Foundation Topics

Introducing Cryptographic Services

To understand cryptographic services, first you must understand the science of cryptology, which in essence is the making and breaking of secret codes. Cryptology can be broken into two distinct areas: cryptography and cryptanalysis. Cryptography is the development and use of codes. Cryptanalysis is all about the breaking of these codes. This section explores these two disciplines to give you a better understanding of cryptographic services as a whole.

Understanding Cryptology

Because cryptography is made up of two halves—the creation of codes and the attempted breaking of those codes—a natural give-and-take relationship is at play. Therefore, it is only natural that at times one side will be ahead of the other.

History offers an excellent example of this during the Hundred Years War between France and England. At that time, the cryptanalysts were ahead of the cryptographers. France believed that the Vigenère cipher was unbreakable. The British, however, cracked the code and broke it.

In another historical example, many historians now believe that the outcome of World War II largely turned on the fact that the winning side on both fronts was much more successful than the other at cracking the encryption of its enemy.

Given these examples, you might wonder who presently has the edge in this game of give and take. For the time being, conventional wisdom within the cryptology community holds that cryptographers currently have the edge. Of course, this can, and likely will, change some day.

So exactly how do we make such judgments about who is ahead or what code is unbreakable? In fact, in cryptography, it truly is impossible to prove that any given algorithm is "secure." The best that can be accomplished is to show that the algorithm is not vulnerable to any known cryptanalytic attacks. This limits our certainty to an extent, because there may be methods that have been developed but as of yet are unknown, that could crack the algorithm. The one exception to this rule is a brute-force attack.

All algorithms are vulnerable to brute force. It is simply in the nature of the attack. In other words, if every possible key is tried, one of them will surely work. The issue is time.

Depending on the complexity of the algorithm, a brute-force attack could take an inordinate amount of time to ultimately succeed. But no algorithm is truly unbreakable.

Cryptography Through the Ages

Cryptography has a long and storied past, dating back to the courts of kings, who would use early encryption to secure messages sent to other courts. Even these early times involved a degree of intrigue, because some of the courts involved would attempt to steal any message sent to an opposing kingdom.

From the courts of kings to the tools of military commanders, encryption was quickly adopted as a means of securing communications. Because messengers sometimes were killed as they transported critical military messages, encryption was seen as an indispensable means of securing these communications even if they fell into enemy hands.

Even dating back to the days of Julius Caesar, encryption was used to secure communications. Caesar's simple substitution cipher was used on the battlefield to quickly encrypt messages to his commanders. Thomas Jefferson even invented an encryption system that many historians believe he used while serving as Secretary of State from 1790 to 1793.

One of the great advances in encryption came with a machine invented by Arthur Scherbius in 1918. This machine served as a template for the systems used by each of the major participants in World War II. Scherbius called his machine the Enigma and sold it to Germany. When speaking about the security of his machine, he estimated that if 1000 cryptanalysts tested four keys per minute, all day, every day, it would take 1.8 billion years to try them all.

Throughout World War II, both the Germans and the Allies created machines modeled on the Scherbius machine. Arguably, these were the most sophisticated encryption devices ever developed. To defend against this level of encryption, the British created what most call the world's first computer, the Colossus. The Colossus was then used to break the encryption that was used by Germany's Enigma.

The Substitution Cipher

Put simply, a cipher is an algorithm for performing encryption and decryption. Typically, ciphers represent a series of well-defined steps that you can follow as a procedure. With a substitution cipher, one letter is substituted for another to encrypt a message. Substitution ciphers vary in complexity, but in their simplest form, the letter frequency of the original message is retained when character substitution is done.

As mentioned, Julius Caesar made use of a simple substitution cipher on ancient battlefields. During these times, each day would have a different key, and that key would be used to adjust the alphabet accordingly. Let's look at an example.

Let's say that the key for today is 10. In this case the letter to be substituted for A is the character in the alphabet that is ten spaces forward. In other words, to represent an A, we would substitute a K. If we follow this through the alphabet, a B would be an L, a C would be an M, and so on. To keep the nature of the substitution secret, each day the key might move a random number of places, and the process would begin again.

One of the shortcomings of this simple cipher is its vulnerability to frequency analysis. Let's say that a message has 15 occurrences of the letter B, and B is to be replaced by L. This would mean that although we are substituting the character, there would still be 15 occurrences of the letter L. So if a message were long enough, it would be vulnerable to frequency analysis. This is because the message would retain the frequency patterns found in the language even though the characters might be different.

Analysts trying to crack this cipher might look at the natural occurrence pattern for each letter in the English language. They could then use this to compare the frequency of a certain letter in the encrypted text. For instance, if the letter S appears in 20 percent of all English words, and the letter X appears in 20 percent of all the words that have been encrypted, analysts might conclude that for this cipher, X equals S. To defend against this core weakness of the substitution cipher, polyalphabetic ciphers were invented.

The Vigenère Cipher

Polyalphabetic ciphers were invented to make up for the shortcomings of the substitution cipher. The Vigenère cipher is an excellent example of this kind of cipher. It encrypts text through the use of a series of different Caesar ciphers based on the letters of a particular keyword. Although this is a simple form of polyalphabetic substitution, it still proves invulnerable to frequency analysis.

This form of encryption dates back to a book written in 1553 by Giovan Batista Belaso, although the name of this cipher came from Blaise de Vigenère, a French cryptographer. He was mistakenly credited with its invention, and to this day it carries his name.

Let's take a look at how we might use this cipher to encrypt a message. We begin with a key of SECRET. This key is then used to encode the message X MARKS THE SPOT. We encode the X by looking at the row starting with S for the letter in the X column. In this case, the X is replaced with P. Next we look for the row that begins with E for the letter M. This results in Q as our second character. To encode the full phrase, we would simply map the characters by row and column and continue our substitution.

Transposition Ciphers

If you have ever seen the beginning of the movie *Sneakers*, as the letters on-screen scramble to then become the correct words, you have a slight feel for a transposition cipher. In these ciphers, no letters are replaced; they are just rearranged. A simple form of this might take a phrase like THE QUICK BROWN FOX and simply transpose the letters so that it becomes XOFNWORBKCIUQEHT.

The Rail Fence Cipher is another kind of transposition cipher in which the words are spelled out as if they were a rail fence. The following example uses a key of three to illustrate how this could be done:

```
T...U...B...N...J...E...E...E...Y..
.H.Q.I.K.R.W.F.X.U.P.D.V.R.H.L.Z.D.G.
..E...C...O...O...M...O...T...A...O
```

To read this message, we need to follow the diagonal pattern along the rail fence. Using this form of encoding, the message THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG would be encoded as TUBNJEEEYHQIKRWFXUPDVRHLZDGECOOMOTAO. Much like the earlier example, no letters have been changed; rather, they have merely been transposed.

Working with the One-Time Pad

The one-time pad has been around for more than 90 years. It was invented and patented in 1917 by Gilbert Vernam of AT&T. The idea behind the one-time pad was to have a stream cipher that would apply the exclusive OR (XOR) operation to plain text with a key. Vernam's idea was enhanced with a contribution from Joseph Mauborgne, a captain in the U.S. Army Signal Corps, who suggested the use of random data as a key.

The one-time pad represents such a significant contribution to cryptography that the NSA has called this patent "perhaps the most important in the history of cryptography." However, using this significant idea in a real-world application has a number of difficulties.

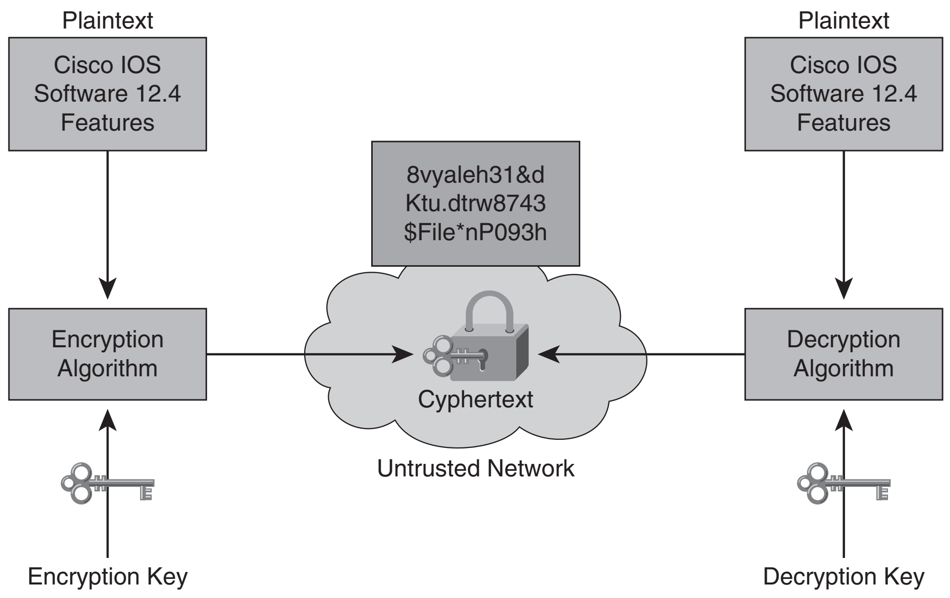
One of the more significant challenges is creating random data. On the surface, this sounds simple enough. However, because computers have a mathematical foundation, they cannot create random data. Another significant issue is that should a key be used more than once, it can easily be broken. Adding to these issues, key distribution can also be quite challenging.

One example in which the Vernam cipher has been successfully used is in RC4, which is widely used across the Internet. However, this is not a true one-time pad, because the key used is not random.

The Encryption Process

Uses of encryption are all around us, from secured online purchases to transferring data through a VPN connection. When encryption is used, some form of plain, readable text is converted to ciphertext. Ciphertext represents this text in an unreadable form, whereas decryption is the process of reversing this process. The goal of encryption is to guarantee the confidentiality of data so that only those who have authorization may read the original message. Figure 12-1 shows plain text being transformed into ciphertext.

Figure 12-1 *Plain Text Transformed into Ciphertext*



With the various older encryption algorithms we have examined, the key to their success was the secrecy of the algorithm. Today, reverse engineering is often quite simple, making this secrecy less important. Therefore, public-domain algorithms are often used. With these algorithms, successful decryption requires knowledge of the appropriate cryptographic keys. In other words, there has been a shift from the importance of the algorithm's secrecy to ensuring the secrecy of the keys.

Encryption is used to provide confidentiality in terms of the Open Systems Interconnection (OSI) layers in these ways:

Key Topic

- At the application layer, data encryption is used for secure e-mail, secure database sessions (Oracle SQL*net), and secure messaging (Lotus Notes sessions).
- At the session layer, data is encrypted using a protocol such as Secure Socket Layer (SSL) or Transport Layer Security (TLS).

- At the network layer, data is encrypted using protocols such as those that make up the IPsec protocol suite.

Cryptanalysis

When we seek to break encoded data, this undertaking is called cryptanalysis. An attacker who is attempting to break an algorithm or encrypted ciphertext may use one of a variety of attacks:

Key Topic

- Chosen plain-text attack
- Chosen ciphertext attack
- Birthday attack
- Meet-in-the-middle attack
- Brute-force attack
- Ciphertext-only attack
- Known plain-text (the usual brute-force) attack

Table 12-2 describes these attack types to help you understand their usage as a form of cryptanalysis.

Key Topic

Table 12-2 *Defining Attack Types*

Type of Attack	Description
Chosen plain-text attack	In this attack the attacker chooses what data the encryption device encrypts and then observes the ciphertext output. This is a more powerful attack than a known plain-text attack, because the attacker gets to choose the plain-text blocks to encrypt. This allows the attacker to choose plain text that could potentially yield more information about the key. However, the practicality of this attack may be called into question. This is because it is often difficult, if not impossible, to capture both the ciphertext and plain text. This would generally require that the trusted network be compromised as well, yielding access to confidential information.
Chosen ciphertext attack	In this attack, the attacker may choose different ciphertexts to be decrypted. The attacker also has access to the decrypted plain text. This combination makes it possible for an attacker to search through the keyspace and determine which key decrypts the chosen ciphertext. This attack is somewhat like the chosen plain-text attack and, like that attack, it may not be too practical. Capturing both the ciphertext and plain text without first breaking into the trusted network would prove nearly impossible.

Table 12-2 *Defining Attack Types*

Key Topic

Type of Attack	Description
Birthday attack	The birthday attack derives its name from the statistical probability involved in two individuals in a group having the same birthday. Statisticians say that in a group of 23 individuals, the likelihood that two people will have the same birthday is greater than 50 percent. This attack is a form of brute-force attack focused on hash functions. If a given function, when supplied with a random input, returns one of *k* equally likely values, repeating the function with various inputs, the same output would be expected after $1.2k^{1/2}$ times.
Meet-in-the-middle attack	This is a known plain-text attack in which the attacker knows a portion of the plain text and the corresponding ciphertext. The attacker encrypts the plain text with every possible key, and the results are stored. The attacker then decrypts the ciphertext using every key until one of the results matches one of the stored values.
Brute-force attack	All encryption algorithms are vulnerable to a brute-force attack. In this attack, an attacker tries every possible key with the decryption algorithm. Generally, a brute-force attack will succeed about 50 percent of the way through the keyspace. To defend against this form of attack, modern cryptographers have to create a sufficiently large keyspace so that attacking it in this way requires too much time and money to be practical.
Ciphertext-only attack	In this form of attack, the attacker has the ciphertext of several messages. Each of these messages has been encrypted using the same encryption algorithm. However, the attacker has no knowledge of the underlying plain text. To be successful, the attacker must recover the ciphertext of as many messages as possible. Alternatively, the attacker could deduce the key or keys used to encrypt the messages and use this to decrypt other messages encrypted with the same keys. This could be achieved using statistical analysis. Today, however, these attacks are no longer practical, because modern algorithms produce pseudorandom output that is resistant to statistical analysis.
Known plain-text attack	In this attack, like the ciphertext-only attack, the attacker has access to the ciphertext of several messages, and he also knows something about the plain text underlying that ciphertext. Knowing the underlying protocol, file type, and some characteristic strings that may appear in the plain text, an attacker then employs a brute-force attack to try keys until decryption with the correct key succeeds. Compared to other attacks, this may be the most practical attack. Attackers can usually assume the type and some features of the underlying plain text after capturing the ciphertext. Although it is more practical, the enormous keyspaces employed by modern algorithms make it unlikely that this attack will succeed.

Understanding the Features of Encryption Algorithms

Good encryption algorithms have several benefits:

Key Topic

- They are resistant to cryptographic attacks.
- They support variable and long key lengths and scalability.
- They create an avalanche effect.
- They have no export or import restrictions.

When an attacker sets out to penetrate data protected by a cryptographic algorithm, the best way to do so is to try to decrypt the data using all the possible keys. Of course, the time required to undertake such an attack is determined by the number of possible keys. In practical terms, this process could take quite a long time. In fact, when appropriately long keys are used, this form of attack generally is infeasible.

A couple of other desirable attributes of a good cryptographic algorithm are variable key lengths and scalability. It stands to reason that the longer the key used for encryption, the longer it will take for an attacker to break it. Having the scalability provided by flexible key lengths lets you select the strength and speed of encryption that you need. Let's compare a couple of possible key lengths.

If we were to use a 16-bit key (nowhere near the strongest possible), there would be 65,536 possible keys. This may sound like a large number of keys, but consider that a 56-bit key would yield $7.2 * 10^{16}$ possible keys. As you can see, variable key lengths can provide an ever-increasing number of keys, creating ever-stronger levels of encryption.

Another desirable attribute is called the avalanche effect. It says that changing only a few bits of a plain-text message causes its ciphertext to be completely different. An encryption algorithm that provides the avalanche effect makes it possible for messages that are quite similar to be sent over an untrusted medium, because the encrypted (ciphertext) messages remain completely different.

Export and import restrictions must also be carefully considered when you use encryption internationally. Certain countries prohibit the export of encryption algorithms or allow only the export of algorithms with smaller (more easily broken) keys. Other countries have strict regulations and restrictions governing the import of cryptographic algorithms. Before importing or exporting a cryptographic algorithm internationally, it is best to check with the governments involved to better understand their laws and regulations.

The U.S. has specific restrictions for the export of cryptographic algorithms, but in January 2000 these restrictions were substantially relaxed. At present, any cryptographic product may be exported under a license exception unless the end users are governments outside the U.S. or are among those nations that have an embargo in place. To learn more about current practices for the import and export of cryptographic algorithms in the U.S., visit http://www.commerce.gov.

Symmetric and Asymmetric Encryption Algorithms

This section discusses both symmetric and asymmetric algorithms, noting their differences and uses. Most striking among these two widely used types of encryption algorithms is their differences in key usage. Whereas symmetric encryption algorithms use a single key for both encryption and decryption, asymmetric encryption algorithms employ two separate keys—one for encryption and the other for decryption. Other differences will also be discussed with regard to the algorithms' speed and complexity.

Encryption Algorithms and Keys

Ciphers are two-part mathematical functions that encrypt and decrypt data. Exposure of the algorithm itself could compromise the security of the encryption system if it is based on the algorithm's secrecy. If this should happen, each party working with the algorithm must change it. However, this is a dated view of cryptography. In modern cryptography, all algorithms are public, and complex cryptographic keys are used to ensure the secrecy of the data.

Cryptographic keys are created from sequences of bits that, together with the data that will be encrypted, are input into an encryption algorithm. Table 12-3 describes the two classes of encryption algorithms and details their use of keys.

Table 12-3 *Classes of Encryption Algorithms*

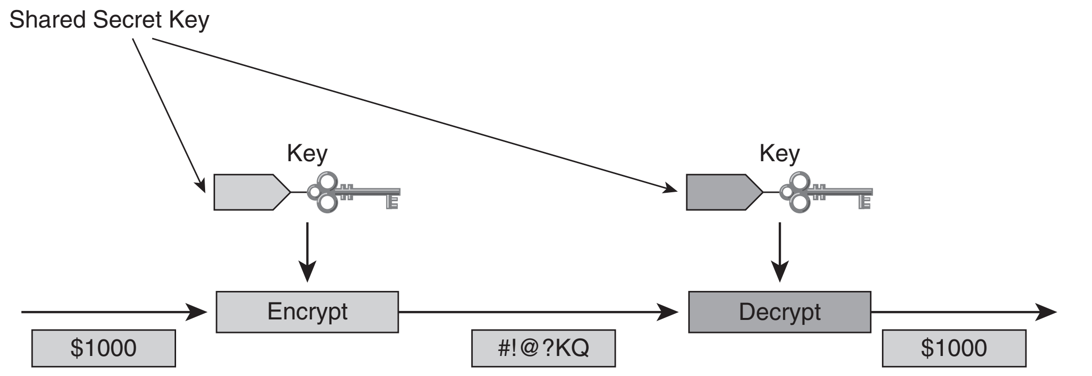
Key Topic

Class of Algorithm	Description
Symmetric encryption algorithms	This class of algorithm employs the same key to both encrypt and decrypt data.
Asymmetric encryption algorithms	This class of algorithm employs two separate keys. One key is used to encrypt data, and the other key is used to decrypt data.

Symmetric Encryption Algorithms

As shown in Table 12-3, symmetric encryption algorithms use the same key for both encryption and decryption. This means that both the sender and receiver must share the same secret key to transfer data securely. Figure 12-2 shows how symmetric encryption encrypts and decrypts data.

Figure 12-2 *Symmetric Encryption and Decryption Process*



- The sender and receiver must share a secret key.
- The usual key length is 40-256 bits.
- Examples of symmetric encryption algorithms are DES, 3DES, AES, IDEA, RC2/4/5/6, and Blowfish.

For a symmetric algorithm to be secure, the key itself must remain a secret. Should this key become available, anyone holding it could encrypt and decrypt messages. Because of this need for security, symmetric encryption is frequently called secret-key encryption. Symmetric encryption represents the more traditional form of cryptography. It uses key lengths ranging from 40 to 256 bits.

A number of well-known symmetric encryption algorithms exist. Table 12-4 details some of these, along with their key sizes.

Key Topic

Table 12-4 *Popular Symmetric Algorithms*

Symmetric Algorithm	**Key Size**
DES	56-bit keys
Triple Data Encryption Standard (3DES)	112-bit and 168-bit keys
AES	128-bit, 192-bit, and 256-bit keys
International Data Encryption Algorithm (IDEA)	128-bit keys
RC2	40-bit and 64-bit keys
RC4	1-bit to 256-bit keys
RC5	0-bit to 2040-bit keys
RC6	128-bit, 192-bit, and 256-bit keys
Blowfish	32-bit to 448-bit keys

Symmetric encryption cryptography uses a number of different techniques. The most common are

Key Topic

- Block ciphers
- Stream ciphers
- Message Authentication Codes (MAC)

Symmetric algorithms generally are quite fast and therefore are a frequent choice to provide wire-speed encryption in data networks. Because symmetric algorithms are based on less-complex mathematical operations, they can be readily accelerated through the use of hardware. Due to their speed, symmetric algorithms may be used for bulk encryption when data privacy is required. One common example of their practical usage in this regard is to protect a VPN.

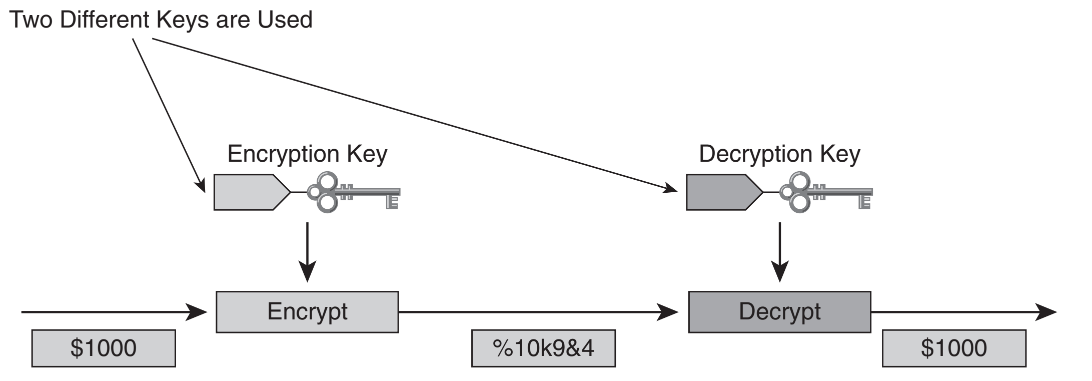
Despite the benefit of their speed, symmetric encryption algorithms do present a challenge with regard to key management. In particular, all parties involved in the communication must exchange the secret key over a secure channel before the encryption process can begin. This means that the security of any cryptographic system hinges on the ability of the key exchange method to protect the keys. Given this need, symmetric algorithms often are used to provide encryption services while additional key management algorithms are used to secure the key exchange.

Asymmetric Encryption Algorithms

Unlike symmetric algorithms, which use the same key for both encryption and decryption, asymmetric algorithms, often called public-key algorithms, use two different keys. One key is used for encryption, and the other is used for decryption. A central facet of this design is that the decryption key cannot feasibly be calculated from the encryption key, and vice versa. Figure 12-3 shows the encryption and decryption process using an asymmetric encryption algorithm.

With asymmetric algorithms, key lengths generally range from 512 to 4096 bits. However, no direct comparison can be made between the key length of asymmetric and symmetric algorithms, because the underlying design of these algorithm classes is quite different. In terms of resistance to brute-force attacks, experts generally agree that an RSA encryption key of 2048 bits generally is equivalent in strength to an RC4 key of only 128 bits.

Figure 12-3 *Encrypting and Decrypting with Asymmetric Algorithms*



- Asymmetric encryption algorithms are best known as key algorithms.
- The usual key length is 512-4096 bits.
- Examples of asymmetric encryption algorithms are RSA, ElGamal, elliptic curves, and Diffie-Hellman.

One downside of symmetric algorithms is that they can be up to 1000 times slower than symmetric algorithms. This is because their design is based on complex mathematical calculations. Often these designs employ such things as factoring extremely large numbers or computing discrete logarithms of extremely large numbers.

Because of the issues with the speed of these algorithms, they generally are used in low-volume cryptographic mechanisms. For instance, they might be employed in digital signatures or for key exchange. One noted benefit is that key management of these algorithms generally is less complex than for symmetric algorithms. This stems from the fact that typically one of the two keys, either the encryption or decryption key, may be made public.

The Difference Between Block and Stream Ciphers

Both block and stream ciphers have their place in encryption algorithms as a mechanism for generating ciphertext from plain text. This section explores their basic differences and their uses in modern cryptography.

Block Ciphers

Block ciphers derive their name from the fact that they transform a fixed-length "block" of plain text into a "block" of ciphertext. These two blocks are of the same length. When the reverse transformation is applied to the ciphertext block, by using the same secret key, it is decrypted. Block ciphers use a fixed length or block size. Generally this is 128 bits, but they can range in size. For instance, DES has a block size of 64 bits.

The concept of block size determines how much data may be encrypted at a given time. This varies according to key length, because the key length refers to the size of the encryption key. To use the example from earlier, DES encrypts blocks in 64-bit chunks but does so using a 56-bit key length.

Because ciphertext must always be a multiple of the block size, the output data from a block cipher is larger than the input data. Block algorithms work with data one chunk at a time, such as 8 bytes, and then use padding to add artificial data (blanks) should there be less input data than one full block. Here are some of the more common block ciphers:

Key Topic

- DES and 3DES, running in Electronic Code Book (ECB) or Cipher Block Chaining (CBC) mode
- Skipjack
- Blowfish
- RSA
- AES
- IDEA
- Secure and Fast Encryption Routine (SAFER)

Stream Ciphers

Stream ciphers use smaller units of plain text than what are used with block ciphers; typically they work with bits. Transformation of these smaller plain-text units also varies, depending on when during the encryption process they are encountered. One of the great benefits of stream ciphers compared to block ciphers is that they are much faster and generally do not increase the message size. This is because they can encrypt an arbitrary number of bits.

Here are some common stream ciphers:

Key Topic

- RC4
- DES and 3DES, running in output feedback (OFB) or cipher feedback (CFB) mode
- Software Encryption Algorithm (SEAL)

Exploring Symmetric Encryption

Encryption algorithms use encryption keys to provide confidentiality of encrypted data. With symmetric encryption algorithms, the same key is used to encrypt and decrypt data. This section explores the principles that underlie symmetric encryption. It also examines some of the major symmetric encryption algorithms and discusses the means by which they operate, their strengths, and their weaknesses.

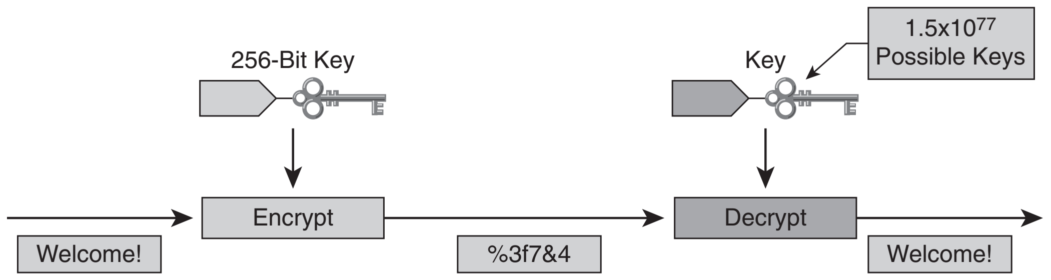
Functionality of Symmetric Encryption Algorithms

Because of the simplicity of their mathematics and the speed at which they operate, symmetric algorithms are the most commonly used form of cryptography. Symmetric encryption algorithms are also stronger. Therefore, they can use shorter key lengths compared to asymmetric algorithms. This helps increase their speed of execution in software.

Key Lengths

Key lengths for current symmetric algorithms range from 40 to 256 bits, giving symmetric algorithms keyspaces that range from 2^{40} (1,099,511,627,776) possible keys to 2^{256} ($1.5 * 10^{77}$) possible keys. As discussed previously, a large key space is central to determining how vulnerable an algorithm will be to a brute-force attack. Figure 12-4 shows a symmetric algorithm with 2^{256} possible keys.

Figure 12-4 *Key Lengths for Symmetric Encryption*



At the low end, a key length of 40 bits may be easily broken using a brute-force attack. On the other hand, if your key length is 256 bits, it is not likely that a brute-force attack will succeed. The keyspace generated with a 256-bit key is simply too large to easily fall victim to a brute-force attack.

Table 12-5 illustrates ongoing expectations for key lengths, assuming that the algorithms are mathematically and cryptographically sound. A further assumption in such calculations is that computing power will continue to keep pace with its present rate of growth and that capacity to perform brute-force attacks will also increase at the same rate. Note that if a method other than brute-force is discovered to crack a given algorithm, the key lengths in the table become obsolete.