides the ability to establish a secure session.

A key element in a PKI is a Certificate Authority.

Certificate Authority

A Certificate Authority (CA) issues, manages, validates, and revokes certificates. In some contexts, you might see a CA referred to as a Certification Authority, but they are the same thing. CAs can be very large, such as Comodo or Symantec, which are public CAs. A CA can also be very small, such as a single service running on a server within a private network.

Public CAs make money by selling certificates. For this to work, the public CA must be trusted. If the CA is trusted, all certificates issued by the CA are trusted.

This is similar to how a driver's license is trusted. The Department of Motor Vehicles (DMV) issues driver's licenses after validating a person's identity. If you want to cash a check, you might present your driver's license to prove your identity. Businesses trust the DMV, so they trust the driver's license. On the other hand, if you purchased an ID from Gibson's Instant IDs, businesses might not trust it.

Although we might trust the DMV, why would a computer trust a CA? The answer is based on the certificate trust path.

Certificate Chaining and Trust Models

CAs are trusted by placing a copy of their root certificate into a trusted root CA store. The **root certificate** is the first certificate created by the CA that identifies it, and the store is just a collection of these root certificates. If the CA's root certificate is placed in this store, all certificates issued by this CA are trusted.

Figure 10.9 shows the Trusted Root Certification Authority store on a

Windows computer. You can see that there are many certificates from many different CAs. In the figure, I've selected one of the certificates from COMODO Certification Authority. One of the labs for this chapter (available at <http://gcgapremium.com/501labs>) shows how to access this on a Windows computer.

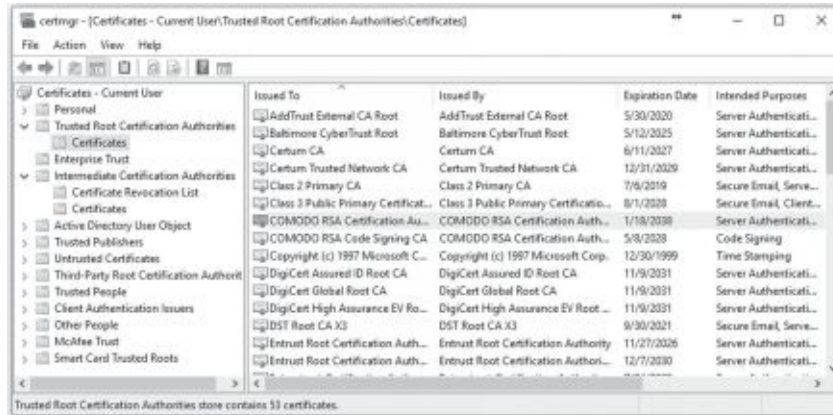


Figure 10.9:

Trusted Root Certification Authorities

Public CAs such as Symantec and Comodo negotiate with web browser developers to have their certificates included with the web browser. This way, any certificates that they sell to businesses are automatically trusted.

The most common trust model is the hierarchical trust model, also known as a centralized trust model. In this model, the public CA creates the first CA, known as the root CA. If the organization is large, it can create intermediate and child CAs. If you look back at Figure 10.9, you can see that it includes a section used to store intermediate CA certificates. A large trust chain works like this:

- The root CA issues certificates to intermediate CAs.



- Intermediate CAs issue certificates to child

CAs.

- Child CAs issue certificates to devices or end users.

Figure 10.10: Certificate chain

Certificate chaining combines all the certificates from the root CA down to the certificate issued to the end user. For example, Figure 10.10 shows the certificate chain for a wildcard certificate issued to google.com. A certificate chain would include all three certificates. In a small organization, the root CA can simply issue certificates to the devices and end users. It's not necessary to have intermediate and child CAs.

Another type of trust model is a web of trust or decentralized trust model, sometimes used with PGP and GPG. A web of trust uses self-signed certificates, and a third party vouches for these certificates. For example, if five of your friends trust a certificate, you can trust the certificate. If the third party is a reliable source, the web of trust provides a secure alternative. However, if the third party does not adequately verify certificates, it can result in the use of certificates that shouldn't be trusted.

Registration and CSRs

Users and systems request certificates from a CA using a registration process. In some cases, a user enters information manually into a web site form. In other cases, a user sends a specifically formatted file to the CA. Within a domain, the system handles much of the process automatically.

As an example, imagine I wanted to purchase a certificate for *GetCertifiedGetAhead.com* for secure HTTPS sessions. I would first create a public and private key pair. Many programs are available to automate this process. For example, OpenSSL is a command-line program included in many Linux distributions. It creates key pairs in one command and allows you to export the public key to a file in a second command. Technically, OpenSSL and similar applications create the private key first. However, these applications appear to create both keys at the same time.

I would then put together a certificate signing request (**CSR**) for the certificate, including the purpose of the certificate and information about the web site, the public key, and me. Most CAs require CSRs to be formatted using the Public-Key Cryptography Standards (PKCS) #10 specification. The CSR includes the public key, but not the private key. A CA typically

publishes a certificate template showing exactly how to format the CSR.

After receiving the CSR, the CA validates my identity and creates a certificate with the public key. The validation process is different based on the usage of the certificate. In some cases, it includes extensive checking, and in other cases, verification comes from the credit card I use to purchase it.

I can then register this certificate with my web site along with the private key. Any time someone initiates a secure HTTPS connection, the web site sends the certificate with the public key and the TLS/SSL session creates the session.

Certificates use object identifiers (OIDs) to identify specific objects within the certificates and some CAs require OIDs within the CSR for certain items. The OID is a string of numbers separated by dots. OIDs can be used to name almost every object type in certificates.

As an example, while looking at a Google certificate, I noticed a certificate with this OID on the General tab, 1.3.6.1.4.1.11129.2.5.1:

- The first 1 indicates that it is an International Organization for Standardization (ISO) OID.
- 1.3 indicates it is an identified organization.
- 1.3.6.1.4.1 indicates it is using an IANA enterprise number.
- 1.3.6.1.4.1.11129 indicates the organization is Google.
- The additional numbers (2.5.1) are specifically defined within Google's enterprise.

In large organizations, a registration authority(RA) can assist the CA by collecting registration information. The RA never issues certificates. Instead, it only assists in the registration process.

If the CA is online, meaning it is accessible over a network, it's possible to submit the CSR using an automated process. However, an organization may choose to keep some CAs offline to protect them from attacks. Offline CAs can only accept CSRs manually.

Remember this

You typically request certificates using a certificate signing request (CSR). The first step is to create the RSA-based private key, which is used to create the public key. You then include the public key in the CSR and the CA will embed the public key in the certificate. The private key is not sent to the CA.

Revoking Certificates

Normally, certificates expire based on the Valid From and Valid To dates. However, there are some instances when a CA will revoke a certificate before it expires.

For example, if a private key is publicly available, the key pair is compromised. It no longer provides adequate security because the private key is no longer private. Similarly, if the CA itself is compromised through a security breach, certificates issued by the CA may be compromised, so the CA can revoke certificates.

In general, any time a CA does not want anyone to use a certificate, the CA revokes it. Although the most common reasons are due to compromise of a key or the CA, there are others. A CA can use any of the following reasons when revoking a certificate:

- Key compromise
- CA compromise
- Change of affiliation
- Superseded
- Cease of operation
- Certificate hold



CAs use certificate revocation lists (CRLs, pronounced “crill”) to revoke a certificate. The **CRL** is a version 2 certificate that includes a list of revoked certificates by serial number. For example, Figure 10.11 shows a copy of a CRL. One of the labs for this chapter shows you how to download and view a CRL.

Figure 10.11: Certificate revocation list

Certificate Issues

Before clients use a certificate, they first verify it is valid with some checks. There are many different certificate issues that can result in an invalid certificate. Browsers typically display an error describing the issue and encouraging users not to use the certificate. Applications that detect a certificate issue might display an error using a certificate, but they are typically coded to not use it. Some of the common issues are:

- **Expired.** The first check is to ensure that it isn't expired. If the certificate is expired, the computer system typically gives the user an error indicating the certificate is not valid.
- **Certificate not trusted.** The next check is to see if the certificate was issued by a trusted CA. For example, a Windows system will look in the Trusted Root Certification Authority store and the Intermediate Certification Authorities store shown previously in Figure 10.9. If the system doesn't have a copy of the CA's certificate, it will indicate the certificate is not trusted. Users can override this warning, though there are often warnings encouraging users not to continue.
- **Improper certificate and key management.** Private keys should remain private. They are typically stored in an encrypted format and never shared publicly. If the certificates holding the private keys aren't managed properly, it can compromise the certificate.

Clients also validate certificates through the CA to ensure they haven't been revoked. First, they verify that the certificate was issued by a trusted CA. Next, they query the CA to verify the CA hasn't revoked the certificate. A common method of validating a certificate is by requesting a copy of the CRL, as shown in Figure 10.12. The following steps outline the process:

1. The client initiates a session requiring a certificate, such as an HTTPS session.
2. The server responds with a copy of the certificate that includes the public key.
3. The client queries the CA for a copy of the CRL.
4. The CA responds with a copy of the CRL.

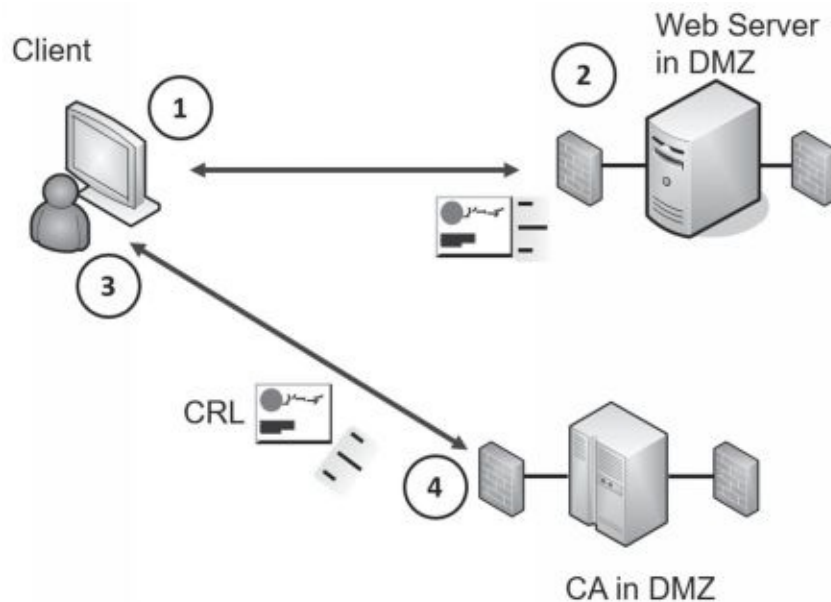


Figure 10.12: Validating a certificate

The client then checks the serial number of the certificate against the list of serial numbers in the CRL. If the certificate is revoked for any reason, the application gives an error message to the user.

CRLs are typically cached after being downloaded the first time. Instead of requesting another copy of the CRL, clients use the cached copy. This generates a lot of traffic between clients and the CA.

Another method of validating a certificate is with the Online Certificate Status Protocol (**OCSP**). OCSP allows the client to query the CA with the serial number of the certificate. The CA then responds with an answer of “good,” “revoked,” or “unknown.” A response of “unknown” could indicate the certificate is a forgery.

Because OCSP provides a real-time response, it is an excellent example of supporting a common use case of low latency. If a CA revokes a certificate, clients using OCSP will know immediately. In contrast, if clients are using a cached CRL, they will be unaware of the revoked certificate until another copy of the CRL is downloaded.

Over time, authorities realized that OCSP was generating a lot of real-time traffic to the CA because it requires a CA to respond to every request. OCSP **stapling** solves this problem. The certificate presenter (such as the web server in Figure 10.12) obtains a timestamped OCSP response from the CA. Before sending it, the CA signs it with a digital signature. The certificate presenter then appends (or metaphorically staples) a timestamped OCSP

response to the certificate during the TLS handshake process. This eliminates the need for clients to query the CA.

Remember this

CAs revoke certificates for several reasons such as when the private key is compromised or the CA is compromised. The certificate revocation list (CRL) includes a list of revoked certificates and is publicly available. An alternative to using a CRL is the Online Certificate Status Protocol (OCSP), which returns answers such as good, revoked, or unknown. OCSP stapling appends a digitally signed OCSP response to a certificate.

Public Key Pinning

Public key ***pinning*** is a security mechanism designed to prevent attackers from impersonating a web site using fraudulent certificates. When configured on a web site server, the server responds to client HTTPS requests with an extra header. This extra header includes a list of hashes derived from valid public keys used by the web site. It also includes a max-age field specifying how long the client should store and use the data.

When clients connect to the same web site again, they recalculate the hashes and then compare the recalculated hashes with the stored hashes. If the hashes match, it verifies that the client is connected to the same web site.

Web site administrators create hashes of one or more certificates used by the web site. This can be the public key used by the web site's certificate. It can also include any public keys from certificates in the certificate chain such as the public key from the root CA certificate, and/or the public key from intermediate CA certificates. Last, it must include a backup key that can be used if the current key becomes invalid.

Remember this

Certificate stapling is an alternative to OCSP. The certificate presenter (such as a web server) appends the certificate with a timestamped digitally signed OCSP response from the CA. This reduces OCSP traffic to and from the CA. Public key pinning helps prevent attackers from impersonating a web site with a fraudulent

certificate. The web server sends a list of public key hashes that clients can use to validate certificates sent to clients in subsequent sessions.

Key Escrow

Key escrow is the process of placing a copy of a private key in a safe environment. This is useful for recovery. If the original is lost, the organization retrieves the copy of the key to access the data. Key escrow isn't required, but if an organization determines that data loss is unacceptable, it will implement a key escrow process.

In some cases, an organization provides a copy of the key to a third party. Another method is to designate employees within the organization who will be responsible for key escrow. These employees maintain and protect copies of the key, and if the original key is lost, they check out a copy of the key to an administrator or user.

Recovery Agent

A key recovery agent is a designated individual who can recover or restore cryptographic keys. In the context of a PKI, a recovery agent can recover private keys to access encrypted data. The recovery agent may be a security professional, administrator, or anyone designated by the company.

In some cases, the recovery agent can recover encrypted data using a different key. For example, Microsoft BitLocker supports encryption of entire drives. It's possible to add a data recovery agent field when creating a BitLocker encrypted drive. In this case, BitLocker uses two keys. The user has one key and uses it to unlock the drive during day-to-day use. The second key is only accessible by the recovery agent and is used for recovery purposes if the original key is lost or becomes inaccessible.

Comparing Certificate Types

Certificates are sometimes identified based on their usage. The following bullets describe some common certificate types:

- **Machine/Computer.** Certificates issued to a device or a computer

are commonly called machine certificates or computer certificates. The certificate is typically used to identify the computer within a domain.

- **User.** Certificates can also be issued to users. They can be used for encryption, authentication, smart cards, and more. For example, Microsoft systems can create user certificates allowing the user to encrypt data using Encrypting File System (EFS).
- **Email.** The two uses of email certificates are for encryption of emails and digital signatures.
- **Code signing.** Developers often use code signing certificates to validate the authentication of executable applications or scripts. The code signing certificate verifies the code has not been modified.
- **Self-signed.** A self-signed certificate is not issued by a trusted CA. Private CAs within an enterprise often create self-signed certificates. They aren't trusted by default. However, administrators can use automated means to place copies of the self-signed certificate into the trusted root CA store for enterprise computers. Self-signed certificates from private CAs eliminate the cost of purchasing certificates from public CAs.
- **Wildcard.** A *wildcard certificate* starts with an asterisk (*) and can be used for multiple domains, but each domain name must have the same root domain. For example, Google uses a wildcard certificate issued to *.google.com. This same certificate can be used for other Google domains, such as accounts.google.com and support.google.com. Wildcard certificates can reduce the administrative burden associated with managing multiple certificates.
- **SAN.** A Subject Alternative Name (SAN) is used for multiple domains that have different names, but are owned by the same organization. For example, Google uses SANs of *.google.com, *.android.com, *.cloud.google.com, and more. It is most commonly used for systems with the same base domain names, but different top-level domains. For example, if Google used names such as google.com and google.net, it could use a single SAN certificate for both domain names. Similarly, a SAN certificate can be used for google.com and www.google.com.
- **Domain validation.** A domain-validated certificate indicates that

the certificate requestor has some control over a DNS domain. The CA takes extra steps to contact the requestor such as by email or telephone. The intent is to provide additional evidence to clients that the certificate and the organization are trustworthy.

- **Extended validation.** Extended validation certificates use additional steps beyond domain validation. If you visit a domain with an extended validation certificate, the address bar includes the name of the company before the actual URL. This helps prevent impersonation from phishing attacks. For example, PayPal uses an extended validation certificate. If you visit, you'll see that it shows PayPal, Inc [US] | before the URL. Imagine an attacker sends a phishing email with a link to paypa1.com (with 1 in paypa1 instead of the letter). If a user clicks the link, she will be able to see that the site isn't truly a PayPal site, assuming she understands extended validation certificates.

Certificate Formats

Most certificates use one of the X.509 v3 formats. The primary exception is certificates used to distribute certificate revocation lists (described later in this chapter), which use the X.509 v2 format.

Certificates are typically stored as binary files or as BASE64 American Standard Code for Information Interchange (ASCII) encoded files. Binary files are stored as 1s and 0s. BASE64 encoding converts the binary data into an ASCII string format. Additionally, some certificates are also encrypted to provide additional confidentiality.

The base format of certificates is Canonical Encoding Rules (**CER**) or Distinguished Encoding Rules (**DER**). CER and DER formats are defined by the International Telegraph Union Telecommunication Standardization Sector (ITU-T) in the X.690 standard. They use a variant of the Abstract Syntax Notation One (ASN.1) format, which defines data structures commonly used in cryptography. CER is a binary format and DER is an ASCII format.

DER-based certificates include headers and footers to identify the contents. As an example, the following text shows a header and a footer for a certificate:

-----BEGIN CERTIFICATE-----

MIIDdTCCA12gAwIBAgILBAAAAAABFUtaW5QwDQYJKoZIhvcNAQEFBQ

... additional ASCII Characters here...
HMUfpIBvFSDJ3gyICh3WZlXi/EjJKSZp4A==
-----END CERTIFICATE-----

Each header starts with five dashes (-----), BEGIN, a label, and five more dashes. The footer starts with five dashes, End, the same label, and five more dashes. In the previous example, the label is CERTIFICATE. Other labels include PUBLIC KEY, PRIVATE KEY, ENCRYPTED PRIVATE KEY, CERTIFICATE REQUEST, and X509 CRL. CER-based certificates are binary encoded so they do not have headers and footers.

Certificate files can have many extensions, such as .crt, .cer, .pem, .key, .p7b, .p7c, .pfx, and .p12. However, it's worth stressing that a certificate with the .cer extension doesn't necessarily mean that it is using the CER format.

Type	Common Extensions	Format	Common Purpose	Can Contain
CER	.cer	Binary	Used for binary certificates	Varies
DER	.der	ASCII	Used for ASCII certificates	Varies
PEM	.pem, .cer, .crt, .key	Binary (CER) or ASCII (DER)	Can be used for almost any certificate purpose	Server certificates, certificate chains, keys, CRL
P7B	.p7b, .p7c	ASCII (DER)	Used to share the public key	Certificates, certificate chains, CRL, but never the private key
P12 PFX	.p12, .pfx	Binary (CER)	Commonly used to store private keys with a certificate	Certificates, certificate chains, and private keys

When comparing the different formats, it's important to know what they can contain and how to identify them. Table 10.3 provides an overview of the primary formats and the following sections describe these in more depth.

Table 10.3: Certificate formats

PEM is derived from the Privacy Enhanced Mail format, but that is misleading. It implies that PEM-based certificates are used for email only. However, PEM-based certificates can be used for just about anything. They can be formatted as CER (binary files) or DER (ASCII files). They can also be used to share public keys within a certificate, request certificates from a CA as a CSR, install a private key on a server, publish a CRL, or share the full certificate chain.

You might see a PEM-encoded certificate with the .pem extension. However, it's more common for the certificate to use other extensions. For example, a PEM-encoded file holding the certificate with the public key

typically uses the .cer or .crt extension. A PEM file holding just the private key typically uses the .key extension.

P7B certificates use the PKCS version 7 (PKCS#7) format and they are DER-based (ASCII). They are commonly used to share public keys with proof of identity of the certificate holder. Recipients use the public keys to encrypt or decrypt data. For example, a web server might use a P7B certificate to share its public key. P7B certificates can also contain a certificate chain or a CRL. However, they never include the private key.

P12 certificates use the PKCS version 12 (PKCS#12) format and they are CER-based (binary). They are commonly used to hold certificates with the private key. For example, when installing a certificate on a server to support HTTPS sessions, you might install a P12 certificate with the private key. Because it holds the private key, it's common to encrypt P12 certificates. It's also possible to include the full certificate chain in a P12 certificate.

Personal Information Exchange (**PFX**) is a predecessor to the P12 certificate and it has the same usage. Administrators often use this format on Windows systems to import and export certificates.

Remember this

CER is a binary format for certificates and DER is an ASCII format. PEM is the most commonly used certificate format and can be used for just about any certificate type. P7B certificates are commonly used to share public keys. P12 and PFX certificates are commonly used to hold the private key.

Chapter 10 Exam Topic Review

When preparing for the exam, make sure you understand these key concepts covered in this chapter.

Introducing Cryptography Concepts

- Integrity provides assurances that data has not been modified. Hashing ensures that data has retained integrity.
- Confidentiality ensures that data is only viewable by authorized users. Encryption protects the confidentiality of data.
- Symmetric encryption uses the same key to encrypt and decrypt data.
- Asymmetric encryption uses two keys (public and private) created as a matched pair.
- A digital signature provides authentication, non-repudiation, and integrity.
 - Authentication validates an identity.
 - Non-repudiation prevents a party from denying an action.
 - Users sign emails with a digital signature, which is a hash of an email message encrypted with the sender's private key.
 - Only the sender's public key can decrypt the hash, providing verification it was encrypted with the sender's private key.

Providing Integrity with Hashing

- Hashing verifies the integrity of data, such as downloaded files and email messages.
- A hash (sometimes listed as a checksum) is a fixed-size string of numbers or hexadecimal characters.
- Hashing algorithms are one-way functions used to create a hash. You cannot reverse the process to re-create the original data.
- Passwords are often stored as hashes instead of the actual password. Salting the password thwarts many password attacks.
- Two commonly used key stretching techniques are bcrypt and

Password-Based Key Derivation Function 2 (PBKDF2). They protect passwords against brute force and rainbow table attacks.

- Common hashing algorithms are Message Digest 5 (MD5), Secure Hash Algorithm (SHA), and Hash-based Message Authentication Code (HMAC). HMAC provides both integrity and authenticity of a message.

Providing Confidentiality with Encryption

- Confidentiality ensures that data is only viewable by authorized users.
- Encryption provides confidentiality of data, including data-at-rest (any type of data stored on disk) or data-in-transit (any type of transmitted data).
- Block ciphers encrypt data in fixed-size blocks. Advanced Encryption Standard (AES) and Twofish encrypt data in 128-bit blocks.
- Stream ciphers encrypt data 1 bit or 1 byte at a time. They are more efficient than block ciphers when encrypting data of an unknown size or when sent in a continuous stream. RC4 is a commonly used stream cipher.
- Cipher modes include Electronic Codebook (ECB), Cipher Block Chaining (CBC), Counter (CTM) mode, and Galois/Counter Mode (GCM). ECB should not be used. GCM is widely used because it is efficient and provides data authenticity.
- Data Encryption Standard (DES), Triple DES (3DES), and Blowfish are block ciphers that encrypt data in 64-bit blocks. AES is a popular symmetric block encryption algorithm, and it uses 128, 192, or 256 bits for the key.
- Asymmetric encryption uses public and private keys as matched pairs.
 - If the public key encrypted information, only the matching private key can decrypt it.
 - If the private key encrypted information, only the matching public key can decrypt it.
 - Private keys are always kept private and never shared.
 - Public keys are freely shared by embedding them in a certificate.

- RSA is a popular asymmetric algorithm. Many cryptographic protocols use RSA to secure data such as email and data transmitted over the Internet. RSA uses prime numbers to generate public and private keys.
- Elliptic curve cryptography (ECC) is an encryption technology commonly used with small wireless devices.
- Diffie-Hellman provides a method to privately share a symmetric key between two parties. Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) is a version of Diffie-Hellman that uses ECC to re-create keys for each session.
- Steganography is the practice of hiding data within a file. You can hide messages in the white space of a file without modifying its size. A more sophisticated method is by modifying bits within a file. Capturing and comparing hashes of files can discover steganography attempts.

Using Cryptographic Protocols

- When using digital signatures with email:
 - The sender's private key encrypts (or signs).
 - The sender's public key decrypts.
- A digital signature provides authentication (verified identification) of the sender, non-repudiation, and integrity of the message.
 - Senders create a digital signature by hashing a message and encrypting the hash with the sender's private key.
 - Recipients decrypt the digital signature with the sender's matching public key.
- When encrypting email:
 - The recipient's public key encrypts.
 - The recipient's private key decrypts.
 - Many email applications use the public key to encrypt a symmetric key, and then use the symmetric key to encrypt the email contents.
- S/MIME and PGP secure email with encryption and digital signatures. They both use RSA, certificates, and depend on a PKI. They can encrypt email at rest (stored on a drive) and in transit (sent over the network).
- TLS is the replacement for SSL. SSL is deprecated and should not

be used.

- When encrypting web site traffic with TLS:
 - The web site's public key encrypts a symmetric key.
 - The web site's private key decrypts the symmetric key.
 - The symmetric key encrypts data in the session.
- Weak cipher suites (such as those supporting SSL) should be disabled to prevent downgrade attacks.

Exploring PKI Components

- A Public Key Infrastructure (PKI) is a group of technologies used to request, create, manage, store, distribute, and revoke digital certificates. A PKI allows two entities to privately share symmetric keys without any prior communication.
- Most public CAs use a hierarchical centralized CA trust model, with a root CA and intermediate CAs. A CA issues, manages, validates, and revokes certificates.
- Root certificates of trusted CAs are stored on computers. If a CA's root certificate is not in the trusted store, web users will see errors indicating the certificate is not trusted or the CA is not recognized.
- You request a certificate with a certificate signing request (CSR). You first create a private/ public key pair and include the public key in the CSR.
- CAs revoke certificates when an employee leaves, the private key is compromised, or the CA is compromised. A CRL identifies revoked certificates as a list of serial numbers.
- The CA publishes the CRL, making it available to anyone. Web browsers can check certificates they receive from a web server against a copy of the CRL to determine if a received certificate is revoked.
- Public key pinning provides clients with a list of hashes for each public key it uses.
- Certificate stapling provides clients with a timestamped, digitally signed OCSP response. This is from the CA and appended to the certificate.
- User systems return errors when a system tries to use an expired certificate.

- A key escrow stores a copy of private keys used within a PKI. If the original private key is lost or inaccessible, the copy is retrieved from escrow, preventing data loss.
- Wildcard certificates use a * for child domains to reduce the administrative burden of managing certificates. Subject Alternative Name (SAN) certificates can be used for multiple domains with different domain names.
- A domain validated certificate indicates that the certificate requestor has some control over a DNS domain. Extended validation certificates use additional steps beyond domain validation to give users a visual indication that they are accessing the site.
- CER is a binary format for certificates and DER is an ASCII format.
- PEM is the most commonly used certificate format and can be used for just about any certificate type.
- P7B certificates are commonly used to share public keys. P12 and PFX certificates are commonly used to hold the private key.

Online References

- Remember, there are additional resources at <http://gcfgapremium.com/501-extras>. They include labs, sample performance-based questions, and more.

Chapter 10 Practice Questions

1. Bart recently sent out confidential data via email to potential competitors. Management suspects he did so accidentally, but Bart denied sending the data. Management wants to implement a method that would prevent Bart from denying accountability in the future. Which of the following are they trying to enforce?
 - A. Confidentiality
 - B. Encryption
 - C. Access control
 - D. Non-repudiation
2. A software company occasionally provides application updates and patches via its web site. It also provides a checksum for each update and patch. Which of the following BEST describes the purpose of the checksum?
 - A. Availability of updates and patches
 - B. Integrity of updates and patches
 - C. Confidentiality of updates and patches
 - D. Integrity of the application
3. A one-way function converts data into a string of characters. It is not possible to convert this string of characters back to the original state. What type of function is this?
 - A. Symmetric encryption
 - B. Asymmetric encryption
 - C. Stream cipher
 - D. Hashing
4. An application developer is working on the cryptographic elements of an application. Which of the following cipher modes should NOT be used in this application?
 - A. CBC
 - B. CTM
 - C. ECB
 - D. GCM
5. The following text shows the ciphertext result of encrypting the

word “passed” with an uppercase P and a lowercase p:

- Passed!—xnBKcndl+25mHjnafwi6Jw
- passed!—RqMbHJqLdPE3RCuUU17FtA

Which of the following BEST describes the cryptography concept demonstrated by comparing the resulting ciphertext of both words?

- A. Confusion
- B. Diffusion
- C. Key stretching
- D. Security through obscurity

6. Which of the following is a symmetric encryption algorithm that encrypts data 1 bit at a time?

- A. Block cipher
- B. Stream cipher
- C. AES
- D. DES
- E. MD5

7. A supply company has several legacy systems connected within a warehouse. An external security audit discovered the company is using DES for data-at-rest. It mandated the company upgrade DES to meet minimum security requirements. The company plans to replace the legacy systems next year, but needs to meet the requirements from the audit. Which of the following is MOST likely to be the simplest upgrade for these systems?

- A. S/MIME
- B. HMAC
- C. 3DES
- D. TLS

8. Bart wants to send a secure email to Lisa, so he decides to encrypt it. Bart wants to ensure that Lisa can verify that he sent it. Which of the following does Lisa need to meet this requirement?

- A. Bart’s public key
- B. Bart’s private key
- C. Lisa’s public key
- D. Lisa’s private key

9. Bart wants to send a secure email to Lisa, so he decides to encrypt it.

He wants to ensure that only Lisa can decrypt it. Which of the following does Lisa need to decrypt Bart's email?

- A. Bart's public key
- B. Bart's private key
- C. Lisa's public key
- D. Lisa's private key

10. An organization requested bids for a contract and asked companies to submit their bids via email. After winning the bid, Acme realized it couldn't meet the requirements of the contract. Acme instead stated that it never submitted the bid. Which of the following would provide proof to the organization that Acme did submit the bid?

- A. Digital signature
- B. Integrity
- C. Repudiation
- D. Encryption

11. Application developers are creating an application that requires users to log on with strong passwords. The developers want to store the passwords in such a way that it will thwart brute force attacks. Which of the following is the BEST solution?

- A. 3DES
- B. MD5
- C. PBKDF2
- D. Database fields

12. Administrators have noticed a significant amount of OCSP traffic sent to an intermediate CA. They want to reduce this traffic. Which of the following is the BEST choice to meet this need?

- A. Pinning
- B. Digital signatures
- C. Stapling
- D. Hashing

13. A web site is using a certificate. Users have recently been receiving errors from the web site indicating that the web site's certificate is revoked. Which of the following includes a list of certificates that have been revoked?

- A. CRL
- B. CA

- C. OCSP
- D. CSR

14. An organization recently updated its security policy. One change is a requirement for all internal web servers to only support HTTPS traffic. However, the organization does not have funds to pay for this. Which of the following is the BEST solution?

- A. Create code signing certificates for the web servers.
- B. Create one wildcard certificate for all the web servers.
- C. Create a public CA and issue certificates from it.
- D. Create certificates signed by an internal private CA.

15. An administrator is installing a certificate with a private key on a server. Which of the following certificate types is he MOST likely installing?

- A. DER
- B. P12
- C. P7B
- D. CRT

Chapter 10 Practice Question

Answers

1. **D.** Non-repudiation methods such as digital signatures prevent users from denying they took an action. Encryption methods protect confidentiality. Access control methods protect access to data.
2. **B.** The checksum (also known as a hash) provides integrity for the updates and patches so that users can verify they have not been modified. Installing updates and patches increases the availability of the application. Confidentiality is provided by encryption. The checksums are for the updates and patches, so they do not provide integrity for the application.
3. **D.** A hash function creates a string of characters (typically displayed in hexadecimal) when executed against a file or message, and hashing functions cannot be reversed to re-create the original data. Encryption algorithms (including symmetric encryption, asymmetric encryption, and stream ciphers) create ciphertext from plaintext data, but they include decryption algorithms to re-create the original data.
4. **C.** The Electronic Codebook (ECB) mode of operation encrypts blocks with the same key, making it easier for attackers to crack. The other cipher modes are secure and can be used. Cipher Block Chaining (CBC) mode is used by some symmetric block ciphers, though it isn't as efficient. Counter (CTM) mode combines an initialization vector (IV) with a counter and effectively converts a block cipher into a stream cipher. Galois/Counter Mode (GCM) combines the Counter mode with hashing techniques for data authenticity and confidentiality.
5. **B.** This demonstrates diffusion because a small change in the plaintext results in a large change in the ciphertext. Confusion indicates that the ciphertext is significantly different than the plaintext. Although this is true for both results, the question is asking you to compare the two results. Key stretching techniques add salts to passwords before hashing them to thwart password cracking attacks. Security through obscurity methods use obfuscation methods to hide

data, but they don't necessarily encrypt data.

6. **B.** A stream cipher encrypts data a single bit or a single byte at a time and is more efficient when the size of the data is unknown, such as streaming audio or video. A block cipher encrypts data in specific-sized blocks, such as 64-bit blocks or 128-bit blocks. Advanced Encryption Standard (AES) and Data Encryption Standard (DES) are block ciphers. Message Digest 5 (MD5) is a hashing algorithm.

7. **C.** The best choice is Triple Data Encryption Standard (3DES). None of the other answers are valid replacements for the symmetric encryption algorithm Data Encryption Standard (DES). Secure/Multipurpose Internet Mail Extensions (S/MIME) is used to digitally sign and encrypt email. Hash-based Message Authentication Code (HMAC) is a hashing algorithm used to verify the integrity and authenticity of messages. Transport Layer Security (TLS) uses both symmetric and asymmetric encryption to encrypt data-in-transit, not data-at-rest.

8. **A.** Lisa would decrypt the digital signature with Bart's public key and verify the public key is valid by querying a Certificate Authority (CA). The digital signature provides verification that Bart sent the message, non-repudiation, and integrity for the message. Bart encrypts the digital signature with his private key, which can only be decrypted with his public key. Lisa's keys are not used for Bart's digital signature, but might be used for the encryption of the email. Although not part of this scenario, Bart would encrypt the email with Lisa's public key and Lisa would decrypt the email with Lisa's private key.

9. **D.** Lisa would decrypt the email with her private key and Bart would encrypt the email with Lisa's public key. Although not part of this scenario, if Bart wanted Lisa to have verification that he sent it, he would create a digital signature with his private key and Lisa would decrypt the private key with Bart's public key. Bart does not use his keys to encrypt email sent to someone else.

10. **A.** If Acme submitted the bid via email using a digital signature, it would provide proof that the bid was submitted by Acme. Digital signatures provide verification of who sent a message, non-repudiation preventing them from denying it, and integrity verifying the message

wasn't modified. Integrity verifies the message wasn't modified. Repudiation isn't a valid security concept. Encryption protects the confidentiality of data, but it doesn't verify who sent it or provide non-repudiation.

11. **C.** Password-Based Key Derivation Function 2 (PBKDF2) is a key stretching technique designed to protect against brute force attempts and is the best choice of the given answers. Another alternative is bcrypt. Both salt the password with additional bits. Triple DES (3DES) is an encryption protocol. Passwords stored using Message Digest 5 (MD5) are easier to crack because they don't use salts. Storing the passwords in encrypted database fields is a possible solution, but just storing them in unencrypted database fields does not protect them at all.

12. **C.** Online Certificate Status Protocol (OCSP) stapling reduces OCSP traffic sent to a Certificate Authority (CA). Certificate presenters append a timestamped, digitally signed OCSP response to a certificate. Public key pinning includes a list of public key hashes in HTTPS responses from the web server. While pinning helps validate certificates, it is unrelated to OCSP. Digital signatures won't reduce traffic. Hashing is used for integrity and it won't reduce OCSP traffic.

13. **A.** A certificate revocation list (CRL) is a list of certificates that a Certificate Authority (CA) has revoked. The CA stores a database repository of revoked certificates and issues the CRL to anyone who requests it. The Online Certificate Status Protocol (OCSP) validates trust with certificates, but only returns short responses such as good, unknown, or revoked. A certificate signing request (CSR) is used to request certificates.

14. **D.** The best solution is to use certificates signed by an internal private Certificate Authority (CA). This ensures connections use Hypertext Transfer Protocol Secure (HTTPS) instead of HTTP. Even if the organization doesn't have an internal CA, it is possible to create one on an existing server without incurring any additional costs. A code signing certificate provides a digital signature for an application or script, not an entire web server. A wildcard certificate is used for a single domain with multiple subdomains. It is not used for multiple web servers unless they all share the same root domain name, but the

scenario doesn't indicate the web servers share the same root domain name. You would not create a public CA to support internal private servers. While it is feasible to purchase certificates from a public CA, that would cost money, but the scenario indicates money isn't available.

15. **B.** P12 (PKCS #12) certificates commonly include a private key and they are used to install a private key on a server. A Distinguished Encoding Rules (DER)-based certificate is an ASCII encoded file, but P12 certificates are Canonical Encoding Rules (CER) binary encoded files. A P7B (PKCS #7) certificate never includes the private key. CRT isn't a valid certificate type, though many certificates do use the .crt extension.