Chapter 15

Combining Block Ciphers

There are many ways to combine block algorithms to get new algorithms. The impetus behind these schemes is to try to increase security without going through the trouble of designing a new algorithm. DES is a secure algorithm; it has been cryptanalyzed for a good 20 years and the most practical way to break it is still brute force. However, the key is too short. Wouldn't it be nice to use DES as a building block for another algorithm with a longer key? We'd have the best of both worlds: the assurance of two decades of cryptanalysis plus a long key.

Multiple encryption is one combination technique: using an algorithm to encrypt the same plaintext block multiple times with multiple keys. Cascading is like multiple encryption, but uses different algorithms. There are other techniques.

Encrypting a plaintext block twice with the same key, whether with the same algorithm or a different one, is not smart. For the same algorithm, it does not affect the complexity of a brute-force search. (Remember, you assume a cryptanalyst knows the algorithm including the number of encryptions used.) For different algorithms, it may or may not. If you are going to use any of the techniques in this chapter, make sure the multiple keys are different and independent.

15.1 Double Encryption

A naïve way of improving the security of a block algorithm is to encrypt a block twice with two different keys. First encrypt a block with the first key, then encrypt the resulting ciphertext with the second key. Decryption is the reverse process.

$$C = E_{K_2}(E_{K_1}(P))$$
$$P = D_{K_1}(D_{K_2}(C))$$

If the block algorithm is a group (see Section 11.3), then there is always a K_3 such that

$$C = E_{K_2}(E_{K_1}(P)) = E_{K_3}(P)$$

If this is not the case, the resultant doubly-encrypted ciphertext block should be much harder to break using an exhaustive search. Instead of 2^n attempts (where n is the bit length of the key), it would require 2^{2n} attempts. If the algorithm is a 64-bit algorithm, the doubly-encrypted ciphertext would require 2^{128} attempts to find the key.

This turns out not to be true for a known-plaintext attack. Merkle and Hellman [1075] developed a time-memory trade-off that could break this double-encryption scheme in 2^{n+1} encryptions, not in 2^{2n} encryptions. (They showed this for DES, but the result can be generalized to any block algorithm.) The attack is called a **meet-in-the-middle attack**; it works by encrypting from one end, decrypting from the other, and matching the results in the middle.

In this attack, the cryptanalyst knows P_1, C_1, P_2, and C_2, such that

$$C_1 = E_{K_2}(E_{K_1}(P_1))$$
$$C_2 = E_{K_2}(E_{K_1}(P_2))$$

For each possible K, he computes $E_K(P_1)$ and stores the result in memory. After collecting them all, he computes $D_K(C_1)$ for each K and looks for the same result in memory. If he finds it, it is possible that the current key is K_2 and the key in memory is K_1. He tries encrypting P_2 with K_1 and K_2; if he gets C_2 he can be pretty sure (with a probability of success of 1 in 2^{2m-2n}, where m is the block size) that he has both K_1 and K_2. If not, he keeps looking. The maximum number of encryption trials he will probably have to run is $2*2^n$, or 2^{n+1}. If the probability of error is too large, he can use a third ciphertext block to get a probability of success of 1 in 2^{3m-2n}. There are still other optimizations [912].

This attack requires a lot of memory: 2^n blocks. For a 56-bit algorithm, this translates to 2^{56} 64-bit blocks, or 10^{17} bytes. This is still considerably more memory storage than one could comfortably comprehend, but it's enough to convince the most paranoid of cryptographers that double encryption is not worth anything.

For a 128-bit key, the amount of memory required is an enormous 10^{39} bytes. If we assume that a way exists to store a bit of information on a single atom of aluminum, the memory device required to launch this attack would be a cube of solid aluminum over a kilometer on a side. And then you need some place to put it! The meet-in-the middle attack seems infeasible for keys this size.

Another double-encryption method, sometimes called **Davies-Price**, is a variant of CBC [435].

$$C_i = E_{K_1}(P_i \oplus E_{K_2}(C_{i-1}))$$
$$P_i = D_{K_1}(C_i) \oplus E_{K_2}(C_{i-1})$$

They claim "no special virtue of this mode," but it seems to be vulnerable to the same meet-in-the-middle attacks as other double-encryption modes.

15.2 TRIPLE ENCRYPTION

Triple Encryption with Two Keys

A better idea, proposed by Tuchman in [1551], operates on a block three times with two keys: with the first key, then with the second key, and finally with the first key again. He suggested that the sender first encrypt with the first key, then decrypt with the second key, and finally encrypt with the first key. The receiver decrypts with the first key, then encrypts with the second key, and finally decrypts with the first key.

$$C = E_{K_1}(D_{K_2}(E_{K_1}(P)))$$
$$P = D_{K_1}(E_{K_2}(D_{K_1}(C)))$$

This is sometimes called **encrypt-decrypt-encrypt (EDE)** mode [55]. If the block algorithm has an n-bit key, then this scheme has a $2n$-bit key. The curious encrypt-decrypt-encrypt pattern was designed by IBM to preserve compatibility with conventional implementations of the algorithm: Setting the two keys equal to each other is identical to encrypting once with the key. There is no security inherent in the encrypt-decrypt-encrypt pattern, but this mode has been adopted to improve the DES algorithm in the X9.17 and ISO 8732 standards [55,761].

K_1 and K_2 alternate to prevent the meet-in-the-middle attack previously described. If $C = E_{K_2}(E_{K_1}(E_{K_1}(P)))$, then a cryptanalyst could precompute $E_{K_1}(E_{K_1}(P)))$ for every possible K_1 and then proceed with the attack. It only requires 2^{n+2} encryptions.

Triple encryption with two keys is not susceptible to the same meet-in-the-middle attack described earlier. But Merkle and Hellman developed another time-memory trade-off that could break this technique in 2^{n-1} steps using 2^n blocks of memory [1075].

For each possible K_2, decrypt 0 and store the result in memory. Then, decrypt 0 with each possible K_1 to get P. Triple-encrypt P to get C, and then decrypt C with K_1. If that decryption is a decryption of 0 with a K_2 (stored in memory), the K_1 K_2 pair is a possible candidate. Check if it is right. If it's not, keep looking.

This is a chosen-plaintext attack, requiring an enormous amount of chosen plaintext to mount. It requires 2^n time and memory, and 2^m chosen plaintexts. It is not very practical, but it is a weakness.

Paul van Oorschot and Michael Wiener converted this to a known-plaintext attack, requiring p known plaintexts. This example assumes EDE mode.

(1) Guess the first intermediate value, a.

(2) Tabulate, for each possible K_1, the second intermediate value, b, when the first intermediate value is a, using known plaintext:

$$b = D_{K_1}(C)$$

where C is the resulting ciphertext from a known plaintext.

(3) Look up in the table, for each possible K_2, elements with a matching second intermediate value, b:

$$b = E_{K_2}(a)$$

(4) The probability of success is p/m, where p is the number of known plaintexts and m is the block size. If there is no match, try another a and start again.

The attack requires $2^{n+m}/p$ time and p memory. For DES, this is $2^{120}/p$ [1558]. For p greater than 256, this attack is faster than exhaustive search.

Triple Encryption with Three Keys

If you are going to use triple encryption, I recommend three different keys. The key length is longer, but key storage is usually not a problem. Bits are cheap.

$$C = E_{K_3}(D_{K_2}(E_{K_1}(P)))$$
$$P = D_{K_1}(E_{K_2}(D_{K_3}(C)))$$

The best time-memory trade-off attack takes 2^{2n} steps and requires 2^n blocks of memory; it's a meet-in-the-middle attack [1075]. Triple encryption, with three independent keys, is as secure as one might naïvely expect double encryption to be.

Triple Encryption with Minimum Key (TEMK)

There is a secure way of using triple encryption with two keys that prevents the previous attack, called Triple Encryption with Minimum Key (TEMK) [858]. The trick is to derive three keys from two: X_1 and X_2:

$$K_1 = E_{X_1}(D_{X_2}(E_{X_1}(T_1)))$$
$$K_2 = E_{X_1}(D_{X_2}(E_{X_1}(T_2)))$$
$$K_3 = E_{X_1}(D_{X_2}(E_{X_1}(T_3)))$$

T_1, T_2, and T_3 are constants, which do not have to be secret. This is a special construction that guarantees that for any particular pair of keys, the best attack is a known-plaintext attack.

Triple-Encryption Modes

It's not enough to just specify triple encryption; there are several ways to do it. The decision of which to use affects both security and efficiency.

Here are two possible triple-encryption modes:

Inner-CBC: Encrypt the entire file in CBC mode three different times (see Figure 15.1a). This requires three different IVs.

$$C_i = E_{K_3}(S_i \oplus C_{i-1});\ S_i = D_{K_2}(T_i \oplus S_{i-1});\ T_i = E_{K_1}(P_i \oplus T_{i-1})$$
$$P_i = T_{i-1} \oplus D_{K_1}(T_i);\ T_i = S_{i-1} \oplus E_{K_2}(S_i);\ S_i = C_{i-1} \oplus D_{K_3}(C_i)$$

C_0, S_0, and T_0 are IVs.

Outer-CBC: Triple-encrypt the entire file in CBC mode (see Figure 15.1b). This requires one IV.

$$C_i = E_{K_3}(D_{K_2}(E_{K_1}(P_i \oplus C_{i-1})))$$
$$P_i = C_{i-1} \oplus D_{K_1}(E_{K_2}(D_{K_3}(C_i)))$$

Both modes require more resources than single encryption: more hardware or more time. However, given three encryption chips, the throughput of inner-CBC is no slower than single encryption. Since the three CBC encryptions are independent, three chips can be kept busy all the time, each feeding back into itself.

On the other hand, outer-CBC feedback is outside the three encryptions. This means that even with three chips, the throughput is only one-third that of single

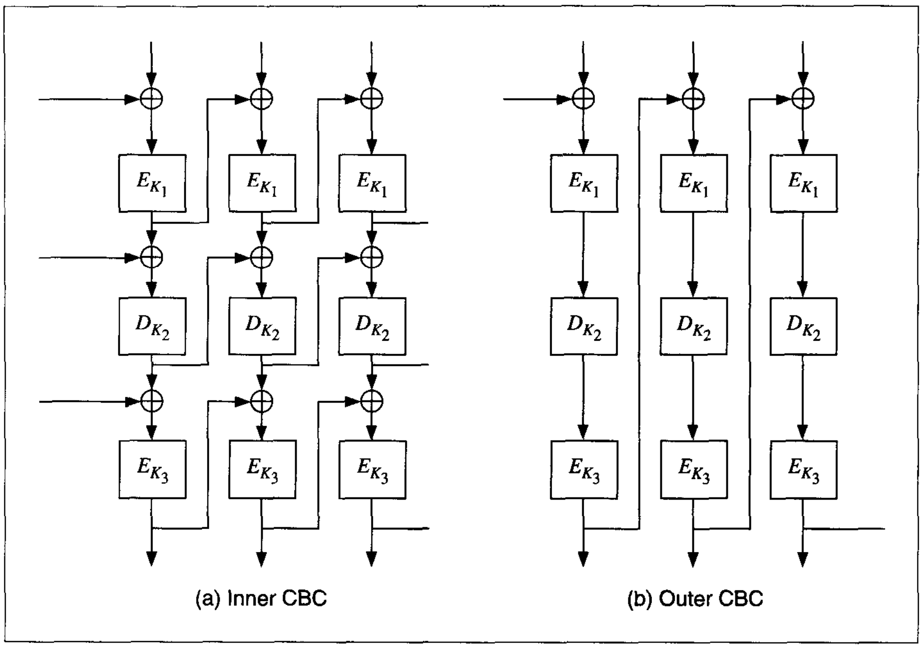

Figure 15.1 Triple encryption in CBC mode.

encryption. To get the same throughput with outer-CBC, you need to interleave IVs (see Section 9.12):

$$C_i = E_{K_3}(D_{K_2}(E_{K_1}(P_i \oplus C_{i-3})))$$

In this case C_0, C_{-1}, and C_{-2} are IVs. This doesn't help software implementations any, unless you have a parallel machine.

Unfortunately, the simpler mode is also the least secure. Biham analyzed various modes with respect to chosen-ciphertext differential cryptanalysis and found that inner-CBC is only slightly more secure than single encryption against a differential attack. If you think of triple encryption as a single larger algorithm, then inner feedbacks allow the introduction of external and known information into the inner workings of the algorithm; this facilitates cryptanalysis. The differential attacks require enormous amounts of chosen ciphertext to mount and are not very practical, but the results should be enough to give the most paranoid pause. Another analysis against meet-in-the-middle and brute-force attacks concludes that they are all equally secure [806].

There are other modes, as well. You can encrypt the entire file once in ECB, then twice in CBC; or once in CBC, once in ECB, and once in CBC; or twice in CBC and once in ECB. Biham showed that these variants are no more secure than single DES against a chosen-plaintext differential cryptanalysis attack [162]. And he doesn't have high hopes for any other variants. If you are going to use triple encryption, use modes with outer feedback.

Variants on Triple Encryption

Before there were proofs that DES does not form a group, several schemes were proposed for multiple encryption. One way to guarantee that triple encryption doesn't reduce to single encryption is to change the effective block size. One simple method is to add a bit of padding. Pad the text with a string of random bits, half a block in length, between the first and second and between the second and third encryptions (see Figure 15.2). If p is the padding function, then:

$$C = E_{K_3}(p(E_{K_2}(p(E_{K_1}(P)))))$$

This padding not only disrupts patterns, but also overlaps encrypted blocks like bricks. It only adds one block to the length of the message.

Another technique, proposed by Carl Ellison, is to use some kind of keyless permutation function between the three encryptions. The permutation could work on large blocks—8 kilobytes or so—and would effectively give this variant a block size of 8 kilobytes. Assuming that the permutation is fast, this variant is not much slower than basic triple encryption.

$$C = E_{K_3}(T(E_{K_2}(T(E_{K_1}(P)))))$$

T collects a block of input (up to 8 kilobytes in length) and uses a pseudo-random-number generator to transpose it. A 1-bit change in the input causes 8 changed output bytes after the first encryption, up to 64 changed output bytes after the second encryption, and up to 512 changed output bytes after the third encryption. If each block algorithm is in CBC mode, as originally proposed, then the effect of a single

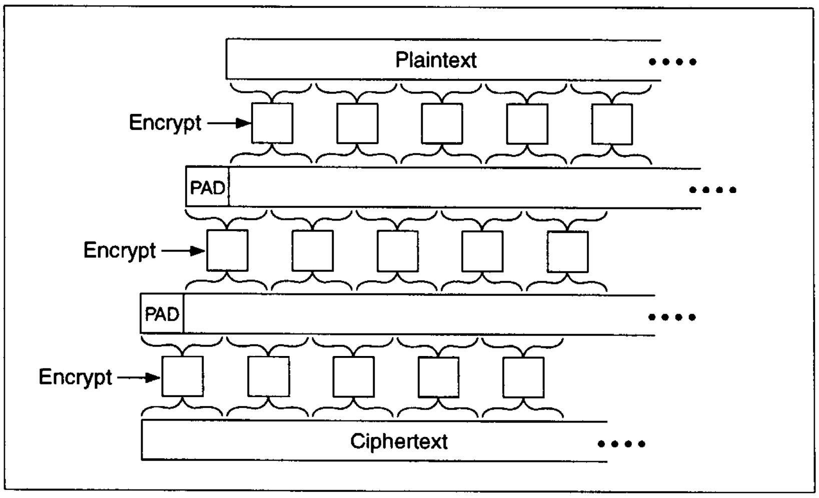

Figure 15.2 Triple encryption with padding.

changed input bit is likely to be the entire 8 kilobyte block, even in blocks other than the first.

The most recent variant of this scheme responded to Biham's attack on inner-CBC by including a whitening pass to hide plaintext patterns. That pass is a stream XOR with a cryptographically secure random-number generator called R below. The T on either side of it prevents the cryptanalyst from knowing *a priori* which key was used to encrypt any given byte on input to the last encryption. The second encryption is labelled nE (encryption with one of n different keys, used cyclically):

$$C = E_{K_3}(R(T(nE_{K_2}(T(E_{K_1}(R))))))$$

All encryptions are in ECB mode and keys are provided at least for the $n + 2$ encryption keys and the cryptographically secure random-number generator.

This scheme was proposed with DES, but works with any block algorithm. I know of no analysis of the security of this scheme.

15.3 Doubling the Block Length

There is some argument in the academic community whether a 64-bit block is long enough. On the one hand, a 64-bit block length only diffuses plaintext over 8 bytes of ciphertext. On the other hand, a longer block length makes it harder to hide patterns securely; there is more room to make mistakes.

Some propose doubling the block length of an algorithm using multiple encryptions [299]. Before implementing any of these, look for the possibility of meet-in-the-middle attacks. Richard Outerbridge's scheme [300], illustrated in Figure 15.3, is no more secure than single-block, two-key triple encryption [859].

However, I advise against this sort of thing. It isn't faster than conventional triple encryption: six encryptions are still required to encrypt two blocks of data. We know the characteristics of triple encryption; constructions like this often have hidden problems.

15.4 Other Multiple Encryption Schemes

The problem with two-key triple encryption is that it only doubles the size of the keyspace, but it requires three encryptions per block of plaintext. Wouldn't it be nice to find some clever way of combining two encryptions that would double the size of the keyspace?

Double OFB/Counter

This method uses a block algorithm to generate two keystreams, which are then used to encrypt the plaintext.

$$S_i = E_{K_1}(S_{i-1} \oplus I_1);\ I_1 = I_1 + 1$$
$$T_i = E_{K_2}(T_{i-1} \oplus I_2);\ I_2 = I_2 + 1$$
$$C_i = P_i \oplus S_i \oplus T_i$$

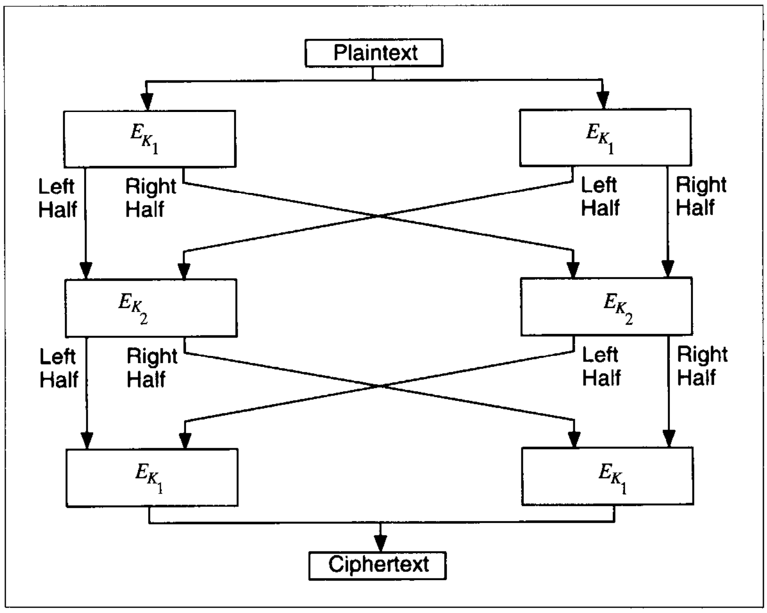

Figure 15.3 Doubling the block length.

S_i and T_i are internal variables, and I_1 and I_2 are counters. Two copies of the block algorithm run in a kind of hybrid OFB/counter mode, and the plaintext, S_i, and T_i are XORed together. The two keys, K_1 and K_2, are independent. I know of no cryptanalysis of this variant.

ECB + OFB

This method was designed for encrypting multiple messages of a fixed length, for example, disk blocks [186,188]. Use two keys: K_1 and K_2. First, use the algorithm and K_1 to generate a mask of the required block length. This mask will be used repeatedly to encrypt messages with the same keys. Then, XOR the plaintext message with the mask. Finally, encrypt the XORed plaintext with the algorithm and K_2 in ECB mode.

This mode has not been analyzed outside the paper in which it was proposed. Clearly it is at least as strong as a single ECB encryption and may be as strong as two passes with the algorithm. Possibly, a cryptanalyst could search for the two keys independently, if several known plaintext files are encrypted with the same key.

To thwart analysis of identical blocks in the same positions of different messages, you can add an IV. Unlike an IV in any other mode, here the IV is XORed with every block of the message before ECB encryption.

Matt Blaze designed this mode for his UNIX Cryptographic File System (CFS). It is a nice mode because the latency is only one encryption in ECB mode; the mask can be generated once and stored. In CFS, DES is the block algorithm.

xDES[i]

In [1644,1645], DES is used as a building block for a series of block algorithms with both larger key sizes and larger block sizes. These constructions do not depend on DES in any way and can be used with any block algorithm.

The first, $xDES^1$, is simply a Luby-Rackoff construction with the block cipher as the underlying function (see Section 14.11). The block size is twice the size of the underlying block cipher and the key size is three times the size of the underlying block cipher. In each of 3 rounds, encrypt the right half with the block algorithm and one of the keys, XOR the result with the left half, and swap the two halves.

This is faster than conventional triple encryption, since three encryptions encrypt a block twice as large as the underlying algorithm. But there is also a simple meet-in-the-middle attack that finds the key with a table the size of 2^k, where k is the key size of the underlying algorithm. Encrypt the right half of a plaintext block with all possible values of K_1, XOR the left half of the plaintext, and store these values in a table. Then, encrypt the right half of the ciphertext with all possible values of K_3 and look for a match in the table. If you find one, the key pair K_1 and K_3 are possible candidates for the right key. Repeat the attack a few times, and only one candidate will remain. This shows that $xDES^1$ is not an ideal solution. Even worse, there is a chosen plaintext attack that proves $xDES^1$ is not much stronger than the underlying block cipher [858].

$xDES^2$ extends this idea to a 5-round algorithm with a block size 4 times that of the underlying block cipher and a key size 10 times that of the underlying block cipher. Figure 15.4 is one round of $xDES^2$; each of the four sub-blocks are the size of the underlying block ciphers and all 10 keys are independent.

This scheme is also faster than triple encryption: Ten encryptions are used to encrypt a block four times the size of the underlying block cipher. However, it is vulnerable to differential cryptanalysis [858] and should not be used. The scheme is even vulnerable if DES with independent round keys is used.

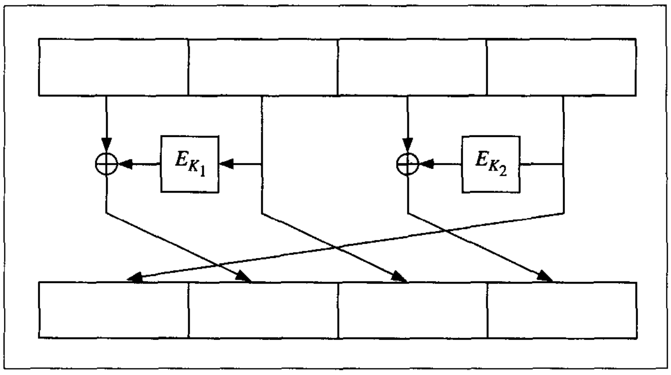

Figure 15.4 One round of $xDES^2$.

For $i \geq 3$, xDES^i is probably too big to be useful as a block algorithm. For example, the block size for xDES^3 is 6 times that of the underlying cipher, the key size is 21 times, and 21 encryptions are required to encrypt a block 6 times that of the underlying block cipher. Triple encryption is faster.

Quintuple Encryption

If triple encryption isn't secure enough—perhaps you need to encrypt triple-encryption keys using an even stronger algorithm—then higher multiples might be in order. Quintuple encryption is very strong against meet-in-the-middle attacks. (Similar arguments to the ones used with double encryption can show that quadruple encryption provides minimal security improvements over triple encryption.)

$$C = E_{K_1}(D_{K_2}(E_{K_3}(D_{K_2}(E_{K_1}(P)))))$$
$$P = D_{K_1}(E_{K_2}(D_{K_3}(E_{K_2}(D_{K_1}(C)))))$$

This construction is backwards compatible with triple encryption if $K_2 = K_3$, and is backwards compatible with single encryption if $K_1 = K_2 = K_3$. Of course, it would be even stronger if all five keys were independent.

15.5 CDMF KEY SHORTENING

This method was designed by IBM for their Commercial Data Masking Facility or CDMF (see Section 24.8) to shrink a 56-bit DES key to a 40-bit key suitable for export [785]. It assumes that the original DES key includes the parity bits.

(1) Zero the parity bits: bits 8, 16, 24, 32, 40, 48, 56, 64.
(2) Encrypt the output of step (1) with DES and the key 0xc408b0540ba1e0ae, and XOR the result with the output of step (1).
(3) Take the output of step (2) and zero the following bits: 1, 2, 3, 4, 8, 16, 17, 18, 19, 20, 24, 32, 33, 34, 35, 36, 40, 48, 49, 50, 51, 52, 56, 64.
(4) Encrypt the output of step (3) with DES and the following key: 0xef2c041ce6382fe6. This key is then used for message encryption.

Remember that this method shortens the key length, and thereby weakens the algorithm.

15.6 WHITENING

Whitening is the name given to the technique of XORing some key material with the input to a block algorithm, and XORing some other key material with the output. This was first done in the DESX variant developed by RSA Data Security, Inc., and then (presumably independently) in Khufu and Khafre. (Rivest named this technique; it's a nonstandard usage of the word.)

The idea is to prevent a cryptanalyst from obtaining a plaintext/ciphertext pair for the underlying algorithm. The technique forces a cryptanalyst to guess not only the algorithm key, but also one of the whitening values. Since there is an XOR both before and after the block algorithm, this technique is not susceptible to a meet-in-the-middle attack.

$$C = K_3 \oplus E_{K_2}(P \oplus K_1)$$
$$P = K_1 \oplus D_{K_2}(C \oplus K_3)$$

If $K_1 = K_3$, then a brute-force attack requires $2^{n + m/p}$ operations, where n is the key size, m is the block size, and p is the number of known plaintexts. If K_1 and K_3 are different, then a brute-force attack requires $2^{n + m + 1}$ operations with three known plaintexts. Against differential and linear cryptanalysis, these measures only provide a few key bits of protection. But computationally this is a very cheap way to increase the security of a block algorithm.

15.7 Cascading Multiple Block Algorithms

What about encrypting a message once with Algorithm A and key K_A, then again with Algorithm B and key K_B? Maybe Alice and Bob have different ideas about which algorithms are secure: Alice wants to use Algorithm A and Bob wants to use Algorithm B. This technique is sometimes called **cascading**, and can be extended far beyond only two algorithms and keys.

Pessimists have said that there is no guarantee that the two algorithms will work together to increase security. There may be subtle interactions between the two algorithms that actually *decrease* security. Even triple encryption with three different algorithms may not be as secure as you think. Cryptography is a black art; if you don't know what you are doing, you can easily get into trouble.

Reality is much rosier. The previous warnings are true only if the different keys are related to each other. If all of the multiple keys are independent, then the resultant cascade is at least as difficult to break as the first algorithm in the cascade [1033]. If the second algorithm is vulnerable to a chosen-plaintext attack, then the first algorithm might facilitate that attack and make the second algorithm vulnerable to a known-plaintext attack when used in a cascade. This potential attack is not limited to encryption algorithms: If you let someone else specify any algorithm which is used on your message before encryption, then you had better be sure that your encryption will withstand a chosen-plaintext attack. (Note that the most common algorithm used for compressing and digitizing speech to modem speeds, used before any encryption, is CELP—designed by the NSA.)

This can be better phrased: Using a chosen-plaintext attack, a cascade of ciphers is at least as hard to break as any of its component ciphers [858]. A previous result showed that the cascade is at least as difficult to break as the strongest algorithm, but that result is based on some unstated assumptions [528]. Only if the algorithms commute, as they do in the case of cascaded stream ciphers (or block ciphers in OFB mode), is the cascade at least as strong as the strongest algorithm.

If Alice and Bob do not trust each other's algorithms, they can use a cascade. If these are stream algorithms, the order doesn't matter. If they are block algorithms, Alice can first use Algorithm A and then use Algorithm B. Bob, who trusts Algorithm B more, can use Algorithm B followed by Algorithm A. They might even add a good stream cipher between the two algorithms; it can't hurt and could very well increase security.

Remember that the keys for each algorithm in the cascade must be independent. If Algorithm A has a 64-bit key and Algorithm B has a 128-bit key, then the resultant cascade must have a 192-bit key. If you don't use independent keys, then the pessimists are much more likely to be right.

15.8 COMBINING MULTIPLE BLOCK ALGORITHMS

Here's another way to combine multiple block algorithms, one that is guaranteed to be at least as secure as both algorithms. With two algorithms (and two independent keys):

(1) Generate a random-bit string, R, the same size as the message M.
(2) Encrypt R with the first algorithm.
(3) Encrypt $M \oplus R$ with the second algorithm.
(4) The ciphertext message is the results of steps (2) and (3).

Assuming the random-bit string is indeed random, this method encrypts M with a one-time pad and then encrypts both the pad and the encrypted message with each of the two algorithms. Since both are required to reconstruct M, a cryptanalyst must break both algorithms. The drawback is that the ciphertext is twice the size of the plaintext.

This method can be extended to multiple algorithms, but the ciphertext expands with each additional algorithm. It's a good idea, but I don't think it's very practical.