Chapter 6

Advanced Cryptanalysis

For there is nothing covered, that shall not be revealed;
neither hid, that shall not be known.
— Luke 12:2

The magic words are squeamish ossifrage
— Solution to RSA challenge problem
posed in 1977 by Ron Rivest, who
estimated that breaking the message
would require 40 quadrillion years.
It was broken in 1994.

6.1 Introduction

Perhaps the best ways to gain a strong understanding of cryptography is by trying to break ciphers. As an added bonus, breaking ciphers puts us in the role of our all-purpose attacker, Trudy, and we need to think like Trudy if we are going to make our systems more secure.

In previous chapters, we've seen a few simple cryptanalytic attacks. In this chapter, we kick it up a few notches and examine some relatively involved attacks. Specifically, we'll discuss the following cryptanalytic attacks.

- An attack on the most famous World War II cipher, the Enigma
- The attack on RC4, as used in WEP
- Linear and differential cryptanalysis of a block cipher
- The lattice reduction attack on the knapsack
- A timing attack on RSA

In World War II, the Nazis believed the Enigma cipher was invincible. Polish and British cryptanalysts proved otherwise. The idea behind the attack we describe was used to break Enigma messages, and yielded invaluable intelligence during the war. The attack illustrates some of the shortcomings of pre-modern ciphers.

Next, we consider an attack on RC4. This attack is specific to the way that RC4 is used in WEP. In this case, a relatively straightforward attack exists, in spite of the fact that RC4 is considered a strong cipher. While this might seem contradictory, the problem arises from the precise details of the way that RC4 is used in WEP. This example shows that a strong cipher can be broken if it is used improperly.

Linear and differential cryptanalysis are generally not practical means of attacking ciphers directly. Instead, they are used to analyze block ciphers for design weaknesses and, as a result, modern block ciphers are built with these techniques in mind. Therefore, to understand the design principles employed in block ciphers today, it is necessary to have some understanding of linear and differential cryptanalysis.

In Chapter 4, we mentioned the attack on the knapsack public key cryptosystem. In this chapter, we'll give more details on the attack. We do not present all of the mathematical nuances, but we provide sufficient information to understand the concept behind the attack and to write a program to implement the attack. It is a relatively straightforward attack that nicely illustrates the role that mathematics and algorithms can play in breaking cryptosystems.

A side channel is an unintended source of information. Recently, it has been shown that power usage or precise timings can often reveal information about an underlying computation. Timing attacks are particularly relevant for public key systems, since the computations involved are costly, and therefore take a relatively long time. Small differences in timings can reveal information about the private key.

Side channel attacks have been used successfully against several public key systems, and we'll discuss a couple of timing attacks on RSA. These attacks are representative of some of the most interesting and surprising cryptanalytic techniques developed in the recent past.

The attacks covered in this chapter represent only a small sample of the many interesting cryptanalytic techniques that are known. For more examples, of "applied" cryptanalysis, that is, attacks that break real ciphers and produce plaintext, see the book by Stamp and Low [285]. In fact, this chapter can be viewed as a warmup exercise for [285]. In contrast, Swenson's book [296] is an excellent source for details on modern block cipher cryptanalysis, where "attacks" mostly serve the role of helping cryptographers build better ciphers, rather than breaking ciphers in the sense of producing plaintext.

6.2 Enigma

I cannot forecast to you the action of Russia.
It is a riddle wrapped in a mystery inside an enigma:
but perhaps there is a key.
— Winston Churchill

The Enigma cipher was used by Nazi Germany prior to and throughout World War II. The forerunner of the military Enigma machine was developed by Arthur Scherbius as a commercial device. The Enigma was patented in the 1920s but it continued to evolve over time and the German military versions were significantly different than the original design. In reality, "Enigma" represents a family of cipher machines, but "the Enigma" invariably refers to the specific German military cipher machine that we discuss here.[1]

It is estimated that approximately 100,000 Enigma machines were constructed, about 40,000 of those during World War II. The version of Enigma that we describe here was used by the German Army throughout World War II [104]. The device was used to send tactical battlefield messages and for high-level strategic communications.

The Enigma was broken by the Allies, and the intelligence it provided was invaluable—as evidence by its cover name, ULTRA. The Germans had an unwavering belief that the Enigma was unbreakable, and they continued to use it for vital communications long after there were clear indications that it had been compromised. Of course, it's impossible to precisely quantify the effect of Enigma decrypts on the outcome of the war, but it is not farfetched to suggest that the intelligence provided by Enigma decrypts may have shortened the war in Europe by a year, saving hundreds of thousands of lives [309].

6.2.1 Enigma Cipher Machine

A picture of an Enigma cipher machine appears in Figure 2.5 in Chapter 2. Note the keyboard—essentially, a mechanical typewriter—and the "lightboard" of letters. Analogous to an old-fashioned telephone switchboard, the front panel has cables that connect pairs of letters. This switchboard (or plugboard) is known by its German name, *stecker*. There are also three *rotors* visible near the top of the machine.

Before encrypting a message, the operator had to initialize the device. The initial settings include various rotor settings and the stecker cable pluggings. These initial settings constitute the key.

[1]In fact, several variants of "the Enigma" were used by the German military and government. For example, the Army version used three rotors while the Naval version had four rotors.

Once the machine had been initialized, the message was typed on the keyboard and as each plaintext letter was typed, the corresponding ciphertext letter was illuminated on the lightboard. The ciphertext letters were written down as they appeared on the lightboard and subsequently transmitted, usually by voice over radio.

To decrypt, the recipient's Enigma had to be initialize in exactly the same way as the sender's. Then when the ciphertext was typed into the keyboard, the corresponding plaintext letters would appear on the lightboard.

The cryptographically significant components of the Enigma are illustrated in Figure 6.1. These components and the ways that they interact are described below.

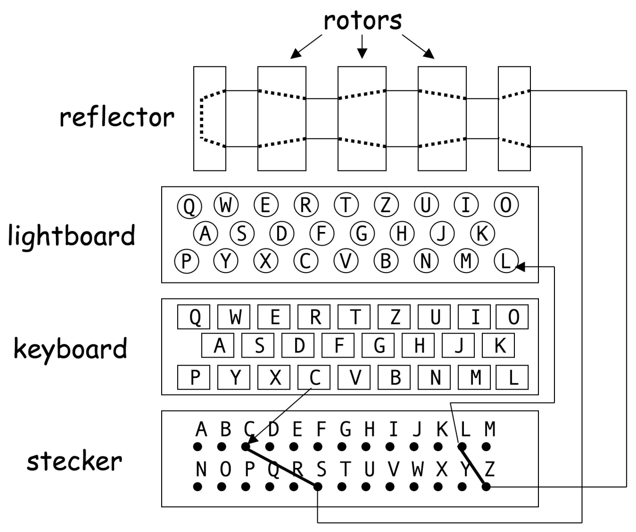

Figure 6.1: Enigma Diagram

To encrypt, a plaintext letter is entered on the keyboard. This letter first passes through the stecker, then, in turn, through each of the three rotors, through the reflector, back through each of the three rotors, back through the stecker, and finally, the resulting ciphertext letter is illuminated on the lightboard. Each rotor—as well as the reflector—consists of a hard-wired permutation of the 26 letters. Rotors as cryptographic elements are discussed in detail below in Section 6.2.3.

In the example illustrated in Figure 6.1, the plaintext letter `C` is typed on the keyboard, which is mapped to `S` due to the stecker cable connecting `C` to `S`. The letter `S` then passes through the rotors, the reflector, and back through the rotors. The net effect of all the rotors and the reflector is a permutation of the alphabet. In the example in Figure 6.1, `S` has been permuted to `Z`, which then becomes `L` due to the stecker cable between `L` and `Z`. Finally, the letter `L` is illuminated on the lightboard.

We use the following notation for the various permutations in the Enigma:

$$\begin{aligned} R_r &= \text{rightmost rotor} \\ R_m &= \text{middle rotor} \\ R_\ell &= \text{leftmost rotor} \\ T &= \text{reflector} \\ S &= \text{stecker} \end{aligned}$$

With this notation, from Figure 6.1 we see that

$$\begin{aligned} y &= S^{-1}R_r^{-1}R_m^{-1}R_\ell^{-1}TR_\ell R_m R_r S(x) \\ &= (R_\ell R_m R_r S)^{-1}T(R_\ell R_m R_r)S(x), \end{aligned} \tag{6.1}$$

where x is a plaintext letter, and y is the corresponding ciphertext letter.

If that's all there were to the Enigma, it would be nothing more than a glorified simple substitution cipher, with the initial settings determining the permutation. However, each time a keyboard letter is typed, the rightmost rotor steps one position, and the other rotors step in an odometer-like fashion—almost [48, 138].[2] That is, the middle rotor steps once for each 26 steps of the right rotor and the left rotor steps once for each 26 steps of the middle rotor. The reflector can be viewed as a fixed rotor since it permutes the letters, but it doesn't rotate. The overall effect is that the permutation changes with each letter typed. Note that, due to the odometer effect, the permutations R_r, R_m, and R_ℓ vary, but T and S do not.

Figure 6.2 illustrates the stepping of a single Engima rotor. This example shows the direction that the rotors step. From the operator's perspective, the letters appear in alphabetical order.

The Enigma is a substitution cipher where each letter is encrypted based on a permutation of the alphabet. But the Enigma is far from simple since, whenever a letter is encrypted (or decrypted), the odometer effect causes the permutation to change. Such a cipher is known as a poly-alphabetic substitution cipher. For the Enigma, the number of possible "alphabets" (i.e., permutations) is enormous.

[2]The "almost" is due to the mechanical system used to step the rotors, which causes the middle rotor to occasionally step twice in succession. Whenever a rotor steps, it causes the rotor to its right to also step. Suppose that the middle rotor just stepped to the position that engages the ratchet mechanism that will cause the leftmost rotor to step when the next letter is typed. Then when the next letter is typed, the left rotor will step, and this will also cause the middle rotor to step again. The middle rotor thereby steps twice in succession, violating the odometer effect. Note that this same ratcheting mechanism causes the right rotor to step whenever the middle rotor steps, but since the right rotor already steps for each letter typed, there is no noticeable effect on the right rotor.

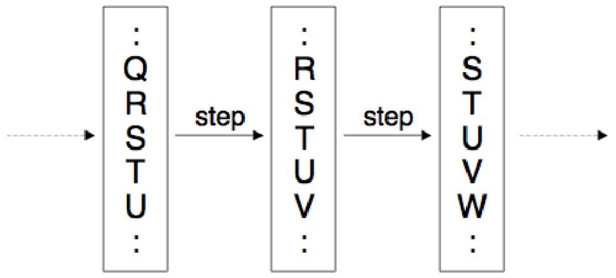

Figure 6.2: Enigma Rotor

6.2.2 Enigma Keyspace

The cryptographically significant components of the Enigma cipher are the stecker, the three rotors, and the reflector. The Enigma key consists of the initial settings for these components when the cipher is used to encrypt or decrypt a particular message. The variable settings that comprise the key are:

1. The choice of rotors.
2. The position of a movable ring on each of the two rightmost rotors. This ring allows the outer part of the rotor (labeled with the 26 letters) to rotate with respect to the inner part of the ring (where the actual permutation is wired).[3] Rotating this ring shifts the point at which the odometer effect occurs relative to the letters on the rotors.
3. The initial position of each rotor.
4. The number and plugging of the wires in the stecker.
5. The choice of reflector.

As mentioned above, each rotor implements a permutation of the 26 letters of the alphabet. The movable rings can be set to any of the 26 positions corresponding to the letters.

Each rotor is initially set to one of the 26 positions on the rotor, which are labeled with `A` through `Z`. The stecker is similar to an old-fashioned telephone switchboard, with 26 holes, each labeled with a letter of the alphabet. The stecker can have from 0 to 13 cables, where each cable connects a pair of letters. The reflector implements a permutation of the 26 letters, with the restriction that no letter can be permuted to itself, since this would cause a short circuit. Consequently, the reflector is equivalent to a stecker with 13 cables.

[3]This is analogous to rotating the position of a car tire relative to the rim.

Since there are three rotors, each containing a permutation of the 26 letters, there are

$$26! \cdot 26! \cdot 26! \approx 2^{265}$$

ways to select and place rotors in the machine. In addition, the number of ways to set the two movable rings—which determine when the odometer-like effects occurs—is $26 \cdot 26 \approx 2^{9.4}$.

The initial position of each of these rotors can be set to any one of 26 positions, so there are $26 \cdot 26 \cdot 26 = 2^{14.1}$ ways to initialize the rotors. However, this number should not be included in our count, since the different initial positions are all equivalent to some other rotor in some standard position. That is, if we assume that each rotor is initially set to, say, `A`, then setting a particular rotor to, say, `B`, is equivalent to some other rotor initially set to `A`. Consequently, the factor of 2^{265} obtained in the previous paragraph includes all possible rotors in all possible initial positions.

Finally, we must consider the stecker. Let $F(p)$ be the number of ways to plug p cables in the stecker. From Problem 2, we have

$$F(p) = \binom{26}{2p}(2p-1)(2p-3)\cdot \cdots \cdot 1.$$

The values of $F(p)$ are tabulated in Table 6.1.

Table 6.1: Stecker Combinations

$F(0) = 2^0$	$F(1) \approx 2^{8.3}$
$F(2) \approx 2^{15.5}$	$F(3) \approx 2^{21.7}$
$F(4) \approx 2^{27.3}$	$F(5) \approx 2^{32.2}$
$F(6) \approx 2^{36.5}$	$F(7) \approx 2^{40.2}$
$F(8) \approx 2^{43.3}$	$F(9) \approx 2^{45.6}$
$F(10) \approx 2^{47.1}$	$F(11) \approx 2^{47.5}$
$F(12) \approx 2^{46.5}$	$F(13) \approx 2^{42.8}$

Summing the entries in Table 6.1, we find that there are more than $2^{48.9}$ possible stecker configurations. Note that maximum occurs with 11 cables and that $F(10) \approx 2^{47.1}$. As mentioned above, the Enigma reflector is equivalent to a stecker with 13 cables. Consequently, there are $F(13) \approx 2^{42.8}$ different reflectors.

Combining all of these results, we find that, in principle, the size of the Enigma keyspace is about

$$2^{265} \cdot 2^{9.4} \cdot 2^{48.9} \cdot 2^{42.8} \approx 2^{366}.$$

That is, the theoretical keyspace of the Enigma is equivalent to a 366 bit key. Since modern ciphers seldom employ more than a 256 bit key, this gives some indication as to why the Germans had such great—but ultimately misplaced—confidence in the Enigma.

However, this astronomical number of keys is misleading. From Problem 1, we see that under the practical limitations of actual use by the German military, only about 2^{77} Enigma keys were available. This is still an enormous number and an exhaustive key search would have been out of the question using 1940s technology. Fortunately for the civilized world, shortcut attacks exist. But before we discuss an attack, we first take a brief detour to consider rotors as cryptographic elements.

6.2.3 Rotors

Rotors were used in many cipher machines during the first half of the 20th century—the Enigma is the most famous, but there were many others. Another interesting example of a rotor cipher machine is the American World War II-era machine Sigaba. The Sigaba cipher is a fascinating design that proved to be much stronger than Enigma. For a detailed cryptanalysis of Sigaba, see [281] or for a slightly abbreviated version see [285].

From a crypto-engineering standpoint, the appeal of a rotor is that it is possible to generate a large number of distinct permutations in a robust manner from a simple electro-mechanical device. Such considerations were important in the pre-computer era. In fact, the Enigma was an extremely durable piece of hardware, which was widely used in battlefield situations.

Hardware rotors are easy to understand, but it is slightly awkward to specify the permutations that correspond to the various positions of the rotor. A good analysis of these issues can be found in [185]. Here, we briefly discuss some of the main issues.

For simplicity, consider a rotor with four letters, `A` through `D`. Assuming the signal travels from left to right, the rotor illustrated in Figure 6.3 permutes `ABCD` to `CDBA`, that is, `A` is permuted to `C`, `B` is permuted to `D`, `C` is permuted to `B`, and `D` is permuted to `A`. The inverse permutation, `DCAB` in our notation, can be obtained by simply passing a signal through the rotors from right-to-left instead of left-to-right. This is a useful feature, since we can decrypt with the same hardware used to encrypt. The Enigma takes this one step further.[4] That is, the Enigma machine is its own inverse, which implies that the same machine with exactly the same settings can be used to encrypt and decrypt (see Problem 5).

Suppose that the rotor in Figure 6.3 steps once. Note that only the rotor itself—represented by the rectangle—rotates, not the electrical contacts at the edge of the rotor. In this example, we assume that the rotor steps "up," that is, the contact that was at `B` is now at `A` and so on, with the contact

[4]No pun intended (for a change...).

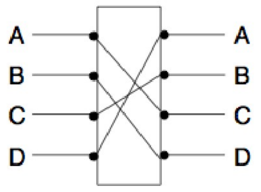

Figure 6.3: Rotor

that was at A wrapping around to D. The shift of the rotor in Figure 6.3 is illustrated in Figure 6.4. The resulting shifted permutation is CADB, which is, perhaps, not so obvious considering that the original permutation was CDBA.

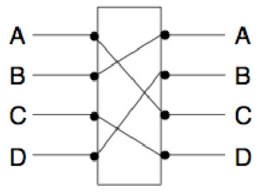

Figure 6.4: Stepped Rotor

In general, it is not difficult to determine the rotor shift of a permutation. The crucial point is that it's the offsets, or displacements, that shift. For example, in the permutation CDBA, the offsets are as follows: The letter A is permuted to C, which is an offset of 2 positions, the letter B is permuted to D, which is an offset of 2, the letter C is permuted to B, which is an offset of 3 (around the rotor), and D is permuted to A, which is an offset of 1. That is, the sequence of offsets for the permutation CDBA is $(2, 2, 3, 1)$. Cyclically shifting this sequence yields $(2, 3, 1, 2)$, which corresponds to the permutation CADB, and this is indeed the rotor shift that appears in Figure 6.4.

Again, physical rotors are actually very simple devices, but they are somewhat awkward to deal with in the abstract. For some additional exercise working with rotors, see Problem 12.

As mentioned above, one of the primary advantages of rotors is that they provide a simple electro-mechanical means to generate a large number of different permutations. Combining multiple rotors in series increases the number of permutations exponentially. For example, in Figure 6.5, C is permuted to A, while a shift of rotor L, denoted by $\sigma(L)$ and illustrated in Figure 6.6, causes C to be permuted to B. That is, stepping any single rotor changes the overall permutation.

With this three-rotor scheme, we can generate a cycle of 64 permutations of the letters ABCD by simply stepping through the 64 settings for the three

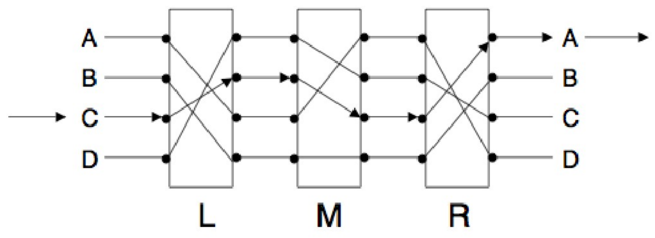

Figure 6.5: Three Rotors

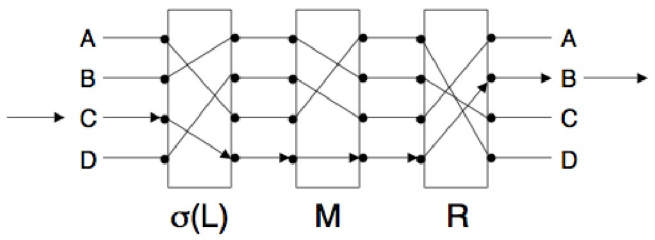

Figure 6.6: Rotor L Steps

rotors. Of course, not all of these permutations will be unique, since there are only 24 distinct permutations of the four letters `ABCD`. Also, by selecting different initial settings for the rotors, we can generate a different sequence of permutations. Furthermore, by selecting a different set of rotors (or reordering the given rotors), we can generate different sequences of permutations. As with a single rotor, it's easy to obtain the inverse permutations from a series of rotors by simply passing the signal through the rotors in the opposite direction. The inverse permutations are needed for decryption.

6.2.4 Enigma Attack

Polish cryptanalysts led by Marian Rejewski, Henryk Zygalski, and Jerzy Różycki were the first to successfully attack the Enigma [306]. Their challenge was greatly complicated by the fact that they did not know which rotors were in use. Through some clever mathematics, and a small but crucial piece of espionage [4], they were able to recover the rotor permutations from ciphertext. This certainly ranks as one of the greatest cryptanalytic successes of the era.

When Poland fell to the Nazis in 1939, Rejewski, Zygalski, and Różycki fled to France. After France fell under the Nazi onslaught, the Poles continued their cryptanalytic work from unoccupied Vichy France. The brilliant cryptanalytic work of Rejewski's team eventually made its way to Britain, where the British were rightly amazed. A group of British cryptanalysts that included Gordon Welchman and computing pioneer Alan Turing took up the Enigma challenge.

The Enigma attack that we describe here is similar to one developed by Turing, but somewhat simplified. This attack requires known plaintext, which in World War II terminology was known as a *crib*.

The essential idea is that, initially, we can ignore the stecker and make a guess for the remainder of the key. From Problem 1, there are less than 2^{30} such guesses. For each of these, we use information derived from a crib (known plaintext) to eliminate incorrect guesses. This attack, which has a work factor on the order 2^{30}, could be easily implemented on a modern computer, but it would have been impractical using World War II technology.

Suppose that we have the plaintext and corresponding ciphertext that appears in Table 6.2. We make use of this data in the attack described below.

Table 6.2: Enigma Known Plaintext Example

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Plaintext	O	B	E	R	K	O	M	M	A	N	D	O	D	E	R	W	E	H	R	M	A	C	H	T
Ciphertext	Z	M	G	E	R	F	E	W	M	L	K	M	T	A	W	X	T	S	W	V	U	I	N	Z

Let $S(x)$ be the result of the letter x passing through the stecker from the keyboard. Then $S^{-1}(x)$ is the result of x passing through the stecker in the other direction. For a given initial setting, let P_i be the permutation at step i, that is, P_i is the permutation determined by the composition of the three rotors, followed by the reflector, followed by the three rotors—in the opposite direction—at step i. Then, using the notation in equation (6.1), the overall permutation is given by

$$P_i = S^{-1}R_r^{-1}R_m^{-1}R_\ell^{-1}TR_\ell R_m R_r S,$$

where, to simplify the notation, we ignore the dependence of R_ℓ, R_m, and R_r on the step i.

Note that since P_i is a permutation, its inverse, P_i^{-1}, exists. Also, as noted above, due to the rotation of the rotors, the permutation varies with each letter typed. Consequently, P_i does indeed depend on i.

The Enigma attack we present here exploits "cycles" that occur in the known plaintext and corresponding ciphertext. Consider, for example, the column labeled 8 in Table 6.2. The plaintext letter A passes through the stecker, then through P_8 and, finally, through S^{-1} to yield the ciphertext M, that is, $S^{-1}P_8S(\texttt{A}) = \texttt{M}$, which we can rewrite as $P_8S(\texttt{A}) = S(\texttt{M})$.

From the known plaintext in Table 6.2, we have

$$P_8 S(\mathtt{A}) = S(\mathtt{M})$$
$$P_6 S(\mathtt{M}) = S(\mathtt{E})$$
$$P_{13} S(\mathtt{E}) = S(\mathtt{A}).$$

These three equations can be combined to yield the cycle

$$S(\mathtt{E}) = P_6 P_8 P_{13} S(\mathtt{E}). \tag{6.2}$$

Now suppose that we select one of the possible initial settings for the machine, ignoring the stecker. Then all P_i and P_i^{-1} that correspond to this setting are known. Next, suppose that we guess, say, $S(\mathtt{E}) = \mathtt{G}$, that is, we guess that `E` and `G` are connected by a cable in the stecker plugboard. If it's actually true that the stecker has a wire connecting `E` and `G`, and if our guess for the initial settings of the machine is correct, then from equation (6.2) we must have

$$\mathtt{G} = P_6 P_8 P_{13}(\mathtt{G}). \tag{6.3}$$

If we try all 26 choices for $S(\mathtt{E})$ and equation (6.2) is never satisfied, then we know that our guess for the rotor settings is incorrect and we can eliminate this choice. We would like to use this observation to reduce the number of rotor settings, ideally, to just one. However, if we find any guess for $S(\mathtt{E})$ for which equation (6.2) holds, then we cannot rule out the current rotor settings. Unfortunately, there are 26 possible guesses for $S(\mathtt{E})$ and, for each, there is a 1/26 chance that equation (6.2) holds at random. Consequently, we obtain no reduction in the number of possible keys when using just one cycle.

Fortunately, all is not lost. If we can find an additional cycle involving $S(\mathtt{E})$, then we can use this in combination with equation (6.2) to reduce the number of possible rotor settings. We're in luck, since we can combine the four equations,

$$S(\mathtt{E}) = P_3 S(\mathtt{R})$$
$$S(\mathtt{W}) = P_{14} S(\mathtt{R})$$
$$S(\mathtt{W}) = P_7 S(\mathtt{M})$$
$$S(\mathtt{E}) = P_6 S(\mathtt{M})$$

to obtain

$$S(\mathtt{E}) = P_3 P_{14}^{-1} P_7 P_6^{-1} S(\mathtt{E}).$$

Now if we guess, say, $S(\mathtt{E}) = \mathtt{G}$, we have two equations that must hold if this guess is correct. There are still 26 choices for $S(\mathtt{E})$, but with two cycles, there is only a $(1/26)^2$ chance that they both hold at random. Therefore, with two cycles in $S(\mathtt{E})$, we can reduce the number of viable machine settings (that is, keys) by a factor of 26. We can easily develop an attack based on these observations.

To reiterate, the crucial observation here is that, once we specify the rotor settings, all permutations $P_0, P_1, P_2, \ldots$ and $P_0^{-1}, P_1^{-1}, P_2^{-1}, \ldots$ are known. Then if we substitute a putative value for $S(\mathtt{E})$, we can immediately check the validity of all cycle equations that are available. For an incorrect guess of $S(\mathtt{E})$ (or incorrect rotor settings) there is a $1/26$ chance any given cycle will hold true. But with n cycles, there is only a $(1/26)^n$ chance that all cycle equations will hold true. Consequently, with n cycles involving $S(\mathtt{E})$, we can reduce the number of possible initial rotor settings by a factor of 26^{n-1}. Since there are only about 2^{30} rotor settings, with enough cycles, we can reduce the number of possible rotor settings to one, which is the key.

Amazingly, by recovering the initial rotor settings in this manner, stecker values are also recovered—essentially for free. However, any stecker values that do not contribute to a cycle will remain unknown, but once the rotor settings have been determined, the remaining unknown stecker settings are easy to determine (see Problem 7). It is interesting to note that, in spite of an enormous number of possible settings, the stecker contributes virtually nothing to the security of the Enigma.

It is important to realize that the attack described here would have been impractical using 1940s technology. The practical attacks of World War II required that the cryptanalyst reduce the number of cases to be tested to a much smaller number than 2^{30}. Many clever techniques were developed to squeeze as much information as possible from ciphertext. In addition, much effort was expended finding suitable cribs (i.e., known plaintext) since all of the practical attacks required known plaintext.

6.3 RC4 as Used in WEP

Suddenly she came upon a little three-legged table, all made of solid glass:
there was nothing on it but a tiny golden key...
— *Alice in Wonderland*

RC4 is described in Section 3.2.2 of Chapter 3 and WEP is described in Section 10.6 of Chapter 10. Here, we provide a detailed description of the cryptanalytic attack that is mentioned in Section 10.6. Note that the RC4 algorithm is considered secure when used properly. However, WEP, which is widely viewed as the "Swiss cheese" of security protocols, somehow managed to implement nearly all of its security functions insecurely, including RC4. As a result, there is a feasible attack on RC4 encryption as used in WEP. Before studying this attack, you might want to preview Section 10.6.

WEP encrypts data with the stream cipher RC4 using a long-term key that seldom (if ever) changes. To avoid repeated keystreams, an initialization vector, or IV, is sent in the clear with each message, where each packet is treated as a new message. The IV is mixed with the long-term key to produce the message key. The upshot is that the cryptanalyst, Trudy, gets to see the IVs, and any time an IV repeats, Trudy knows that the same keystream is being used to encrypt the data. Since the IV is only 24 bits, repeated IVs occur relatively often. A repeated IV implies a repeated keystream, and a repeated keystream is bad—at least as bad as reuse of a one-time pad. That is, a repeated keystream provides statistical information to the attacker who could then conceivably liberate the keystream from the ciphertext. Once the keystream for a packet is known, it can be used to decrypt any packet that uses the same IV.

However, in WEP, there are several possible shortcuts that make an attacker's life easier, as discussed in Section 10.6. Here, we discuss a cryptanalytic attack on the RC4 stream cipher as it is used in WEP. Again, this attack is only possible due to the specific way that WEP uses RC4—specifically, the way that it creates the session key from an initialization vector IV and the long-term key.[5]

This cryptanalytic attack has a small work factor, and it will succeed provided that a sufficient number of IVs are observed. This clever attack, which can be considered a type of *related key* attack, is due to Fluhrer, Mantin, and Shamir [112].

6.3.1 RC4 Algorithm

RC4 is simplicity itself. At any given time, the state of the cipher consists of a lookup table S containing a permutation of all byte values, $0, 1, 2, \ldots, 255$, along with two indices i and j. When the cipher is initialized, the permutation is scrambled using a key, denoted key[i], for $i = 0, 1, \ldots, N-1$, which can be of any length from 0 to 256 bytes. In the initialization routine, the lookup table S is modified (based on the key) in such a way that S always contains a permutation of the byte values. The RC4 initialization algorithm appears in Table 6.3.

The RC4 keystream is generated one byte at a time. An index is determined based on the current contents of S, and the indexed byte is selected as the keystream byte. Similar to the initialization routine, at each step the permutation S is modified so that S always contains a permutation of $\{0, 1, 2, \ldots, 255\}$. The keystream generation algorithm appears in Table 6.4. For more details on the RC4 algorithm, see Section 3.2.2.

[5]The attack does highlight a shortcoming in the RC4 initialization process—a shortcoming that can be fixed without modifying the underlying RC4 algorithm.

Table 6.3: RC4 Initialization

```
for $i = 0$ to 255
    $S_i = i$
    $K_i = \text{key}[i \pmod N]$
next $i$
$j = 0$
for $i = 0$ to 255
    $j = (j + S_i + K_i) \pmod{256}$
    swap$(S_i, S_j)$
next $i$
$i = j = 0$
```

Table 6.4: RC4 Keystream Generator

```
$i = (i + 1) \pmod{256}$
$j = (j + S_i) \pmod{256}$
swap$(S_i, S_j)$
$t = (S_i + S_j) \pmod{256}$
keystreamByte $= S_t$
```

6.3.2 RC4 Cryptanalytic Attack

In 2000, Fluhrer, Mantin, and Shamir [112] published a practical attack on RC4 encryption as it is used in WEP. In WEP, a non-secret 24-bit initialization vector, denoted as IV, is prepended to a long-term key and the result is used as the RC4 key. Note that the role of the IV in WEP encryption is somewhat similar to the role that an IV plays in various block cipher encryption modes (see Section 3.3.7 of Chapter 3). The WEP IV is necessary to prevent messages from being sent in depth. Recall that two ciphertext messages are in depth if they were encrypted using the same key. Messages in depth are a serious threat to a stream cipher.

We assume that Trudy, the cryptanalyst, knows many WEP ciphertext messages (packets) and their corresponding IVs. Trudy would like to recover the long-term key. The Fluhrer-Mantin-Shamir attack provides a clever, efficient, and elegant way to do just that. This attack has been successfully used to break real WEP traffic [295].

Suppose that for a particular message, the three-byte initialization vector is of the form

$$\text{IV} = (3, 255, V), \tag{6.4}$$

where V can be any byte value. Then these three IV bytes become K_0, K_1, and K_2 in the RC4 initialization algorithm of Table 6.3, while K_3 is the first byte of the unknown long-term key. That is, the message key is

$$K = (3, 255, V, K_3, K_4, \ldots), \tag{6.5}$$

where V is known to Trudy, but $K_3, K_4, K_5, \ldots$ are unknown. To understand the attack, we need to carefully consider what happens to the table S during the RC4 initialization phase when K is of the form in equation (6.5).

In the RC4 initialization algorithm, which appears in Table 6.3, we first set S to the identity permutation, so that we have

i	0	1	2	3	4	5	$\ldots$
S_i	0	1	2	3	4	5	$\ldots$

Suppose that K is of the form in (6.5). Then at the $i = 0$ initialization step, we compute the index $j = 0 + S_0 + K_0 = 3$ and elements i and j are swapped, resulting in the table

i	0	1	2	3	4	5	$\ldots$
S_i	3	1	2	0	4	5	$\ldots$

At the next step, $i = 1$ and $j = 3 + S_1 + K_1 = 3 + 1 + 255 = 3$, since the addition is modulo 256. Elements i and j are again swapped, giving

i	0	1	2	3	4	5	$\ldots$
S_i	3	0	2	1	4	5	$\ldots$

At step $i = 2$ we have $j = 3 + S_2 + K_2 = 3 + 2 + V = 5 + V$ and after the swap,

i	0	1	2	3	4	5	$\ldots$	$5+V$	$\ldots$
S_i	3	0	$5+V$	1	4	5	$\ldots$	2	$\ldots$

At the next step, $i = 3$ and $j = 5 + V + S_3 + K_3 = 6 + V + K_3$, where K_3 is unknown. After swapping, the lookup table is

i	0	1	2	3	4	5	$\ldots$
S_i	3	0	$5+V$	$6+V+K_3$	4	5	$\ldots$

i	$\ldots$	$5+V$	$\ldots$	$6+V+K_3$	$\ldots$
S_i	$\ldots$	2	$\ldots$	1	$\ldots$

assuming that, after reduction modulo 256, we have $6 + V + K_3 > 5 + V$. If this is not the case, then $6 + V + K_3$ will appear to the left of $5 + V$, which has no effect on the success of the attack.

Now suppose for a moment that the RC4 initialization algorithm were to stop after the $i = 3$ step. Then, if we generate the first byte of the keystream according to the algorithm in Table 6.4, we find $i = 1$ and $j = S_i = S_1 = 0$, so that $t = S_1 + S_0 = 0 + 3 = 3$. Then the first keystream byte would be

$$\text{keystreamByte} = S_3 = (6 + V + K_3) \pmod{256}. \tag{6.6}$$

Assuming that Trudy knows (or can guess) the first byte of the plaintext, she can determine the first byte of the keystream. If this is the case, Trudy can simply solve equation (6.6) to obtain the first unknown key byte, since

$$K_3 = (\text{keystreamByte} - 6 - V) \pmod{256}. \tag{6.7}$$

Unfortunately (for Trudy), the initialization phase is 256 steps instead of just four. But notice that as long as S_0, S_1 and S_3 are not altered in any subsequent initialization step, then equation (6.7) will hold. What is the chance that these three elements remain unchanged? The only way that an element can change is if it is swapped for another element. From $i = 4$ to $i = 255$ of the initialization, the i index will not affect any of these elements since it steps regularly from 4 to 255. If we treat the j index as random, then at each step the probability that the three indices of concern are all unaffected is 253/256. The probability that this holds for all of the final 252 initialization steps is, therefore,

$$\left(\frac{253}{256}\right)^{252} \approx 0.0513.$$

Consequently, we expect equation (6.7) to hold slightly more than 5% of the time. Then with a sufficient number of IVs of the form in equation (6.4) Trudy can determine K_3 from equation (6.7), assuming she knows the first keystream byte in each case.

What is a sufficient number of IVs to recover K_3? If we observe n encrypted packets, each with an IV of the form in equation (6.4), then we expect to solve for the actual K_3 using equation (6.7) for about $0.05n$ of these. For the remaining $0.95n$ of the cases, we expect the result of the subtraction in equation (6.7) to be a random value in $\{0, 1, 2, \ldots, 255\}$. Then the expected number of times that any particular value other than K_3 appears is about $0.95n/256$, and the correct value will have an expected count of $0.05n + 0.95n/256 \approx 0.05n$. We need to choose n large enough so that we can, with high probability, distinguish K_3 from the random "noise." If we choose $n = 60$, then we expect to see K_3 three times, while it is unlikely that we will see any random value more than twice (see also Problem 13).

This attack is easily extended to recover the remaining unknown key bytes. We illustrate the next step—assuming that Trudy has recovered K_3, we show that she can recover the key byte K_4. In this case, Trudy will look for initialization vectors of the form

$$\text{IV} = (4, 255, V), \tag{6.8}$$

where V can be any value. Then, at the $i = 0$ step of the initialization, $j = 0 + S_0 + K_0 = 4$ and elements i and j are swapped, resulting in

i	0	1	2	3	4	5	...
S_i	4	1	2	3	0	5	...

At the next step, $i = 1$ and $j = 4 + S_1 + K_1 = 4$ (since the addition is mod 256) and elements S_1 and S_4 are swapped, giving

i	0	1	2	3	4	5	...
S_i	4	0	2	3	1	5	...

At step $i = 2$ we have $j = 4 + S_2 + K_2 = 6 + V$, and after the swap

i	0	1	2	3	4	5	...	$6+V$	...
S_i	4	0	$6+V$	3	1	5	...	2	...

At the next step, $i = 3$ and $j = 5 + V + S_3 + K_3 = 9 + V + K_3$, and K_3 is known. After swapping

i	0	1	2	3	4	5	...
S_i	4	0	$6+V$	$9+V+K_3$	1	5	...

i	...	$6+V$	...	$9+V+K_3$	...
S_i	...	2	...	3	...

assuming that $9 + V + K_3 > 6 + V$ when the sums are taken mod 256.

Carrying this one step further, we have $i = 4$ and

$$j = 9 + V + K_3 + S_4 + K_4 = 10 + V + K_3 + K_4,$$

where only K_4 is unknown. After swapping, the table S is of the form

i	0	1	2	3	4	5	...
S_i	4	0	$6+V$	$9+V+K_3$	$10+V+K_3+K_4$	5	...

i	...	$6+V$	...	$9+V+K_3$	...	$10+V+K_3+K_4$	...
S_i	...	2	...	3	...	1	...

If the initialization were to stop at this point (after the $i = 4$ step) then for first byte of the keystream we would find $i = 1$ and $j = S_i = S_1 = 0$, so that $t = S_1 + S_0 = 4 + 0 = 4$. The resulting keystream byte would be

$$\text{keystreamByte} = S_4 = (10 + V + K_3 + K_4) \pmod{256},$$

where the only unknown is K_4. As a result,

$$K_4 = (\text{keystreamByte} - 10 - V - K_3) \pmod{256}. \tag{6.9}$$

Of course, the initialization does not stop after the $i = 4$ step, but, as in the K_3 case, the chance that equation (6.9) holds is about 0.05. Consequently, with a sufficient number of IVs of the form in equation (6.8), Trudy can determine K_4. Continuing, any number of key bytes can be recovered, provided enough IVs of the correct form are available and Trudy knows the first keystream byte of each corresponding packet.

This same technique can be extended to recover additional key bytes, $K_5, K_6, \ldots$. In fact, if a sufficient number of packets are available, a key of any length can be recovered with a trivial amount of work. This is one reason why WEP is said to be "unsafe at any key size" [322].

Consider once again the attack to recover the first unknown key byte K_3. It is worth noting that some IVs that are not of the form $(3, 255, V)$ will be useful to Trudy. For example, suppose the IV is $(2, 253, 0)$. Then after the $i = 3$ initialization step, the array S is

i	0	1	2	3	4	$\ldots$	$3 + K_3$	$\ldots$
S_i	0	2	1	$3 + K_3$	4	$\ldots$	3	$\ldots$

If S_1, S_2, and S_3 are not altered in the remaining initialization steps, the first keystream byte will be $3 + K_3$, from which Trudy can recover K_3. Notice that for a given three-byte IV, Trudy can compute the initialization up through the $i = 3$ step and, by doing so, she can easily determine whether a given IV will be useful for her attack. Similar comments hold for subsequent key bytes. By using all of the useful IVs, Trudy can reduce the number of packets she must observe before recovering the key.

Finally, it is worth noting that it is also possible to recover the RC4 key if the IV is appended to the unknown key instead of being prepended (as in WEP); see [196] for the details.

6.3.3 Preventing Attacks on RC4

There are several possible ways to prevent attacks on RC4 that target its initialization phase. The standard suggestion is to, in effect, add 256 steps to the initialization process. That is, after the initialization in Table 6.3 has run its course, generate 256 keystream bytes according to the RC4 keystream generation algorithm in Table 6.4, discarding these bytes. After this process has completed, generate the keystream in the usual way. If the sender and receiver follow this procedure, the attack discussed in this section is not feasible. Note that no modification to the inner workings of RC4 is required.

Also, there are many alternative ways to combine the key and IV that would effectively prevent the attack described in this section; Problem 17 asks for such methods. As with so many other aspects of WEP, its designers managed to choose one of the most insecure possible approaches to using the RC4 cipher.

6.4 Linear and Differential Cryptanalysis

We sent the [DES] S-boxes off to Washington.
They came back and were all different.
— Alan Konheim, one of the designers of DES

I would say that, contrary to what some people believe, there is no evidence of tampering with the DES so that the basic design was weakened.
— Adi Shamir

As discussed in Section 3.3.2, the influence of the Data Encryption Standard (DES) on modern cryptography can't be overestimated. For one thing, both linear and differential cryptanalysis were developed to attack DES. As mentioned above, these techniques don't generally yield practical attacks. Instead, linear and differential "attacks" point to design weaknesses in block ciphers. These techniques have become basic analytic tools that are used to analyze all block ciphers today.

Differential cryptanalysis is, at least in the unclassified realm, due to Biham and Shamir (yes, that Shamir, yet again) who introduced the technique in 1990. Subsequently, it has become clear that someone involved in the design of DES (that is, someone at the National Security Agency) was aware of differential cryptanalysis prior to the mid 1970s. Note that differential cryptanalysis is a chosen plaintext attack, which makes it somewhat difficult to actually apply in the real world.

Linear cryptanalysis was apparently developed by Matsui in 1993. Since DES was not designed to offer optimal resistance to a sophisticated linear cryptanalysis attacks, either NSA did not know about the technique in the 1970s, or they were not concerned about such an attack on the DES cipher. Linear cryptanalysis is slightly more realistic as a real-world attack than differential cryptanalysis, primarily because it is a known plaintext attack instead of a chosen plaintext attack.

6.4.1 Quick Review of DES

We don't require all of the details of DES here, so we'll give a simplified overview that only includes the essential facts that we'll need below. DES has eight S-boxes, each of which maps six input bits, denoted $x_0x_1x_2x_3x_4x_5$, to four output bits, denoted $y_0y_1y_2y_3$. For example, DES S-box number one, in hexadecimal notation, appears in Table 6.5.

Figure 6.7 gives a much simplified view of DES, which is sufficient for our purposes. Below, we are mostly interested in analyzing the nonlinear parts of DES, so the diagram highlights the fact that the S-boxes are the only

Table 6.5: DES S-box Number One

	$x_1x_2x_3x_4$															
x_0x_5	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7
1	0	F	7	4	E	2	D	1	A	6	C	B	9	5	3	4
2	4	1	E	8	D	6	2	B	F	C	9	7	3	A	5	0
3	F	C	8	2	4	9	1	7	5	B	3	E	A	0	6	D

nonlinearity in DES. Figure 6.7 also illustrates the way that the subkey K_i enters into a DES round. This will also be important in the discussion to follow.

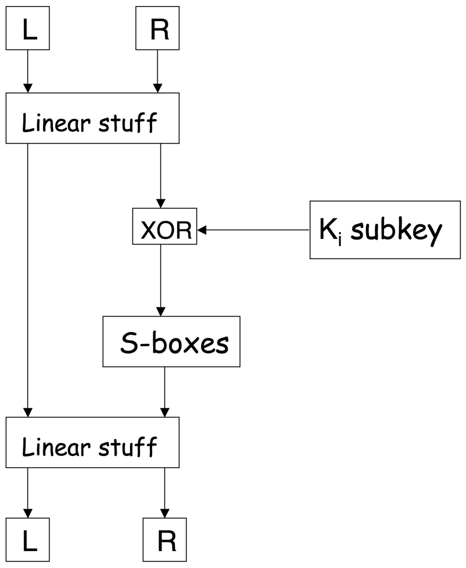

Figure 6.7: Simplified View of DES

Next, we'll present a quick overview of differential cryptanalysis followed by a similar overview of linear cryptanalysis. We'll then present a simplified version of DES, which we've called Tiny DES, or TDES. We'll present both linear and differential attacks on TDES.

6.4.2 Overview of Differential Cryptanalysis

Since differential cryptanalysis was developed to analyze DES, let's discuss it in the context of DES. Recall that all of DES is linear except for the

S-boxes. We'll see that the linear parts of DES play a significant role in its security, however, from a cryptanalytic point of view, the linear parts are easy. Mathematicians are good at solving linear equations, so it is the nonlinear parts that represent the major cryptanalytic hurdles. As a result, both differential and linear cryptanalysis are focused on dealing with the nonlinear parts of DES, namely, the S-boxes.

The idea behind a differential attack is to compare input and output differences. For simplicity, we'll first consider a simplified S-box. Suppose that a DES-like cipher uses the 3-bit to 2-bit S-box

$$\begin{array}{c|cccc} & \multicolumn{4}{c}{\text{column}} \\ \text{row} & 00 & 01 & 10 & 11 \\ \hline 0 & 10 & 01 & 11 & 00 \\ 1 & 00 & 10 & 01 & 11 \end{array} \tag{6.10}$$

where, for input bits $x_0x_1x_2$, the bit x_0 indexes the row, while x_1x_2 indexes the column. Then, for example, $\texttt{Sbox}(010) = 11$, since the bits in row 0 and column 10 are 11.

Consider the two inputs, $X_1 = 110$ and $X_2 = 010$, and suppose the key is $K = 011$. Then $X_1 \oplus K = 101$ and $X_2 \oplus K = 001$ and we have

$$\texttt{Sbox}(X_1 \oplus K) = 10 \text{ and } \texttt{Sbox}(X_2 \oplus K) = 01. \tag{6.11}$$

Now suppose that K in equation (6.11) is unknown, but the inputs, namely, $X_1 = 110$ and $X_2 = 010$, are known as well as the corresponding outputs $\texttt{Sbox}(X_1 \oplus K) = 10$ and $\texttt{Sbox}(X_2 \oplus K) = 01$. Then from the S-box in (6.10) we see that $X_1 \oplus K \in \{000, 101\}$ and $X_2 \oplus K \in \{001, 110\}$. Since X_1 and X_2 are known, we have that

$$K \in \{110, 011\} \cap \{011, 100\}$$

which implies that $K = 011$. This "attack" is essentially a known plaintext attack on the single S-box in (6.10) for the key K. The same approach will work on a single DES S-box.

However, attacking one S-box in one round of DES does not appear to be particularly useful. In addition, the attacker will not know the input to any round except for the first, and the attacker will not know the output of any round but the last. The intermediate rounds appear to be beyond the purview of the cryptanalyst.

For this approach to prove useful in analyzing DES, we must be able to extend the attack to one complete round, that is, we must take into account all eight S-boxes simultaneously. Once we have extended the attack to one round, we must then extend the attack to multiple rounds. On the surface, both of these appear to be daunting tasks.

However, we'll see that by focusing on input and output differences, it becomes easy to make some S-boxes "active" and others "inactive." As a result, we can, in some cases, extend the attack to a single round. To then extend the attack to multiple rounds, we must choose the input difference so that the output difference is in a useful form for the next round. This is challenging and depends on the specific properties of the S-boxes, as well as the linear mixing that occurs at each round.

The crucial point here is that we'll focus on input and output differences. Suppose we know inputs X_1 and X_2. Then for input X_1, the actual input to the S-box is $X_1 \oplus K$ and for input X_2 the actual input to S-box is $X_2 \oplus K$, where the key K is unknown. Differences are defined modulo 2, implying that the difference operation is the same as the sum operation, namely, XOR. Then the S-box input difference is

$$(X_1 \oplus K) \oplus (X_2 \oplus K) = X_1 \oplus X_2. \tag{6.12}$$

Note that the input difference is independent of the key K. This is the fundamental observation that enables differential cryptanalysis to work.

Let $Y_1 = \texttt{Sbox}(X_1 \oplus K)$ and let $Y_2 = \texttt{Sbox}(X_2 \oplus K)$. Then the output difference $Y_1 \oplus Y_2$ is almost the input difference to next round. The goal is to carefully construct the input difference, so that we can "chain" differences through multiple rounds. Since the input difference is independent of the key—and since differential cryptanalysis is a chosen plaintext attack—we have the freedom to choose the inputs so that the output difference has any particular form that we desire.

Another crucial element of a differential attack is that an S-box input difference of zero always results in an output difference of zero. Why is this the case? An input difference of zero simply means that the input values, say, X_1 and X_2, are the same, in which case the output values Y_1 and Y_2 must be the same, that is, $Y_1 \oplus Y_2 = 0$. The importance of this elementary observation is that we can make S-boxes "inactive" with respect to differential cryptanalysis by choosing their input differences to be zero.

A final observation is that it is not necessary that things happen with certainty. In other words, if an outcome only occurs with some nontrivial probability, then we may be able to develop a probabilistic attack that will still prove useful in recovering the key.

Given any S-box, we can analyze it for useful input differences as follows. For each possible input value X, find all pairs X_1 and X_2 such that

$$X = X_1 \oplus X_2$$

and compute the corresponding output differences

$$Y = Y_1 \oplus Y_2,$$

where

$$Y_1 = \texttt{Sbox}(X_1) \text{ and } Y_2 = \texttt{Sbox}(X_1).$$

By tabulating the resulting counts, we can find the most biased input values. For example for the S-box in (6.10), this analysis yields the results in Table 6.6.

Table 6.6: S-box Difference Analysis

	$\texttt{Sbox}(X_1) \oplus \texttt{Sbox}(X_2)$			
$X_1 \oplus X_2$	00	01	10	11
000	8	0	0	0
001	0	0	4	4
010	0	8	0	0
011	0	0	4	4
100	0	0	4	4
101	4	4	0	0
110	0	0	4	4
111	4	4	0	0

For any S-box, an input difference of 000 is not interesting—the input values are the same and the S-box is "inactive" (with respect to differences), since the output values must be the same. For the example in Table 6.6, an input difference of 010 always gives an output of 01, which is the most biased possible result. And, as noted in equation (6.12), by selecting, say, $X_1 \oplus X_2 = 010$, the actual input difference to the S-box would be 010 since the key K drops out of the difference.

Differential cryptanalysis of DES is fairly complex. To illustrate the technique more concretely, but without all of the complexity inherent in DES, we'll present a scaled-down version of DES that we call Tiny DES, or TDES. Then we'll perform differential and linear cryptanalysis on TDES. But first we present a quick overview of linear cryptanalysis.

6.4.3 Overview of Linear Cryptanalysis

Ironically, linear cryptanalysis—like differential cryptanalysis—is focused on the nonlinear part of a block cipher. Although linear cryptanalysis was developed a few years after differential cryptanalysis, it's conceptually simpler, it's more effective on DES, and it only requires known plaintext—as opposed to chosen plaintext.

In differential cryptanalysis, we focused on input and output differences. In linear cryptanalysis, the objective is to approximate the nonlinear part of a cipher with linear equations. Since mathematicians are good at solving linear equations, if we can find such approximations, it stands to reason that we can use these to attack the cipher. Since the only nonlinear part of DES is its S-boxes, linear cryptanalysis will be focused on the S-boxes.

Consider again the simple S-box in (6.10). We denote the three input bits as $x_0x_1x_2$ and the two output bits as y_0y_1. Then x_0 determines the row, and x_1x_2 determines the column. In Table 6.7, we've tabulated the number of values for which each possible linear approximation holds. Note that any table entry that is not 4 indicates a nonrandom output.

Table 6.7: S-box Linear Analysis

	output bits		
input bits	y_0	y_1	$y_0 \oplus y_1$
0	4	4	4
x_0	4	4	4
x_1	4	6	2
x_2	4	4	4
$x_0 \oplus x_1$	4	2	2
$x_0 \oplus x_2$	0	4	4
$x_1 \oplus x_2$	4	6	6
$x_0 \oplus x_1 \oplus x_2$	4	6	2

The results in Table 6.7 show that, for example, $y_0 = x_0 \oplus x_2 \oplus 1$ with probability 1 and $y_0 \oplus y_1 = x_1 \oplus x_2$ with probability 3/4. Using information such as this, in our analysis we can replace the S-boxes by linear functions. The result is that, in effect, we've traded the nonlinear S-boxes for linear equations, where the linear equations do not hold with certainty, but instead the equations hold with some nontrivial probability.

For these linear approximations to be useful in attacking a block cipher such as DES, we'll try to extend this approach so that we can solve linear equations for the key. As with differential cryptanalysis, we must somehow "chain" these results through multiple rounds.

How well can we approximate a DES S-box with linear functions? Each DES S-box was designed so that no linear combination of inputs is a good approximation to a single output bit. However, there are linear combinations of output bits that can be approximated by linear combinations of input bits. As a result, there is potential for success in the linear cryptanalysis of DES.

As with differential cryptanalysis, the linear cryptanalysis of DES is complex. To illustrate a linear attack, we'll next describe TDES, a scaled-down DES-like cipher. Then we'll perform differential and linear cryptanalysis on TDES.

6.4.4 Tiny DES

Tiny DES, or TDES, is a DES-like cipher that is simpler and easier to analyze than DES. TDES was designed by your contriving author to make linear and differential attacks easy to study—it is a contrived cipher that is trivial to break. Yet it's similar enough to DES to illustrate the principles.

TDES is a much simplified version of DES with the following numerology.

- A 16-bit block size
- A 16-bit key size
- Four rounds
- Two S-boxes, each mapping 6 bits to 4 bits
- A 12-bit subkey in each round

TDES has no P-box, initial or final permutation. Essentially, we have eliminated all features of DES that contribute nothing to its security, while at the same time scaling down the block and key sizes.

Note that the small key and block sizes imply that TDES cannot offer any real security, regardless of the underlying algorithm. Nevertheless, TDES will be a useful design for illustrating linear and differential attacks, as well as the larger issues of block cipher design.

TDES is a Feistel cipher and we denote the plaintext as (L_0, R_0). Then for $i = 1, 2, 3, 4$,

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus F(R_{i-1}, K_i) \end{aligned}$$

where the ciphertext is (L_4, R_4). A single round of TDES is illustrated in Figure 6.8, where the numbers of bits are indicated on each line. Next, we'll completely describe all of the pieces of the TDES algorithm.

TDES has two S-boxes, denoted $\texttt{SboxLeft}(X)$ and $\texttt{SboxRight}(X)$. Both S-boxes map 6 bits to 4 bits, as in standard DES. The parts of TDES that we'll be most interested in are the S-boxes and their input. To simplify the notation, we'll define the function

$$F(R, K) = \texttt{Sboxes}(\texttt{expand}(R) \oplus K), \tag{6.13}$$

where

$$\texttt{Sboxes}(x_0x_1x_2 \ldots x_{11}) = (\texttt{SboxLeft}(x_0x_1 \ldots x_5), \texttt{SboxRight}(x_6x_7 \ldots x_{11})).$$

The expansion permutation is given by

$$\texttt{expand}(R) = \texttt{expand}(r_0r_1 \ldots r_7) = (r_4r_7r_2r_1r_5r_7r_0r_2r_6r_5r_0r_3). \tag{6.14}$$

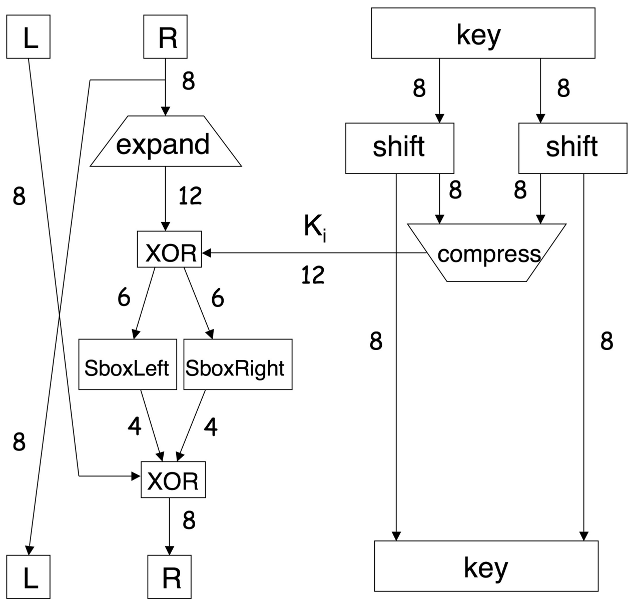

Figure 6.8: One Round of Tiny DES

We denote the left TDES S-box by `SboxLeft`(X). In hexadecimal, this S-box is

$$\begin{array}{c|cccccccccccccccc}
 & \multicolumn{16}{c}{x_1x_2x_3x_4} \\
x_0x_5 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & A & B & C & D & E & F \\
\hline
0 & 6 & 9 & A & 3 & 4 & D & 7 & 8 & E & 1 & 2 & B & 5 & C & F & 0 \\
1 & 9 & E & B & A & 4 & 5 & 0 & 7 & 8 & 6 & 3 & 2 & C & D & 1 & F \\
2 & 8 & 1 & C & 2 & D & 3 & E & F & 0 & 9 & 5 & A & 4 & B & 6 & 7 \\
3 & 9 & 0 & 2 & 5 & A & D & 6 & E & 1 & 8 & B & C & 3 & 4 & 7 & F
\end{array} \qquad (6.15)$$

whereas the right S-box, `SboxRight`(X), is

$$\begin{array}{c|cccccccccccccccc}
 & \multicolumn{16}{c}{x_1x_2x_3x_4} \\
x_0x_5 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & A & B & C & D & E & F \\
\hline
0 & C & 5 & 0 & A & E & 7 & 2 & 8 & D & 4 & 3 & 9 & 6 & F & 1 & B \\
1 & 1 & C & 9 & 6 & 3 & E & B & 2 & F & 8 & 4 & 5 & D & A & 0 & 7 \\
2 & F & A & E & 6 & D & 8 & 2 & 4 & 1 & 7 & 9 & 0 & 3 & 5 & B & C \\
3 & 0 & A & 3 & C & 8 & 2 & 1 & E & 9 & 7 & F & 6 & B & 5 & D & 4
\end{array} \qquad (6.16)$$

As with DES, each row in a TDES S-box is a permutation of the hexadecimal digits $0, 1, 2, \ldots, E, F$.

The TDES key schedule is very simple. The 16-bit key is denoted

$$K = k_0k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}k_{11}k_{12}k_{13}k_{14}k_{15}$$

and the subkey is generated as follows. Let

$$\begin{aligned} LK &= k_0k_1\dots k_7 \\ RK &= k_8k_9\dots k_{15} \end{aligned}$$

and for each round $i = 1, 2, 3, 4$,

$$\begin{aligned} LK &= \text{rotate } LK \text{ left by } 2 \\ RK &= \text{rotate } RK \text{ left by } 1. \end{aligned}$$

Then K_i is obtained by selecting bits $0, 2, 3, 4, 5, 7, 9, 10, 11, 13, 14,$ and 15 of the current (LK, RK). The subkeys K_i can be given explicitly as follows:

$$\begin{aligned} K_1 &= k_2k_4k_5k_6k_7k_1k_{10}k_{11}k_{12}k_{14}k_{15}k_8 \\ K_2 &= k_4k_6k_7k_0k_1k_3k_{11}k_{12}k_{13}k_{15}k_8k_9 \\ K_3 &= k_6k_0k_1k_2k_3k_5k_{12}k_{13}k_{14}k_8k_9k_{10} \\ K_4 &= k_0k_2k_3k_4k_5k_7k_{13}k_{14}k_{15}k_9k_{10}k_{11}. \end{aligned}$$

In the next section, we'll describe a differential attack on TDES. After that, we'll describe a linear attack on TDES. These attacks illustrate the crucial principles that apply to differential and linear cryptanalysis of DES and other block ciphers.

6.4.5 Differential Cryptanalysis of TDES

Our differential attack on TDES will focus on the right S-box, which appears above in (6.16). Suppose that we tabulate $\texttt{SboxRight}(X_1) \oplus \texttt{SboxRight}(X_2)$ for all pairs X_1 and X_2, where $X_1 \oplus X_2 = 001000$. Then we find that

$$X_1 \oplus X_2 = 001000 \implies \texttt{SboxRight}(X_1) \oplus \texttt{SboxRight}(X_2) = 0010 \quad (6.17)$$

with probability 3/4. Recall that for any S-box,

$$X_1 \oplus X_2 = 000000 \implies \texttt{SboxRight}(X_1) \oplus \texttt{SboxRight}(X_2) = 0000. \quad (6.18)$$

Our goal is to make use of these observations to develop a viable differential attack on TDES.

Differential cryptanalysis is a chosen plaintext attack. Suppose we encrypt two chosen plaintext blocks, $P = (L, R)$ and $\tilde{P} = (\tilde{L}, \tilde{R})$ that satisfy

$$P \oplus \tilde{P} = (L, R) \oplus (\tilde{L}, \tilde{R}) = 0000\ 0000\ 0000\ 0010 = \texttt{0x0002}. \quad (6.19)$$

Then P and $\tilde{P}$ differ in the one specified bit and agree in all other bit positions. Let's carefully analyze what happens to this difference as P and $\tilde{P}$ are encrypted with TDES.

First, consider

$$F(R,K) \oplus F(\tilde{R},K) = \texttt{Sboxes}(\texttt{expand}(R) \oplus K) \oplus \texttt{Sboxes}(\texttt{expand}(\tilde{R}) \oplus K).$$

From the definition of `expand` in (6.14) we see that

$$\texttt{expand}(0000\ 0010) = 000000\ 001000.$$

Since `expand` is linear, if $X_1 \oplus X_2 = 0000\ 0010$ then

$$\begin{aligned}\texttt{expand}(X_1) \oplus \texttt{expand}(X_2) &= \texttt{expand}(X_1 \oplus X_2)\\ &= \texttt{expand}(0000\ 0010)\\ &= 000000\ 001000. \end{aligned} \tag{6.20}$$

For the chosen plaintext in equation (6.19) we have $R \oplus \tilde{R} = 0000\ 0010$. Then from the observation in equation (6.20) it follows that

$$\begin{aligned}F(R,K) \oplus F(\tilde{R},K) &= \texttt{Sboxes}(\texttt{expand}(R) \oplus K) \oplus \texttt{Sboxes}(\texttt{expand}(\tilde{R}) \oplus K)\\ &= (\texttt{SboxLeft}(A \oplus K), \texttt{SboxRight}(B \oplus K))\\ &\oplus (\texttt{SboxLeft}(\tilde{A} \oplus K), \texttt{SboxRight}(\tilde{B} \oplus K))\\ &= (\texttt{SboxLeft}(A \oplus K) \oplus \texttt{SboxLeft}(\tilde{A} \oplus K)),\\ &\quad (\texttt{SboxRight}(B \oplus K) \oplus \texttt{SboxRight}(\tilde{B} \oplus K)),\end{aligned}$$

where $A \oplus \tilde{A} = 000000$ and $B \oplus \tilde{B} = 001000$. This result, together with equations (6.17) and (6.18), imply

$$F(R,K) \oplus F(\tilde{R},K) = 0000\ 0010$$

with probability 3/4.

In summary, if $R \oplus \tilde{R} = 0000\ 0010$, then for any (unknown) subkey K, we have

$$F(R,K) \oplus F(\tilde{R},K) = 0000\ 0010 \tag{6.21}$$

with probability 3/4. In other words, for certain input values, the output difference of the round function is the same as the input difference, with a high probability. Next, we'll show that we can chain this results through multiple rounds of TDES.

Since differential cryptanalysis is a chosen plaintext attack, we'll choose P and $\tilde{P}$ to satisfy equation (6.19). In Table 6.8, we carefully analyze the TDES encryption of such plaintext values. By the choice of P and $\tilde{P}$, we have

$$R_0 \oplus \tilde{R}_0 = 0000\ 0010 \ \text{ and } \ L_0 \oplus \tilde{L}_0 = 0000\ 0000.$$

Then from equation (6.21),

$$R_1 \oplus \tilde{R}_1 = 0000\ 0010$$

with probability 3/4. From this result it follows that

$$\begin{aligned} R_2 \oplus \tilde{R}_2 &= (L_1 \oplus F(R_1, K_2)) \oplus (\tilde{L}_1 \oplus F(\tilde{R}_1, K_2)) \\ &= (L_1 \oplus \tilde{L}_1) \oplus (F(R_1, K_2) \oplus F(\tilde{R}_1, K_2)) \\ &= (R_0 \oplus \tilde{R}_0) \oplus (F(R_1, K_2) \oplus F(\tilde{R}_1, K_2)) \\ &= 0000\ 0010 \oplus 0000\ 0010 \\ &= 0000\ 0000 \end{aligned}$$

with probability $(3/4)^2 = 9/16 = 0.5625$. The results given in Table 6.8 for $R_3 \oplus \tilde{R}_3$ and $R_4 \oplus \tilde{R}_4$ are obtained in a similar manner.

Table 6.8: Differential Cryptanalysis of TDES

$(L_0, R_0) = P$	$(\tilde{L}_0, \tilde{R}_0) = \tilde{P}$	$P \oplus \tilde{P} =$ 0x0002	Prob.
$L_1 = R_0$ $R_1 = L_0 \oplus F(R_0, K_1)$	$\tilde{L}_1 = \tilde{R}_0$ $\tilde{R}_1 = \tilde{L}_0 \oplus F(\tilde{R}_0, K_1)$	$(L_1, R_1) \oplus (\tilde{L}_1, \tilde{R}_1) =$ 0x0202	3/4
$L_2 = R_1$ $R_2 = L_1 \oplus F(R_1, K_2)$	$\tilde{L}_2 = \tilde{R}_1$ $\tilde{R}_2 = \tilde{L}_1 \oplus F(\tilde{R}_1, K_2)$	$(L_2, R_2) \oplus (\tilde{L}_2, \tilde{R}_2) =$ 0x0200	$(3/4)^2$
$L_3 = R_2$ $R_3 = L_2 \oplus F(R_2, K_3)$	$\tilde{L}_3 = \tilde{R}_2$ $\tilde{R}_3 = \tilde{L}_2 \oplus F(\tilde{R}_2, K_3)$	$(L_3, R_3) \oplus (\tilde{L}_3, \tilde{R}_3) =$ 0x0002	$(3/4)^2$
$L_4 = R_3$ $R_4 = L_3 \oplus F(R_3, K_4)$	$\tilde{L}_4 = \tilde{R}_3$ $\tilde{R}_4 = \tilde{L}_3 \oplus F(\tilde{R}_3, K_4)$	$(L_4, R_4) \oplus (\tilde{L}_4, \tilde{R}_4) =$ 0x0202	$(3/4)^3$
$C = (L_4, R_4)$	$C = (\tilde{L}_4, \tilde{R}_4)$	$C \oplus \tilde{C} =$ 0x0202	

We can derive an algorithm from Table 6.8 to recover some of the unknown key bits. We'll choose P and $\tilde{P}$ as in equation (6.19) and obtain the corresponding ciphertext C and $\tilde{C}$. Since TDES is a Feistel cipher,

$$R_4 = L_3 \oplus F(R_3, K_4) \ \text{ and } \ \tilde{R}_4 = \tilde{L}_3 \oplus F(\tilde{R}_3, K_4).$$

In addition, $L_4 = R_3$ and $\tilde{L}_4 = \tilde{R}_3$. Consequently,

$$R_4 = L_3 \oplus F(L_4, K_4) \ \text{ and } \ \tilde{R}_4 = \tilde{L}_3 \oplus F(\tilde{L}_4, K_4),$$

which can be rewritten as

$$L_3 = R_4 \oplus F(L_4, K_4) \ \text{ and } \ \tilde{L}_3 = \tilde{R}_4 \oplus F(\tilde{L}_4, K_4).$$

Now if

$$C \oplus \tilde{C} = \texttt{0x0202}, \tag{6.22}$$

then from Table 6.8 we almost certainly have $L_3 \oplus \tilde{L}_3 = 0000\ 0000$, that is, $L_3 = \tilde{L}_3$. It follows that

$$R_4 \oplus F(L_4, K_4) = \tilde{R}_4 \oplus F(\tilde{L}_4, K_4)$$

which we rewrite as

$$R_4 \oplus \tilde{R}_4 = F(L_4, K_4) \oplus F(\tilde{L}_4, K_4). \tag{6.23}$$

Note that, in equation (6.23), the only unknown is the subkey K_4. Next, we show how to use this result to recover some of the bits of K_4.

For a chosen plaintext pair that satisfies equation (6.19), if the resulting ciphertext pairs satisfy equation (6.22), then we know that equation (6.23) holds. Then since

$$C \oplus \tilde{C} = (L_4, R_4) \oplus (\tilde{L}_4, \tilde{R}_4) = \texttt{0x0202},$$

we have

$$R_4 \oplus \tilde{R}_4 = 0000\ 0010 \tag{6.24}$$

and we also have

$$L_4 \oplus \tilde{L}_4 = 0000\ 0010. \tag{6.25}$$

Let

$$L_4 = \ell_0\ell_1\ell_2\ell_3\ell_4\ell_5\ell_6\ell_7 \ \text{ and } \ \tilde{L}_4 = \tilde{\ell}_0\tilde{\ell}_1\tilde{\ell}_2\tilde{\ell}_3\tilde{\ell}_4\tilde{\ell}_5\tilde{\ell}_6\tilde{\ell}_7.$$

Then equation (6.25) implies that $\ell_i = \tilde{\ell}_i$ for $i = 0, 1, 2, 3, 4, 5, 7$ and $\ell_6 \neq \tilde{\ell}_6$. Now substituting equation (6.24) into equation (6.23) and expanding the definition of F, we find

$$\begin{aligned} 0000\ 0010 = \Big(&\texttt{SboxLeft}(\ell_4\ell_7\ell_2\ell_1\ell_5\ell_7 \oplus k_0k_2k_3k_4k_5k_7), \\ &\texttt{SboxRight}(\ell_0\ell_2\ell_6\ell_5\ell_0\ell_3 \oplus k_{13}k_{14}k_{15}k_9k_{10}k_{11})\Big) \\ \oplus \Big(&\texttt{SboxLeft}(\tilde{\ell}_4\tilde{\ell}_7\tilde{\ell}_2\tilde{\ell}_1\tilde{\ell}_5\tilde{\ell}_7 \oplus k_0k_2k_3k_4k_5k_7), \\ &\texttt{SboxRight}(\tilde{\ell}_0\tilde{\ell}_2\tilde{\ell}_6\tilde{\ell}_5\tilde{\ell}_0\tilde{\ell}_3 \oplus k_{13}k_{14}k_{15}k_9k_{10}k_{11})\Big). \end{aligned} \tag{6.26}$$

The left four bits of equation (6.26) give us

$$\begin{aligned} 0000 = &\texttt{SboxLeft}(\ell_4\ell_7\ell_2\ell_1\ell_5\ell_7 \oplus k_0k_2k_3k_4k_5k_7) \\ &\oplus \texttt{SboxLeft}(\tilde{\ell}_4\tilde{\ell}_7\tilde{\ell}_2\tilde{\ell}_1\tilde{\ell}_5\tilde{\ell}_7 \oplus k_0k_2k_3k_4k_5k_7), \end{aligned}$$

which holds for any choice of the bits $k_0k_2k_3k_4k_5k_7$, since $\ell_i = \tilde{\ell}_i$ for all $i \neq 6$. Therefore, we gain no information about the subkey K_4 from the left S-box.

On the other hand, the right four bits of equation (6.26) give us

$$\begin{aligned} 0010 &= \texttt{SboxRight}(\ell_0\ell_2\ell_6\ell_5\ell_0\ell_3 \oplus k_{13}k_{14}k_{15}k_9k_{10}k_{11}) \\ &\oplus \texttt{SboxRight}(\tilde{\ell}_0\tilde{\ell}_2\tilde{\ell}_6\tilde{\ell}_5\tilde{\ell}_0\tilde{\ell}_3 \oplus k_{13}k_{14}k_{15}k_9k_{10}k_{11}), \end{aligned} \tag{6.27}$$

which must hold for the correct choice of subkey bits $k_{13}k_{14}k_{15}k_9k_{10}k_{11}$ and will only hold with some probability for an incorrect choice of these subkey bits. Since the right S-box and the bits of L_4 and $\tilde{L}_4$ are known, we can determine the unknown subkey bits that appear in equation (6.27). The algorithm for recovering these key bits is given in Table 6.9.

Table 6.9: Algorithm to Recover Subkey Bits

```
count[$i$] = 0, for $i = 0, 1, \ldots, 63$
for $i = 1$ to iterations
    Choose $P$ and $\tilde{P}$ with $P \oplus \tilde{P} =$ 0x0002
    Obtain corresponding $C = c_0c_1 \ldots c_{15}$ and $\tilde{C} = \tilde{c}_0\tilde{c}_1 \ldots \tilde{c}_{15}$
    if $C \oplus \tilde{C} =$ 0x0202 then
        $\ell_i = c_i$ and $\tilde{\ell}_i = \tilde{c}_i$ for $i = 0, 1, \ldots, 7$
        for $K = 0$ to 63
            if 0010 == (SboxRight($\ell_0\ell_2\ell_6\ell_5\ell_0\ell_3 \oplus K$)
                        $\oplus$ SboxRight($\tilde{\ell}_0\tilde{\ell}_2\tilde{\ell}_6\tilde{\ell}_5\tilde{\ell}_0\tilde{\ell}_3 \oplus K$)) then
                increment count[$K$]
            end if
        next $K$
    end if
next $i$
```

Each time the `for` loop in Table 6.9 is executed, count[K] will be incremented for the correct subkey bits, that is, for $K = k_{13}k_{14}k_{15}k_9k_{10}k_{11}$, while for other indices K the count will be incremented with some probability. Consequently, the maximum counts indicate possible subkey values. There may be more than one such maximum count, but with a sufficient number of iterations, the number of such counts should be small.

In one particular test case of the algorithm in Table 6.9, we generated 100 pairs P and $\tilde{P}$ that satisfy $P \oplus \tilde{P} =$ `0x0002`. We found that 47 of the resulting ciphertext pairs satisfied $C \oplus \tilde{C} =$ `0x0202`, and for each of these we tried all 64 possible 6-bit subkeys as required by the algorithm in Table 6.9. In this experiment, we found that each of the four putative subkeys 000001, 001001, 110000, and 000111 had the maximum count of 47, while no other had a count greater than 39. We conclude that subkey K_4 must be one of these four values. Then from the definition of K_4 we have

$$k_{13}k_{14}k_{15}k_9k_{10}k_{11} \in \{000001, 001001, 110000, 111000\},$$

which is equivalent to

$$k_{13}k_{14}k_9k_{10}k_{11} \in \{00001, 11000\}. \tag{6.28}$$

In this case, the key is

$$K = 1010\ 1001\ 1000\ 0111,$$

so that $k_{13}k_{14}k_9k_{10}k_{11} = 11000$, which appears in equation (6.28), as expected.

Of course, if we're the attacker, we don't know the key, so, to complete the recovery of K, we could exhaustively search over the remaining 2^{11} unknown key bits, and for each of these try both of the possibilities in equation (6.28). For each of these 2^{12} putative keys K, we would try to decrypt the ciphertext, and for the correct key, we will recover the plaintext. We expect to try about half of the possibilities—about 2^{11} keys—before finding the correct key K.

The total expected work to recover the entire key K by this method is about 2^{11} encryptions, plus the work required for the differential attack, which is insignificant in comparison. As a result, we can recover the entire 16-bit key with a work factor of about 2^{11} encryptions, which is much better than an exhaustive key search, since an exhaustive search has an expected work of 2^{15} encryptions. This shows that a shortcut attack exists, and as a result TDES is insecure.

6.4.6 Linear Cryptanalysis of TDES

The linear cryptanalysis of TDES is simpler than the differential cryptanalysis. Whereas the differential cryptanalysis of TDES focused on the right S-box, our linear cryptanalysis attack will focus on the left S-box, which appears above in (6.15).

With the notation

$$y_0y_1y_2y_3 = \texttt{SboxLeft}(x_0x_1x_2x_3x_4x_5),$$

it's easy to verify that for the left S-box of TDES, the linear approximations

$$y_1 = x_2 \text{ and } y_2 = x_3 \tag{6.29}$$

each hold with probability 3/4. To develop a linear attack based on these equations, we must be able to chain these results through multiple rounds.

Denote the plaintext by $P = (L_0, R_0)$ and let $R_0 = r_0r_1r_2r_3r_4r_5r_6r_7$. Then the expansion permutation is given by

$$\texttt{expand}(R_0) = \texttt{expand}(r_0r_1r_2r_3r_4r_5r_6r_7) = r_4r_7r_2r_1r_5r_7r_0r_2r_6r_5r_0r_3. \quad (6.30)$$

From the definition of F in equation (6.13), we see that the input to the S-boxes in round one is given by $\texttt{expand}(R_0) \oplus K_1$. Then, from equation (6.30) and the definition of subkey K_1, we see that the input to the left S-box in round one is

$$r_4r_7r_2r_1r_5r_7 \oplus k_2k_4k_5k_6k_7k_1.$$

Let $y_0y_1y_2y_3$ be the round-one output of the left S-box. Then equation (6.29) implies that

$$y_1 = r_2 \oplus k_5 \text{ and } y_2 = r_1 \oplus k_6, \quad (6.31)$$

where each equality holds with probability 3/4. In other words, for the left S-box, output bit number 1 is input bit number 2, XORed with a bit of key, and output bit number 2 is input bit number 1, XORed with a key bit, where each of these hold with probability 3/4.

In TDES (as in DES) the output of the S-boxes is XORed with the bits of the old left half. Let $L_0 = \ell_0\ell_1\ell_2\ell_3\ell_4\ell_5\ell_6\ell_7$ and let $R_1 = \tilde{r}_0\tilde{r}_1\tilde{r}_2\tilde{r}_3\tilde{r}_4\tilde{r}_5\tilde{r}_6\tilde{r}_7$. Then the output of the left S-box from round one is XORed with $\ell_0\ell_1\ell_2\ell_3$ to yield $\tilde{r}_0\tilde{r}_1\tilde{r}_2\tilde{r}_3$. Combining this notation with equation (6.31), we have

$$\tilde{r}_1 = r_2 \oplus k_5 \oplus \ell_1 \text{ and } \tilde{r}_2 = r_1 \oplus k_6 \oplus \ell_2, \quad (6.32)$$

where each of these equations holds with probability 3/4. An analogous result holds for subsequent rounds, where the specific key bits depend on the subkey K_i.

As a result of equation (6.32), we can chain the linear approximation in equation (6.29) through multiple rounds. This is illustrated in Table 6.10. Since linear cryptanalysis is a known plaintext attack, the attacker knows the plaintext $P = p_0p_1p_2 \ldots p_{15}$ and corresponding ciphertext $C = c_0c_1c_2 \ldots c_{15}$.

The final row in Table 6.10 follows from the fact $L_4 = c_0c_1c_2c_3c_4c_5c_6c_7$. We can rewrite these equations as

$$k_0 \oplus k_1 = c_1 \oplus p_{10} \quad (6.33)$$

and

$$k_7 \oplus k_2 = c_2 \oplus p_9 \quad (6.34)$$

where both hold with probability $(3/4)^3$. Since c_1, c_2, p_9, and p_{10} are all known, we have obtained some information about the key bits k_0, k_1, k_2, and k_7.

Table 6.10: Linear Cryptanalysis of TDES

$(L_0, R_0) = (p_0 \ldots p_7, p_8 \ldots p_{15})$	Bits 1 and 2 (numbered from 0)	Probability
$L_1 = R_0$	p_9, p_{10}	1
$R_1 = L_0 \oplus F(R_0, K_1)$	$p_1 \oplus p_{10} \oplus k_5,\ p_2 \oplus p_9 \oplus k_6$	$3/4$
$L_2 = R_1$	$p_1 \oplus p_{10} \oplus k_5,\ p_2 \oplus p_9 \oplus k_6$	$3/4$
$R_2 = L_1 \oplus F(R_1, K_2)$	$p_2 \oplus k_6 \oplus k_7,\ p_1 \oplus k_5 \oplus k_0$	$(3/4)^2$
$L_3 = R_2$	$p_2 \oplus k_6 \oplus k_7,\ p_1 \oplus k_5 \oplus k_0$	$(3/4)^2$
$R_3 = L_2 \oplus F(R_2, K_3)$	$p_{10} \oplus k_0 \oplus k_1,\ p_9 \oplus k_7 \oplus k_2$	$(3/4)^3$
$L_4 = R_3$	$p_{10} \oplus k_0 \oplus k_1,\ p_9 \oplus k_7 \oplus k_2$	$(3/4)^3$
$R_4 = L_3 \oplus F(R_3, K_4)$		
$C = (L_4, R_4)$	$c_1 = p_{10} \oplus k_0 \oplus k_1,\ c_2 = p_9 \oplus k_7 \oplus k_2$	$(3/4)^3$

It's easy to implement a linear attack based on the results in Table 6.10. We are given the known plaintexts $P = p_0p_1p_2 \ldots p_{15}$ along with the corresponding ciphertext $C = c_0c_1c_2 \ldots c_{15}$. For each such pair, we increment a counter depending on whether

$$c_1 \oplus p_{10} = 0 \ \text{ or } \ c_1 \oplus p_{10} = 1$$

and another counter depending on whether

$$c_2 \oplus p_9 = 0 \ \text{ or } \ c_2 \oplus p_9 = 1.$$

Using 100 known plaintexts the following results were obtained:

$$\begin{aligned} c_1 \oplus p_{10} &= 0 \ \text{ occurred 38 times} \\ c_1 \oplus p_{10} &= 1 \ \text{ occurred 62 times} \\ c_2 \oplus p_9 &= 0 \ \text{ occurred 62 times} \\ c_2 \oplus p_9 &= 1 \ \text{ occurred 38 times.} \end{aligned}$$

In this case, we conclude from equation (6.33) that

$$k_0 \oplus k_1 = 1$$

and from equation (6.34) that

$$k_7 \oplus k_2 = 0.$$

In this example, the actual key is

$$K = 1010\ 0011\ 0101\ 0110,$$

and it's easily verified that $k_0 \oplus k_1 = 1$ and $k_7 \oplus k_2 = 0$ as we determined via the linear attack.

In this linear attack, we have only recovered the equivalent of two bits of information. To recover the entire key K, we could do an exhaustive key search for the remaining unknown bits. This would require an expected work of about 2^{13} encryptions and the work for the linear attack, which is negligible in comparison. While this may not seem too significant, it is a shortcut attack, and so it shows that TDES is insecure according to our definition.

6.4.7 Implications Block Cipher Design

Since there is no way to prove that a practical cipher is secure and since it's difficult to protect against unknown attacks, cryptographers focus on preventing known attacks. For block ciphers, the known attacks are, primarily, linear and differential cryptanalysis—and variations on these approaches. Thus the primary goal in block cipher design is to make linear and differential attacks infeasible.

How can cryptographers make linear and differential attacks more difficult? For an iterated block cipher, there is a fundamental trade-off between the number of rounds and the complexity of each round. That is, a simple round function will generally require a larger number of rounds to achieve the same degree of confusion and diffusion as a more complex function could achieve in fewer iterations.

In both linear and differential attacks, any one-round success probability that is less than 1 will almost certainly diminish with each subsequent round. Consequently, all else being equal, a block cipher with more rounds will be more secure from linear and differential attacks.

Another way to make linear and differential attacks more difficult is to have a high degree of confusion. That is, we can strive to reduce the success probability per round. For a DES-like cipher, this is equivalent to building better S-boxes. All else being equal—which it never is—more confusion means more security.

On the other hand, better diffusion will also tend to make linear and differential attacks harder to mount. In both types of attacks, it is necessary to chain results through multiple rounds, and better diffusion will make it harder to connect one-round successes into usable chains.

In TDES, the number of rounds is small, and, as a result, the one-round success probabilities are not sufficiently diminished during encryption. Also, the TDES S-boxes are poorly designed, resulting in limited confusion. Finally, the TDES `expand` permutation—the only source of diffusion in the cipher—does a poor job of mixing the bits of one round into the next round. All of these combine to yield a cipher that is highly susceptible to both linear and differential attacks.

To complicate the lives of block cipher designers, they must construct ciphers that are secure and efficient. One of the fundamental issues that block cipher designers must contend with is the inherent trade-off between the number of rounds and the complexity of each round. That is, a block cipher with a simple round structure will tend to provide limited mixing (diffusion) and limited nonlinearity (confusion), and consequently more rounds will be required.

The Tiny Encryption Algorithm (TEA) is a good example of a block cipher with a simple round structure. Since each round of TEA is extremely simple, the resulting confusion and diffusion properties are fairly weak, which necessitates a large number of rounds. At the other extreme, each round of the Advanced Encryption Standard (AES) has strong linear mixing and excellent nonlinear properties. So a relatively small number of AES rounds are needed, but each AES round is more complex than a round of TEA. Finally, DES could be viewed as residing in between these two extremes.

6.5 Lattice Reduction and the Knapsack

Every private in the French army carries a Field Marshal wand in his knapsack.
— Napoleon Bonaparte

In this section we present the details of the attack on the original Merkle-Hellman knapsack cryptosystem. This knapsack cryptosystem is discussed in Section 4.2 of Chapter 4. For a more rigorous (but still readable) presentation of the attack discussed here, see [176]. Note that some elementary linear algebra is required in this section. The Appendix contains a review of the necessary material.

Let b_1,b_2,…,b_n be vectors in $\mathtt{R}^m$, that is, each b_i is a (column) vector consisting of exactly m real numbers. A *lattice* is the set of all multiples of the vector b_i of the form

$$\alpha_1 b_1 + \alpha_2 b_2 + \cdots + \alpha_n b_n,$$

where each α_i in an integer.

For example, consider the vectors

$$b_1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \text{ and } b_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}. \tag{6.35}$$

Since b_1 and b_2 are linearly independent, any point in the plane can be written as $\alpha_1 b_1 + \alpha_2 b_2$ for some real numbers α_1 and α_2. We say that the plane $\mathtt{R}^2$ is *spanned* by the pair (b_1, b_2). If we restrict α_1 and α_2 to integers, then the resulting span, that is, all points of the form $\alpha_1 b_1 + \alpha_2 b_2$, is a *lattice*. A lattice consists of a discrete set of points. For example, the lattice spanned by the vectors in equation (6.35) is illustrated in Figure 6.9.

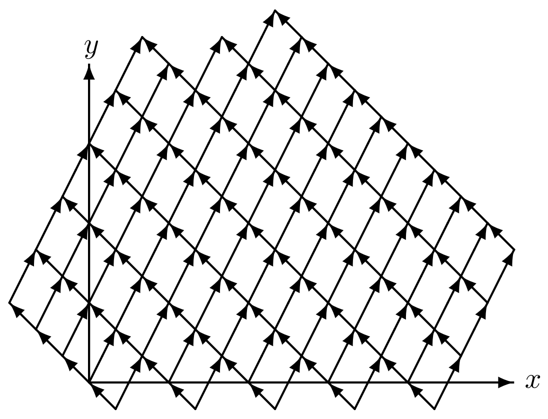

Figure 6.9: A Lattice in the Plane

Many combinatorial problems can be reduced to the problem of finding a "short" vector in a lattice. The knapsack is one such problem. Short vectors in a lattice can be found using a technique known as *lattice reduction*.

Before discussing the lattice reduction attack on the knapsack, let's first consider another combinatorial problem that can be solved using this technique. The problem that we'll consider is the *exact cover*, which can be stated as follows. Given a set S and a collection of subsets of S, find a collection of these subsets where each element of S is in exactly one subset. It's not always possible to find such a collection of subsets, but if it is, we'll see that the solution is a short vector in a particular lattice.

Consider the following example of the exact cover problem. Let

$$S = \{0,1,2,3,4,5,6\}$$

and suppose we are given 13 subsets of S, which we label s_0 through s_{12} as follows:

$$\begin{aligned}&s_0 = \{0,1,3\}, s_1 = \{0,1,5\}, s_2 = \{0,2,4\}, s_3 = \{0,2,5\},\\ &s_4 = \{0,3,6\}, s_5 = \{1,2,4\}, s_6 = \{1,2,6\}, s_7 = \{1,3,5\},\\ &s_8 = \{1,4,6\}, s_9 = \{1\}, s_{10} = \{2,5,6\}, s_{11} = \{3,4,5\}, s_{12} = \{3,4,6\}.\end{aligned}$$

Denote the number of elements of S by m and the number of subsets by n. In this example, we have $m = 7$ and $n = 13$. Can we find a collection of these 13 subsets where each element of S is in exactly one subset?

There are 2^{13} different collections of the 13 subsets, so we could exhaustively search through all possible collections until we find such a collection—or until we've tried them all, in which case we would conclude that no such collection exists. But if there are too many subsets, then we need an alternative approach.

One alternative is to try a *heuristic search* technique. There are many different types of heuristic search strategies, but what they all have in common is that they search through the set of possible solutions in a nonrandom manner. The goal of such a search strategy is to search in a "smart" way to improve the odds of finding a solution sooner than an exhaustive search.

Lattice reduction can be viewed as a form of heuristic search. As a result, we are not assured of finding a solution using lattice reduction, but for many problems this techniques yields a solution with a high probability, yet the work required is small in comparison to an exhaustive search.

Before we can apply the lattice reduction method, we first need to rewrite the exact cover problem in matrix form. We define an $m \times n$ matrix A, where $a_{ij} = 1$ if element i of S is in subset s_j and otherwise $a_{ij} = 0$. Also, we define B to be a vector of length m consisting of all 1s. Then, if we can solve the matrix equation $AU = B$ for a vector U of 0s and 1s, we have solved the exact cover problem.

For the exact cover example above, the matrix equation $AU = B$ has the form

$$\begin{bmatrix} 1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 1&1&0&0&0&1&1&1&1&1&0&0&0 \\ 0&0&1&1&0&1&1&0&0&0&1&0&0 \\ 1&0&0&0&1&0&0&1&0&0&0&1&1 \\ 0&0&1&0&0&1&0&0&1&0&0&1&1 \\ 0&1&0&1&0&0&0&1&0&0&1&1&0 \\ 0&0&0&0&1&0&1&0&1&0&1&0&1 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \\ u_{10} \\ u_{11} \\ u_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

and we seek a solution U where each $u_i \in \{0, 1\}$, that is, $u_i = 1$ if the subset s_i is in the exact cover and $u_i = 0$ if subset s_i is not in the exact cover. In this particular case, it's easy to verify that a solution is given by $U = [0001000001001]$, that is, s_3, s_9, and s_{12} form an exact cover of the set S.

We have shown that the exact cover problem can be restated as finding a solution U to a matrix equation $AU = B$, where U consists entirely of 0s and 1s. This is not a standard linear algebra problem, since solutions to linear equations are not restricted to contain only 0s and 1s. This turns out to be a problem that can be solved using lattice reduction techniques. But first we need an elementary fact from linear algebra.

Suppose $AU = B$, where A is a matrix and U and B are column vectors. Let $a_1, a_2, \ldots, a_n$ denote the columns of A and $u_1, u_2, \ldots, u_n$ the elements of U. Then

$$B = u_1 a_1 + u_2 a_2 + \cdots + u_n a_n. \tag{6.36}$$

For example,

$$\begin{bmatrix} 3 & 4 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} 2 \\ 6 \end{bmatrix} = 2 \begin{bmatrix} 3 \\ 1 \end{bmatrix} + 6 \begin{bmatrix} 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 30 \\ 32 \end{bmatrix}.$$

Now given $AU = B$, consider the matrix equation

$$\begin{bmatrix} I_{n\times n} & 0_{n\times 1} \\ A_{m\times n} & -B_{m\times 1} \end{bmatrix} \begin{bmatrix} U_{n\times 1} \\ 1_{1\times 1} \end{bmatrix} = \begin{bmatrix} U_{n\times 1} \\ 0_{m\times 1} \end{bmatrix},$$

which we denote as $MV = W$. Multiplying, we find that $U = U$ (which is not very informative) and the nontrivial equation $AU - B = 0$. Therefore, finding a solution V to $MV = W$ is equivalent to finding a solution U to the original equation $AU = B$.

The benefit of rewriting the problem as $MV = W$ is that the columns of M are linearly independent. This is easily seen to be the case, since the $n \times n$ identity matrix appears in the upper left, and the final column begins with n zeros.

Let $c_0, c_1, c_2, \ldots, c_n$ be the $n + 1$ columns of M and let $v_0, v_1, v_2, \ldots, v_n$ be the elements of V. Then, by the observation in equation (6.36),

$$W = v_0 c_0 + v_1 c_1 + \cdots + v_n c_n. \tag{6.37}$$

Let $\mathtt{L}$ be the lattice spanned by $c_0, c_1, c_2, \ldots, c_n$, the columns of M. Then $\mathtt{L}$ consists of all integer multiples of the columns of M. Recall that $MV = W$, where

$$W = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Our goal is to find U. However, instead of solving linear equations for V, we can solve for U by finding W. By equation (6.37), this desired solution W is in the lattice $\mathtt{L}$.

The Euclidean length of a vector $Y = (y_0, y_1, \ldots, y_{n-1}) \in \mathtt{R}^n$ is given by the formula

$$||Y|| = \sqrt{y_0^2 + y_1^2 + \cdots + y_{n-1}^2}$$

Then the length of W is

$$||W|| = \sqrt{u_0^2 + u_1^2 + \cdots + u_{n-1}^2} \leq \sqrt{n}.$$

Since most vectors in L will have a length far greater than $\sqrt{n}$, we see that W is a short vector in the lattice L. Furthermore, W has a very special form, with its first n entries all equal to 0 or 1 and its last m entries all equal to 0. These facts distinguish W from typical vectors in L. Can we use this information to find W, which would give us a solution to the exact cover problem?

In fact, there is an algorithm known as the LLL algorithm [170, 190] (because it was invented by three guys whose names start with "L") to efficiently find short vectors in a lattice. Our strategy will be to use LLL to find short vectors in L, the lattice spanned by the columns of M. Then we'll examine these short vectors to see whether any have the special form of W. If we find such a vector, then it is highly probably that we have found a solution U to the original problem.

Pseudo-code for the LLL algorithm appears in Table 6.11, where the $(n+m) \times (n+1)$ matrix M has columns $b_0, b_1, b_2, \ldots, b_n$ and the columns of matrix X are denoted $x_0, x_1, x_2 \ldots, x_n$ and the elements of Y are denoted as y_{ij}. Note that the y_{ij} can be negative, so care must be taken when implementing the floor function in $\lfloor y_{ij} + 1/2 \rfloor$.

For completeness, we've given the Gram-Schmidt orthogonalization algorithm in Table 6.12. Combined, these two algorithms only require about 30 lines of pseudo-code.

It's important to realize there is no guarantee that the LLL algorithm will find the desired vector W. But for certain types of problems, the probability of success is high.

By now, you may be wondering what any of this has to do with the knapsack cryptosystem. Next, we'll show that we can attack the knapsack via lattice reduction.

Let's consider the superincreasing knapsack

$$S = [s_0, s_1, \ldots, s_7] = [2, 3, 7, 14, 30, 57, 120, 251]$$

and choose the multiplier $m = 41$ and modulus $n = 491$ (note that this is the same knapsack example that appears in Section 4.2 of Chapter 4). Next, we observe that $m^{-1} = 12 \bmod 491$. Now to find the corresponding public knapsack, we compute $t_i = 41 s_i \bmod 491$ for $i = 0, 1, \ldots, 7$, and the result is

$$T = [t_0, t_1, \ldots, t_7] = [82, 123, 287, 83, 248, 373, 10, 471].$$

This yields the knapsack cryptosystem defined by

Public key: T

Table 6.11: LLL Algorithm

```
     // find short vectors in the lattice spanned
     // by columns of $M = (b_0, b_1, \ldots, b_n)$
     loop forever
         $(X, Y) = \text{GS}(M)$
         for $j = 1$ to $n$
             for $i = j - 1$ to 0
                 if $|y_{ij}| > 1/2$ then
                     $b_j = b_j - \lfloor y_{ij} + 1/2 \rfloor b_i$
                 end if
             next $i$
         next $j$
         $(X, Y) = \text{GS}(M)$
         for $j = 0$ to $n - 1$
             if $||x_{j+1} + y_{j,j+1} x_j||^2 < \frac{3}{4} ||x_j||^2$
                 swap$(b_j, b_{j+1})$
                 goto abc
             end if
         next $j$
         return$(M)$
abc:     continue
     end loop
```

and

$$\text{Private key: } S \text{ and } m^{-1} \bmod n.$$

For example, 10010110 is encrypted as

$$\begin{aligned} 1 \cdot t_0 + 0 \cdot t_1 + 0 \cdot t_2 + 1 \cdot t_3 + 0 \cdot t_4 + 1 \cdot t_5 + 1 \cdot t_6 + 0 \cdot t_7 \\ = 82 + 83 + 373 + 10 \\ = 548. \end{aligned}$$

To decrypt the ciphertext 548, the holder of the private key computes

$$548 \cdot 12 = 193 \bmod 491$$

and then uses the superincreasing knapsack S to easily solve for the plaintext 10010110.

In this particular example, the attacker Trudy knows the public key T and the ciphertext 548. Trudy can break the system if she can find $u_i \in \{0, 1\}$ so that

$$82u_0 + 123u_1 + 287u_2 + 83u_3 + 248u_4 + 373u_5 + 10u_6 + 471u_7 = 548. \quad (6.38)$$

Table 6.12: Gram-Schmidt Algorithm

```
// Gram-Schmidt $M = (b_0, b_1, \ldots, b_n)$
GS($M$)
    $x_0 = b_0$
    for $j = 1$ to $n$
        $x_j = b_j$
        for $i = 0$ to $j - 1$
            $y_{ij} = (x_i \cdot b_j)/||x_i||^2$
            $x_j = x_j - y_{ij} x_i$
        next $i$
    next $j$
    return($X$,$Y$)
end GS
```

To put this problem into the correct framework for lattice reduction, we rewrite the problem in matrix form as

$$T \cdot U = 548,$$

where T is the public knapsack and $U = [u_0, u_1, \ldots, u_7]$ appears in equation (6.38). This has the same form as $AU = B$ discussed above, so we rewrite this to put it into the form $MV = W$, which is then suitable for the LLL algorithm. In this case, we have

$$M = \left[\begin{array}{cc} I_{8\times 8} & 0_{8\times 1} \\ T_{1\times 8} & -C_{1\times 1} \end{array}\right] = \left[\begin{array}{cccccccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 82 & 123 & 287 & 83 & 248 & 373 & 10 & 471 & -548 \end{array}\right].$$

We can now apply LLL to the matrix M to find short vectors in the lattice spanned by the columns of M. The output of LLL, which we denote by M' is a matrix of short vectors in the lattice spanned by the columns of M. In this example, LLL yields

$$M' = \left[\begin{array}{cccccccc|c} -1 & -1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & -1 & 1 & 2 \\ 1 & -1 & -1 & 1 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -2 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 \\ \hline 1 & -1 & 1 & 0 & 0 & 1 & -1 & 2 & 0 \end{array}\right].$$

The 4th column of M' has the correct form to be a solution to the knapsack problem. For this column, Trudy obtains the putative solution

$$U = [1, 0, 0, 1, 0, 1, 1, 0]$$

and using the public key and the ciphertext, she can then easily verify that the putative solution 10010110 is, in fact, the correct solution. One interesting aspect of this particular attack is that Trudy can find the plaintext from the ciphertext without recovering the private key.

The lattice reduction attack on the knapsack is fast and efficient—it was originally demonstrated using an Apple II computer in 1983 [266]. Although the attack is not always successful, the probability of success against the original Merkle-Hellman knapsack is high.

Lattice reduction was a surprising method of attack on the knapsack cryptosystem. The lesson here is that clever mathematics (and algorithms) can sometimes break cryptosystems.

6.6 RSA Timing Attacks

All things entail rising and falling timing.
You must be able to discern this.
— Miyamoto Musashi

Often it's possible to attack a cipher without directly attacking the algorithm [89]. Many processes produce unintended "side channels" that leak information. This incidental information can arise due to the way that a computation is performed, the media used, the power consumed, electromagnetic emanations, and so on. In some cases, this information can be used to recover a cryptographic key.

Paul Kocher, the father of side channel attacks [167], originally developed the technique as a way to demonstrate the vulnerablity of smartcards. Kocher singlehandedly delayed the widespread acceptance of smartcards by several years.

A large potential source of side channel information arises from so-called unintended emanations. There is an entire branch of security devoted to emissions security, or EMSEC, which also goes by the name of TEMPEST [200]. For example, Anderson [14] describes how electromagnetic fields, or EMF, from a computer screen can allow the screen image to be reconstructed at a distance.

Smartcards have been attacked via their EMF emanations as well as by *differential power analysis*, or DPA, which exploits the fact that some computations require more energy consumption than others [168]. Attacks on EMF emissions and DPA attacks are passive. More active attacks often go by the name of *differential fault analysis*, or DFA, where faults are induced with the goal of recovering information [11]. For example, excessive power may be put into a device to induce a fault. Such attacks may or may not be destructive. A smartcard used in some GSM cell phones could be attacked using DFA techniques [229].

In this section, we'll examine two timing attack on RSA. The first approach is impractical, but provides a relatively simple illustration of the concept, while the second attack has been used in the real world to break real systems.

Timing attacks exploit the fact that some computations in RSA take longer than others. By carefully measuring the time that an operation takes, we can determine the RSA private key, or at least some bits of the key [330]. More advanced versions of timing attacks have been used to successfully attack the RSA implementation in OpenSSL over a network connection [41]. For a discussion of timing attacks that apply to more general RSA implementations, see [285].

6.6.1 A Simple Timing Attack

Let M be a message that Alice is to sign using her private key d. Suppose that Alice signs M itself,[6] that is, Alice computes $M^d \bmod N$. As usual, Trudy's goal is to recover d. We'll assume that d is $n+1$ bits in length, with n unknown bits, and we'll denote the bits of d as

$$d = d_0 d_1 \ldots d_n \text{ where } d_0 = 1.$$

Recall that the method of repeated squaring provides an efficient means of computing modular exponentiation. Suppose repeated squaring is used

[6]The astute reader will recall that in Chapter 5 we said that Alice signs $h(M)$, not M. However, in security protocols, it's common to sign a random challenge without any hash being used—see Chapters 9 and 10. Many timing attacks arise in the context of security protocols, so here we'll consider the case where the message M is signed, without any hash.

to compute $M^d \bmod N$. Pseudo-code for the repeated squaring algorithm appears in Table 6.13.

Table 6.13: Repeated Squaring

```
$x = M$
for $j = 1$ to $n$
    $x = \text{mod}(x^2, N)$
    if $d_j == 1$ then
        $x = \text{mod}(xM, N)$
    end if
next $j$
return $x$
```

Suppose that the $\text{mod}(x, N)$ function in Table 6.13 is implemented as shown in Table 6.14. For efficiency, the expensive mod operation, denoted by "%," is only executed if a modular reduction is actually required.

Table 6.14: Efficient Mod Function

```
function $\text{mod}(x, N)$
if $x >= N$
    $x = x \,\%\, N$
end if
return $x$
```

Now consider the repeated squaring algorithm in Table 6.13. If $d_j = 0$, then $x = \text{mod}(x^2, N)$, but if $d_j = 1$ then two operations occur, namely, $x = \text{mod}(x^2, N)$ and $x = \text{mod}(xM, N)$. As a result, the computation times might differ when $d_j = 0$ compared with when $d_j = 1$. Can Trudy take advantage of this to recover Alice's private key?

We'll assume that Trudy can conduct a "chosen plaintext" attack, that is, Alice will sign messages of Trudy's choosing. Suppose clever Trudy chooses two values, Y and Z, with $Y^3 < N$ and $Z^2 < N < Z^3$ and Alice signs both.

Let $x = Y$ and consider the $j = 1$ step in the repeated squaring algorithm of Table 6.13. We have

$$x = \text{mod}(x^2, N)$$

and since $x^2 = Y^2 < Y^3 < N$, the "%" operation does not occur. Then, if $d_1 = 1$, we have

$$x = \text{mod}(xY, N),$$

and since $xY = Y^3 < N$, again the "%" operation does not occur. Of course, if $d_1 = 0$, this "%" operation does not occur either.

Now let $x = Z$ and consider the $j = 1$ step in the algorithm of Table 6.13. In this case, we have

$$x = \text{mod}(x^2, N)$$

and, since $x^2 = Z^2 < N$, the "%" operation does not occur. But if $d_1 = 1$, we have

$$x = \text{mod}(xZ, N)$$

and the "%" operation occurs, since $xZ = Z^3 > N$. However, if $d_1 = 0$, then this "%" operation does not occur. That is, an additional "%" operation occurs only if $d_1 = 1$. As a result, if $d_1 = 1$ then the $j = 1$ step requires more computation and will take longer to complete for Z than for Y. If, on the other hand, $d_1 = 0$, the $j = 1$ computation step will take about the same amount of time for both Z and Y. Using this fact, can Trudy recover the bit d_1 of the private key d?

The problem for Trudy is that the repeated squaring algorithm does not stop after the $j = 1$ step. So, any timing difference in the $j = 1$ step might be swamped by timing differences that occur at later steps. But suppose Trudy can repeat this experiment many times with distinct Y and Z values, all of which satisfy the conditions given above, namely, $Y^3 < N$ and $Z^2 < N < Z^3$. Then if $d_1 = 0$, on average, Trudy would expect the Y and Z signatures to take about the same time. On the other hand, if $d_1 = 1$, then Trudy would expect the Z signatures to take longer than the Y signatures, on average. That is, timing differences for later steps in the algorithm would tend to cancel out, allowing the timing difference (or not) for the $j = 1$ step show through the noise. The point is that Trudy will need to rely on statistics gathered over many test cases to make this attack reliable.

Trudy can use the following algorithm to determine the unknown private key bit d_1. For $i = 0, 1, \ldots, m-1$, Trudy chooses Y_i with $Y_i^3 < N$. Let y_i be the time required for Alice to sign Y_i, that is, the time required to compute $Y_i^d \bmod N$, for $i = 0, 1, \ldots, m-1$. Then Trudy computes the average timing

$$y = (y_0 + y_1 + \cdots + y_{m-1})/m.$$

Next, for $i = 0, 1, \ldots, m-1$, Trudy chooses Z_i with $Z_i^2 < N < Z_i^3$. Let z_i be the time required to compute $Z_i^d \bmod N$, for $i = 0, 1, \ldots, m-1$. Again, Trudy computes the average timing

$$z = (z_0 + z_1 + \cdots + z_{m-1})/m.$$

Now if $z > y + \varepsilon$ then Trudy would assume that $d_1 = 1$, and otherwise she would assume $d_1 = 0$, where an appropriate value for ε could be determined by experimentation.

Once d_1 has been recovered, Trudy can use an analogous process to find d_2, although for this next step the Y and Z values will need to be chosen to satisfy different criteria. And once d_2 is known, Trudy can proceed to d_3 and so on—see Problem 31.

The attack discussed in this section is only practical for recovering the first few bits of the private key. Next, we discuss a more realistic timing attack that has been used to recover RSA private keys from smartcards and other resource-constrained devices.

6.6.2 Kocher's Timing Attack

The basic idea behind Kocher's timing attack [167] is elegant, yet reasonably straightforward. Suppose that the repeated squaring algorithm in Table 6.15 is used for modular exponentiation in RSA. Also, suppose that the time taken by the multiplication operation, $s = s \cdot x \pmod N$ in Table 6.15, varies depending on the values of s and x. Furthermore, we assume the attacker is able to determine the timings that will occur, given particular values of s and x.

Table 6.15: Repeated Squaring

```
// Compute $y = x^d \pmod N$,
// where $d = d_0 d_1 d_2 \ldots d_n$ in binary, with $d_0 = 1$
$s = x$
for $i = 1$ to $n$
    $s = s^2 \pmod N$
    if $d_i == 1$ then
        $s = s \cdot x \pmod N$
    end if
next $i$
return($s$)
```

Kocher views this as a signal detection problem, where the "signal" consists of the timing variations, which are dependent on the unknown private key bits d_i, for $i = 1, 2, \ldots, n$. The signal is corrupted by "noise," which is the result of the unknown private key bits, d_i. The objective is to recover the bits d_i one (or a few) at a time, beginning with the first unknown bit d_1. In practice, it is not necessary to recover all of the bits, since an algorithm due to Coppersmith [68] is feasible once a sufficient number of the high-order bits of d are known.

Suppose we have successfully determined bits $d_0, d_1, \ldots, d_{k-1}$ and we want to determine bit d_k. Then we randomly select several ciphertexts, say, C_j, for $j = 0, 1, 2, \ldots, m-1$, and for each we obtain the timing $T(C_j)$ for the decryption (or signature) $C_j^d \pmod N$. For each of these ciphertext values, we can precisely emulate the repeated squaring algorithm in Table 6.15 for $i = 1, 2, \ldots, k-1$, and at the $i = k$ step we can emulate both of the possible bit values, $d_k = 0$ and $d_k = 1$. Then we tabulate the differences between the measured timing and both of the emulated results. Kocher's crucial observation is that the statistical variance of the differences will be smaller for the correct choice of d_k than for the incorrect choice.

For example, suppose we are trying to obtain a private key that is only eight bits in length. Then

$$d = (d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7) \text{ with } d_0 = 1.$$

Furthermore, suppose that we are certain that

$$d_0 d_1 d_2 d_3 \in \{1010, 1001\}.$$

Then we generate some number of random ciphertexts C_j, and for each we obtain the corresponding timing $T(C_j)$. We can emulate the first four steps of the repeated squaring algorithm for both

$$d_0 d_1 d_2 d_3 = 1010 \text{ and } d_0 d_1 d_2 d_3 = 1001$$

for each of these ciphertexts. For a given timing $T(C_j)$, let t_ℓ be the actual time taken in step ℓ for the squaring and multiplying steps of the repeated squaring algorithm. That is, t_ℓ includes the timing of $s = s^2 \pmod N$ and, if $d_\ell = 1$, it also includes $s = s \cdot C_j \pmod N$ (see the algorithm in Table 6.15). Also, let $\tilde{t}_\ell$ be the time obtained when emulating the square and multiply steps for an assumed private exponent bit ℓ. For $m > \ell$, define the shorthand notation

$$\tilde{t}_{\ell \ldots m} = \tilde{t}_\ell + \tilde{t}_{\ell+1} + \cdots + \tilde{t}_m.$$

Of course, $\tilde{t}_\ell$ depends on the precise bits emulated, but to simplify the notation we do not explicitly state this dependence (it should be clear from context).

Now suppose we select four ciphertexts, C_0, C_1, C_2, C_3, and we obtain the timing results in Table 6.16. In this example we see that for $d_0 d_1 d_2 d_3 = 1010$ we have a mean timing of

$$E(T(C_j) - \tilde{t}_{0 \ldots 3}) = (7 + 6 + 6 + 5)/4 = 6,$$

while the corresponding variance is

$$\operatorname{var}(T(C_j) - \tilde{t}_{0 \ldots 3}) = (1^2 + 0^2 + 0^2 + (-1)^2)/4 = 1/2.$$

On the other hand, for $d_0 d_1 d_2 d_3 = 1001$, we have

$$E(T(C_j) - \tilde{t}_{0 \ldots 3}) = 6,$$

but the variance is

$$\text{var}(T(C_j) - \tilde{t}_{0\ldots3}) = ((-1)^2 + 1^2 + (-1)^2 + 1^2)/4 = 1.$$

Although the mean is the same in both cases, Kocher's attack tells us that the smaller variance indicates that $d_0d_1d_2d_3 = 1010$ is the correct answer. But this begs the question of why we should observe a smaller variance in case of a correct guess for $d_0d_1d_2d_3$.

Table 6.16: Timings

		Emulate 1010		Emulate 1001	
j	$T(C_j)$	$\tilde{t}_{0\ldots3}$	$T(C_j) - \tilde{t}_{0\ldots3}$	$\tilde{t}_{0\ldots3}$	$T(C_j) - \tilde{t}_{0\ldots3}$
0	12	5	7	7	5
1	11	5	6	4	7
2	12	6	6	7	5
3	13	8	5	6	7

Consider $T(C_j)$, the timing of a particular computation $C_j^d \pmod{N}$ in Table 6.16. As above, for this $T(C_j)$, let $\tilde{t}_\ell$ be the emulated timing for the square and multiply steps corresponding to the ℓth bit of the private exponent. Also, let t_ℓ be the actual timing of the square and multiply steps corresponding to the ℓth bit of the private exponent. Let u include all timing not accounted for in the t_ℓ. The value u can be viewed as representing the measurement "error." In the example above, we assumed the private exponent d is eight bits, so for this case

$$T(C_j) = t_0 + t_1 + t_2 + \cdots + t_7 + u.$$

Now suppose that the high-order bits of d are $d_0d_1d_2d_3 = 1010$. Then for the timing $T(C_j)$ we have

$$\text{var}(T(C_j) - \tilde{t}_{0\ldots3}) = \text{var}(t_4) + \text{var}(t_5) + \text{var}(t_6) + \text{var}(t_7) + \text{var}(u),$$

since $\tilde{t}_\ell = t_\ell$, for $\ell = 0, 1, 2, 3$ and, consequently, there is no variance due to these emulated timings $\tilde{t}_\ell$. Note that here we are assuming the t_ℓ are independent and that the measurement error u is independent of the t_ℓ, which appear to be valid assumptions. If we denote the common variance of each t_ℓ by $\text{var}(t)$, we have

$$\text{var}(T(C_j) - \tilde{t}_{0\ldots3}) = 4\,\text{var}(t) + \text{var}(u).$$

However, if $d_0d_1d_2d_3 = 1010$, but we emulate $d_0d_1d_2d_3 = 1001$, then from the point of the first d_j that is in error, our emulation will fail, giving us essentially random timing results. In this case, the first emulation error occurs at d_2 so that we find

$$\begin{aligned}\operatorname{var}(T - \tilde{t}_{0\ldots3}) &= \operatorname{var}(t_2 - \tilde{t}_2) + \operatorname{var}(t_3 - \tilde{t}_3) + \operatorname{var}(t_4) + \operatorname{var}(t_5) \\ &\qquad + \operatorname{var}(t_6) + \operatorname{var}(t_7) + \operatorname{var}(u) \\ &\approx 6\operatorname{var}(t) + \operatorname{var}(u)\end{aligned}$$

since the emulated timings $\tilde{t}_2$ and $\tilde{t}_3$ can vary from the actual timings t_2 and t_3, respectively. That is, we see a larger variance when our guess for the private key bits is incorrect.

Although conceptually simple, Kocher's technique gives a powerful and practical approach to conducting a timing attack on an RSA implementation that uses repeated squaring (but not more advanced techniques). For the attack to succeed, the variance of the error term u must not vary too greatly between the different cases that are tested. Assuming that a simple repeated squaring algorithm is employed, this would almost certainly be the case since u only includes loop overhead and timing error. For more advanced modular exponentiation techniques, $\operatorname{var}(u)$ could differ greatly for different emulated bits, effectively masking the timing information needed to recover the bits of d.

The amount of data required for Kocher's attack (that is, the number of chosen decryptions that must be timed) depends on the error term u. However, the timings can be reused as bits of d are determined, since, given additional bits of d, only the emulation steps need to change. Therefore, the required number of timings is not nearly as daunting as it might appear at first blush. Again, this attack has been used to break real systems.

The major limitation to Kocher's attack is that it has only been successfully applied to RSA implementations that only use repeated squaring. Most RSA implementations also use various other techniques (Chinese Remainder Theorem, Montgomery multiplication, Karatsuba multiplication) to speed up the modular exponentiations. Only in highly resource-constrained environments (such as smartcards) is repeated squaring used without any of these other techniques.

In [167], Kocher argues that his timing attack should work for RSA implementations that employ techniques other than repeated squaring. However, Schindler [258] (among others) disputes this assertion. In any case, different timing techniques have been developed that succeed against more highly optimized RSA implementations. As previously noted, the RSA implementation in a recent version of OpenSSL was broken using a timing attack due to Brumley and Boneh [41].

The lesson of side channel attacks is an important one that extends far beyond the details of any particular attack. Side channels demonstrate that even if crypto is secure in theory, it may not be so in practice. That is, it's not sufficient to analyze a cipher in isolation—for a cipher to be considered secure in practice, it must be analyzed in the context of a specific implementation and the larger system in which it resides. Many of these factors don't directly relate to the mathematical properties of the cipher itself. Schneier has a good article that addresses some of these issues [262].

Side channel attacks nicely illustrate that attackers don't always play by the (presumed) rules. Attackers will try to exploit the weakest link in any security system. The best way to protect against such attacks is to think like an attacker and find these weak links before Trudy does.

6.7 Summary

In this chapter, we presented several advanced cryptanalytic attacks and techniques. We started with a classic World War II cipher, the Enigma, where the attack illustrated a "divide and conquer" approach. That is, an important component of the device (the stecker) could be split off from the rest of the cipher with devastating consequences. Then we considered a stream cipher attack, specifically, RC4 as used in WEP. This attack showed that even a strong cipher can be broken if used incorrectly.

In the block cipher realm, we discussed differential and linear cryptanalysis and these attacks were applied to TDES, a simplified version of DES. Some knowledge of these topics is necessary to understand the fundamental trade-offs in block cipher design.

Next, we presented a classic attack on the Merkle-Hellman knapsack public key cryptosystem. This attack nicely illustrates the impact that mathematical advances and clever algorithms can have on cryptography.

Side channel attacks have become important in recent years. It's crucial to be aware of such attacks, which go beyond the traditional concept of cryptanalysis, since they represent a real threat to otherwise secure ciphers. We discussed specific side channel attacks on RSA.

As usual, we've only scratched the surface in this chapter. Many other cryptanalytic attacks and techniques have been developed, and cryptanalysis remains an active area of research. The cryptanalytic attacks discussed here provide a reasonably representative sample of the methods that are used to attack and analyze ciphers.

6.8 Problems

1. In World War II, the German's usually used 10 cables on the stecker, only five different rotors were in general use, one reflector was in common use, and the reflector and five rotors were known to the Allies.

a. Under these restrictions, show that there are only about 2^{77} possible Enigma keys.

b. Show that if we ignore the stecker, under these restrictions there are fewer than 2^{30} settings.

2. Let $F(p)$, for $p = 0, 1, 2, \ldots, 13$, be the number of ways to plug p cables into the Enigma stecker. Show that

$$F(p) = \binom{26}{2p} \cdot (2p-1) \cdot (2p-3) \cdot \cdots \cdot 1.$$

3. Recall that for the Enigma attack described in Section 6.2.4, we found the cycles

$$S(\texttt{E}) = P_6 P_8 P_{13} S(\texttt{E})$$

and

$$S(\texttt{E}) = P_6 P_{14}^{-1} P_7 P_6^{-1} S(\texttt{E}).$$

Find two more independent cycles involving $S(\texttt{E})$ that can be obtained from the matched plaintext and ciphertext in Table 6.2.

4. How many pairs of cycles are required to uniquely determine the Enigma rotor settings?

5. In the text, we mentioned that the Enigma cipher is its own inverse.

a. Prove that the Enigma is its own inverse. Hint: Suppose that the ith plaintext letter is x, and that the corresponding ith ciphertext letter is y. This implies that when the ith letter typed into the keyboard is x, the letter y is illuminated on the lightboard. Show that for the same key settings, if the ith letter typed into the keyboard is y, then the letter x is illuminated on the lightboard.

b. What is the advantage of a cipher machine that is its own inverse (such as the Enigma), as compared to a cipher that is not (such as Purple and Sigaba)?

6. This problem deals with the Enigma cipher.

a. Show that a ciphertext letter cannot be the same as the corresponding plaintext letter.

b. Explain how the restriction in part a gives the cryptanalyst an advantage when searching for a crib.[7]

[7]In modern parlance, a crib is known as known plaintext.

7. Consider the Enigma attack discussed in the text and suppose that only cycles of $S(\texttt{E})$ are used to recover the correct rotor settings. Then, after the attack is completed, only the stecker value of $S(\texttt{E})$ is known. Using only the matched plaintext and ciphertext in Table 6.2, how many additional stecker values can be recovered?

8. Write a program to simulate the Enigma cipher. Use your program to answer the following questions, where the rotor and reflector permutations are known to be

$$\begin{aligned} R_\ell &= \texttt{EKMFLGDQVZNTOWYHXUSPAIBRCJ} \\ R_m &= \texttt{BDFHJLCPRTXVZNYEIWGAKMUSQO} \\ R_r &= \texttt{ESOVPZJAYQUIRHXLNFTGKDCMWB} \\ T &= \texttt{YRUHQSLDPXNGOKMIEBFZCWVJAT} \end{aligned}$$

where R_ℓ is the left rotor, R_m is the middle rotor, R_r is the right rotor, and T is the reflector. The "notch" that causes the odometer effect is at position Q for R_ℓ, V for R_m, and J for R_r. For example, the middle rotor steps when the right rotor steps from V to W.

   a. Recover the initial rotor settings given the following matched plaintext and ciphertext.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Plaintext	A	D	H	O	C	A	D	L	O	C	Q	U	I	D	P	R	O	Q	U	O	S	O
Ciphertext	S	W	Z	S	O	F	C	J	M	D	C	V	U	G	E	L	H	S	M	B	G	G
i	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43
Plaintext	L	I	T	T	L	E	T	I	M	E	S	O	M	U	C	H	T	O	K	N	O	W
Ciphertext	N	B	S	M	Q	T	Q	Z	I	Y	D	D	X	K	Y	N	E	W	J	K	Z	R

   b. Recover as many of the stecker settings as is possible from the known plaintext.

9. Suppose that the same Enigma rotors (in the same order) and reflector are used as in Problem 8, and the stecker has no cables connected. Solve for the initial rotor settings and recover the plaintext given the following ciphertext.

```
ERLORYROGGPBIMYNPRMHOUQYQETRQXTYUGGEZVBFPRIJGXRSSCJTXJBMW
JRRPKRHXYMVVYGNGYMHZURYEYYXTTHCNIRYTPVHABJLBLNUZATWXEMKRI
WWEZIZNBEOQDDDCJRZZTLRLGPIFYPHUSMBCAMNODVYSJWKTZEJCKPQYYN
ZQKKJRQQHXLFCHHFRKDHHRTYILGGXXVBLTMPGCTUWPAIXOZOPKMNRXPMO
AMSUTIFOWDFBNDNLWWLNRWMPWWGEZKJNH
```

Hint: The plaintext is English.

10. Develop a ciphertext-only attack on the Enigma, assuming that all you know about the plaintext is that it is English. Analyze the work factor of your proposed attack and also estimate the minimum amount of ciphertext necessary for your attack to succeed. Assume that Enigma rotors, the rotor order, the movable ring positions, and the reflector are all known. Then you need to solve for the initial settings of the three rotors and the stecker. Hint: Since `E` is the most common letter in English, guess that the plaintext is `EEEEEE`... and use this "noisy" plaintext to solve for the rotor and stecker settings.

11. Suggest modifications to the Enigma design that would make the attack discussed in Section 6.2 infeasible. Your objective is to make minor modifications to the design.

12. Consider a rotor with a hardwired permutation of $\{0, 1, 2, \ldots, n-1\}$. Denote this permutation as $P = (p_0, p_1, \ldots, p_{n-1})$, where P permutes i to p_i. Let d_i be the displacement of p_i, that is, $d_i = p_i - i \pmod{n}$. Find a formula for the elements of the kth rotor shift of P, which we denote P_k, where the shift is in the same direction as the rotors described in Section 6.2.3. Your formula must be in terms of p_i and d_i.

13. In the RC4 attack, suppose that 60 IVs of the form $(3, 255, V)$ are available. Empirically determine the probability that the key byte K_3 can be distinguished. What is the smallest number of IVs for which this probability is greater than 1/2?

14. In equations (6.7) and (6.9) we showed how to recover RC4 key bytes K_3 and K_4, respectively.
    a. Assuming that key bytes K_3 through K_{n-1} have been recovered, what is the desired form of the IVs that will be used to recover K_n?
    b. For K_n, what is the formula corresponding to (6.7) and (6.9)?

15. For the attack on RC4 discussed in Section 6.3, we showed that the probability that (6.7) holds is $(253/256)^{252}$. What is the probability that equation (6.9) holds? What is the probability that the corresponding equation holds for K_n?

16. In the discussion of the attack on RC4 keystream byte K_3 we showed that IVs of the form $(3, 255, V)$ are useful to the attacker. We also showed that IVs that are not of this form are sometimes useful to the attacker, and we gave the specific example of the $(2, 253, 0)$. Find an IV of yet another form that is useful in the attack on K_3.

17. The attack on RC4 discussed in this chapter illustrates that prepending an IV to a long-term key is insecure. In [112] it is shown that appending

the IV to the long-term key is also insecure. Suggest more secure ways to combine a long-term key with an IV for use as an RC4 key.

18. Suppose that Trudy has a ciphertext message that was encrypted with the RC4 cipher. Since RC4 is a stream cipher, the actual encryption formula is given by $c_i = p_i \oplus k_i$, where k_i is the ith byte of the keystream, p_i is the ith byte of the plaintext, c_i is the ith byte of the ciphertext. Suppose that Trudy knows the first ciphertext byte, and the first plaintext byte, that is, Trudy knows c_0 and p_0.

    a. Show that Trudy also knows the first byte of the keystream used to encrypt the message, that is, she knows k_0.

    b. Suppose that Trudy also happens to know that the first three bytes of the key are $(K_0, K_1, K_2) = (2, 253, 0)$. Show that Trudy can determine the next byte of the key, K_3, with a probability of success of about 0.05. Note that from part a, Trudy knows the first byte of the keystream. Hint: Suppose that the RC4 initialization algorithm were to stop after the $i = 3$ step. Write an equation that you could solve to determine the first byte of the key. Then show that this equation holds with a probability of about 0.05 when the entire 256-step initialization algorithm is used.

    c. If Trudy sees several messages encrypted with the same key that was used in part b, how can Trudy improve on the attack to recover K_3? That is, how can Trudy recover the key byte K_3 with a much higher probability of success (ideally, with certainty)?

    d. Assuming that the attack in part b (or part c) succeeds, and Trudy recovers K_3, extend the attack so that Trudy can recover K_4, with some reasonable probability of success. What is the probability that this step of the attack succeeds?

    e. Extend the attack in part d to recover the remaining key bytes, that is, $K_5, K_6, \ldots$. Show that this attack has essentially the same work factor regardless of the length of the key.

    f. Show that the attack in part a (and hence, the attack in parts a through e) also works if the first three key bytes are of the form $(K_0, K_1, K_2) = (3, 255, V)$ for any byte V.

    g. Why is this attack relevant to the (in)security of WEP?

19. The file `outDiff` (available at the textbook website) contains 100 chosen plaintext pairs P and $\tilde{P}$ that satisfy $P \oplus \tilde{P} = $ `0x0002`, along with the corresponding TDES-encrypted ciphertext pairs C and $\tilde{C}$. Use this information to determine the key bits $k_{13}, k_{14}, k_{15}, k_9, k_{10}, k_{11}$ using the differential cryptanalysis attack on TDES that is described in this chapter. Then use your knowledge of these key bits to exhaustively search

for the remaining key bits. Give the key as $K = k_0k_1k_2\cdots k_{15}$ in hexadecimal.

20. The file `outLin`, which is available at the textbook website, contains 100 known plaintext P along with the corresponding TDES-encrypted ciphertext C. Use this information to determine the value of $k_0 \oplus k_1$ and $k_2 \oplus k_7$ using the linear cryptanalysis attack on TDES that is described in this chapter. Then use your knowledge of these key bits to exhaustively search for the remaining key bits. Give the key in hexadecimal as $K = k_0k_1k_2\cdots k_{15}$.

21. Find a 16-bit key that encrypts

$$\text{plaintext} = \texttt{0x1223} = 0001001000100011$$

to

$$\text{ciphertext} = \texttt{0x5B0C} = 0101101100001100$$

using the cipher TDES.

22. Suppose that a DES-like cipher uses the S-box below.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	4	6	8	9	E	5	A	C	0	2	F	B	1	7	D	3
01	6	4	A	B	3	0	7	E	2	C	8	9	D	5	F	1
10	8	5	0	B	D	6	E	C	F	7	4	9	A	2	1	3
11	A	2	6	9	F	4	0	E	D	5	7	C	8	B	3	1

If the input to this S-box is 011101, what is the output? If inputs X_0 and X_1 yield outputs Y_0 and Y_1, respectively, and $X_0 \oplus X_1 = 000001$, what is the most likely value for $Y_0 \oplus Y_1$ and what is its probability?

23. Consider the S-box below. For the input $x_0x_1x_2$, the bit x_0 indexes the row, while x_1x_2 is the column index. We denote the output by y_0y_1.

	00	01	10	11
0	10	01	00	11
1	11	00	01	10

Find the best linear approximation to y_1 in terms of x_0, x_1, and x_2. With what probability does this approximation hold?

24. Construct a difference table analogous to that in Table 6.6 for S-box 1 of DES. The DES S-box 1 appears in Table 3.3 of Chapter 3. What is the most biased difference and what is the bias?

25. Construct a difference table analogous to that in Table 6.6 for the right S-box of TDES. Verify the results in equation (6.17). What is the second most biased difference, and what is the bias?

26. Construct a linear approximation table analogous to that in Table 6.7 for S-box 1 of DES. The DES S-box 1 appears in Table 3.3 of Chapter 3. Note that your table will have 64 rows and 15 columns. What is the best linear approximation, and how well does it approximate?

27. Construct a linear approximation table analogous to that in Table 6.7 for the left S-box of TDES. Verify the results in equation (6.29). What is the next best linear approximation and how well does it approximate?

28. Recall the linear cryptanalysis of TDES discussed in Section 6.4.6. Assume that equation (6.33) holds with probability $(3/4)^3 \approx 0.42$. Also assume that the key satisfies $k_0 \oplus k_1 = 0$. Then if we conduct the attack using 100 known plaintexts, what are the expected counts for $c_1 \oplus p_{10} = 0$ and $c_1 \oplus p_{10} = 1$? Compare your answer with the empirical results presented in the text. Why do you think the theoretical and empirical results differ?

29. Suppose that Bob's knapsack public key is

$$T = [168, 280, 560, 393, 171, 230, 684, 418].$$

Suppose that Alice encrypts a message with Bob's public key and the resulting ciphertext is $C_1 = 1135$. Implement the LLL attack and use your program to solve for the plaintext P_1. For the same public key, find the plaintext P_2 for the ciphertext $C_2 = 2055$. Can you determine the private key?

30. Suppose that Bob's knapsack public key is

$$T = [2195, 4390, 1318, 2197, 7467, 5716, 3974, 3996, 7551, 668].$$

Suppose that Alice encrypts a message with Bob's public key and the resulting ciphertext is $C_1 = 8155$. Implement the LLL attack and use your program to solve for the plaintext P_1. For the same public key, find the plaintext P_2 for the ciphertext $C_2 = 14748$. Can you determine the private key?

31. Consider the "simple" timing attack on RSA discussed in Section 6.6.1.

    a. Extend the timing attack to recover the bit d_2. That is, assuming that bit d_1 has been recovered, what conditions must Y and Z satisfy so that the attack presented in the text can be used to determine d_2?

b. Extend the attack to recover d_3, assuming that d_1 and d_2 have been recovered.

c. In practice, we need to recover about half of the private key bits. Why is this attack not a practical means for recovering such a large number of private key bits?

32. Suppose that in Kocher's timing attack, we obtain the timings $T(C_j)$ and the emulated timings $\tilde{t}_{0\ldots 2}$ for $d_0d_1d_2 \in \{100, 101, 110, 111\}$, as given in the table below.

		$\tilde{t}_{0\ldots 2}$			
j	$T(C_j)$	100	101	110	111
0	20	5	7	5	8
1	21	4	7	4	1
2	19	1	6	4	7
3	22	2	8	5	2
4	24	10	6	8	8
5	23	11	5	7	7
6	21	1	1	6	5
7	19	7	1	2	3

a. What is the most likely value of $d_0d_1d_2$ and why?

b. Why does this attack not succeed if CRT or Montgomery multiplication is used?

33. Write a program to recover the 64-bit key for STEA (Simplified TEA) given one known plaintext block and the corresponding ciphertext block. The STEA algorithm and a description of the attack on STEA can be found at [209].

34. If DES were a *group* [117], then given keys K_1 and K_2, there would exist a key K_3 such that

$$E(P, K_3) = E(E(P, K_1), K_2) \text{ for all plaintext } P, \tag{6.39}$$

and we could also find such a key K_3 if any of the encryptions were replaced by decryptions. If equation (6.39) holds, then triple DES is no more secure than single DES. It was established in [45] that DES is not a group and, consequently, triple DES is more secure than single DES. Show that TDES is not a group. Hint: Select TDES keys K_1 and K_2. You will be finished if you can verify that there does not exist any key K_3 for which $E(P, K_3) = E(E(P, K_1), K_2)$ for all possible choices of P.