Chapter 11

Making Public Keys: Math Tricks

The simple math trick used in most commercial public key cryptography is called a *modular inverse*. Cryptographers use modular inverses to manufacture public/private key pairs, such as those shown in Chapter 10. Here, you'll see how modular inverses force BlackHat to solve a difficult, time-consuming problem but give Alice a trapdoor to an easy, quick solution.

First, let's briefly review what led us here. Chapter 9 describes a system that enables Alice and Bob to exchange secret keys over a public line. Alice sends 1,000,000 secret keys, and Bob selects one of them. It frustrates BlackHat because he must try about 500,000 keys before stumbling onto the one Bob selected (see Figure 11-1).

Chapter 10 examines a major goal of public key cryptography: confidentiality. Alice openly (that is, publicly) distributes her public key to her clients so that each client can use the public key to encrypt a message to Alice. The message is confidential because only Alice's private key can decrypt a message encrypted with her matching public key (see Figure 11-2).

Compared with the Merkle puzzles in Chapter 9, this chapter shows a more math-based public key system. Now, instead of distributing 1,000,000 potential secret keys, Alice distributes a list of numbers. Her customers encode messages by choosing and adding numbers from the list. It's easy to choose numbers from the list and calculate their sum, but after they're totaled, it's far more difficult to figure out which numbers were added.

Easy and hard problems

Many math problems are easy to solve one way and far more time-consuming to solve another way. For example, without a calculator, squaring 1.234

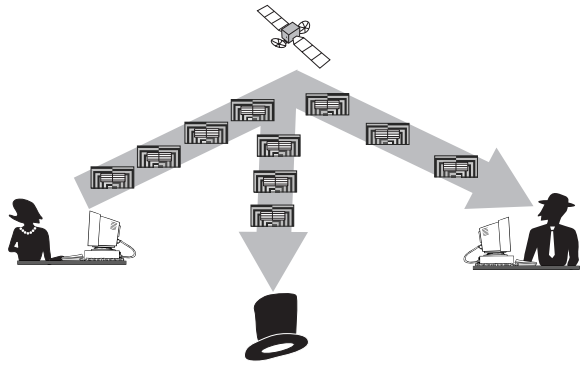


Figure 11-1 Alice sends 1,000,000 potential secret keys to Bob. BlackHat listens but doesn't know which one Bob selected.

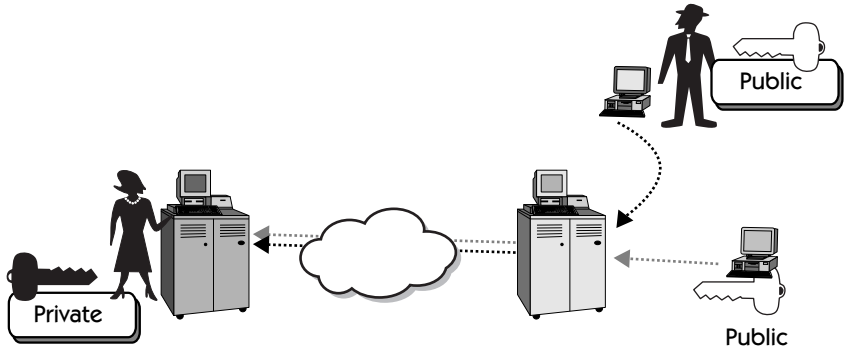


Figure 11-2 In a public/private key pair, senders have the encryption key. The receiver has the decryption key.

(1.234 x 1.234 = 1.522756) may take a little time. But it's easy compared with finding the square root of 1.522756. That is a far more time-consuming problem.

Alice's Easy Problem

Alice publishes a newsletter, and each issue contains stock reports requested by her clients. But her clients want only Alice to know which stocks they want to read about. So Alice sends them a list of stock names and stock numbers (shown in Figure 11-3) and gives them a way to communicate with her publicly so that only she can decipher the information they send over the Internet.

ALICE'S HOT PICKS

Select and Sum Stock Numbers

Name	Number
Amazon.com	1
Barnes & Noble	3
Ford Motor	5
General Motors	10
IBM	20
Microsoft	40

Figure 11-3 Alice sends her clients a list of stocks with special numbers that they can use to encipher the information.

Summing specially created stock numbers

Each stock number identifies a particular stock. Alice's clients sum the stock numbers and send the total to Alice. The stock numbers are designed so that every possible sum is unique. For example, if Bob sends 9, Alice knows that Bob must have added 1 (Amazon.com), 3 (Barnes & Noble), and 5 (Ford). Only one group of the stock numbers sums to 9. Similarly, only one combination sums to 4, one combination sums to 29, one combination sums to 41, and so on; each combination of stock numbers makes a unique sum.

Definition: super-increasing sequence

The uniqueness of each sum is guaranteed because Alice's choice of stock numbers—1, 3, 5, 10, 20, and 40—ensures it. It's a math trick. Here's how it works. Note that each stock number is greater than the sum of all the preceding numbers. That is, 5 is greater than 1 + 3, 10 is greater than 1 + 3 + 5, 20 is greater than 1 + 3 + 5 + 10, and 40 is greater than 1 + 3 + 5 + 10 + 20. Any such sequence of numbers is called a *super-increasing sequence*.

Alice chose a super-increasing sequence because it's easy and quick to figure out the individual numbers that make a sum. For example, it's easy to figure out that 53 is the sum of 40 + 10 + 3 or that 33 is the sum of 20 + 10 + 3. The math trick is to start with the biggest number in the sequence that is also less than the sum. That is, any sum must contain the greatest stock number that's less than the sum. Here's the math trick in action.

1. Bob sends Alice 53; 53 must be made with 40. 53 – 40 = 13. 13 must be made with 10. 13 – 10 = 3. So 53 = 40 + 10 + 3 or 53 = M + G + B.[1]
2. Casey sends Alice 33. 33 must be made with 20. 33 – 20 = 13. 13 must be made with 10. 13 – 10 = 3. So 33 = 20 + 10 + 3.

But because Alice's list in Figure 11-3 is public, BlackHat can intercept the transmission and figure out Bob's request, as shown in Figure 11-4.

1. We refer to each stock by the first letter of its name.

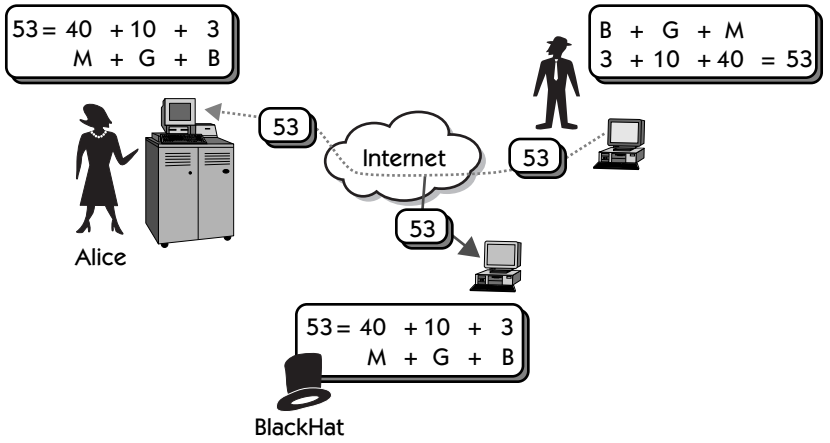


Figure 11-4 BlackHat intercepts Bob's message and figures out his stock requests.

Alice's problem is just like those that plague Merkle's puzzles in Chapter 9. Alice wants to solve quick and easy problems but needs to transform them into time-consuming and difficult problems for BlackHat. To do that, Alice uses modular inverses.

Although the simple cryptographic transformation technique we show in the remainder of this chapter is no longer used, it's instructive because almost all public key cryptography depends on modular inverse transformations.[2] If you don't need or want to know how these transformations work, feel free to skip the rest of this chapter. The rest of the book is not contingent on material in this chapter.

Before we turn to modular inverses, let's review some simple math you may not have used for many years.

Grade School Math Tricks

Unless you're a mathematician (or of that nature), you may not be familiar with the math tricks used in public key cryptography, but almost everyone who studies math learns a few math tricks.

Multiplying by 10, 100, 1,000, and so on is an easy trick we've all learned (see Figure 11-5). How about multiplying a number by 11—say, 24 x 11 (see

2. More precisely, most schemes depend on the discrete logarithm problem. For our discussion, understanding that difference is not important. For more details, see Appendix A.

Figure 11-6)? If you don't know the trick, you must do traditional multiplication. But there's a trick to use when multiplying a number by 11. First, you put 11 on top. (Most people put 11 on the bottom because it's easier when you're multiplying by 11 the conventional way.) For this math trick, instead of multiplying by 11 you add the digits in 24, 2 + 4, to get the middle number, 6. Multiplying by 11 is much faster and easier if you know this trick.[3]

There are thousands of creative math tricks, and public key cryptography uses some of them. No one wants to hide secrets with tricks (it doesn't sound very secure, does it?), so cryptographers call some of these math tricks by their formal mathematical name, *number theory*. Number theory can sound intimidating if you've never studied it; but if you think "math tricks" instead, it's more like figuring out a puzzle than getting a Ph.D.

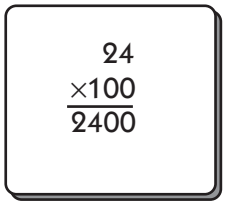


Figure 11-5 Multiplying by 100 is a simple math trick.

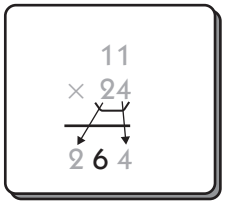


Figure 11-6 Here's a simple math trick for multiplying by 11.

More Grade School Math

Simple multiplicative inverses

All public key cryptographic systems depend on something we all learned in grade school math: the *identity* property.

In grade school we all learned that multiplying any number by 1 equals that number. It's called the multiplicative identity property: 6 x 1 = 6; 20 x 1 = 20; 1,234,567 x 1 = 1,234,567; and so forth. There's also an additive identity

3. 36 x 11 = 396, 71 x 11 = 781, and so on.

property: A number plus 0 equals that number. We focus here on multiplicative identity because public key cryptographers use it extensively.

We learned some tricks with the identity property. For example, because 8 x 1/8 equals 1, any number multiplied by 8 and then multiplied by 1/8 must be the same as multiplying that number by 1. For example, 2 x 8 = 16, and 16 x 1/8 = 2. We are back where we started, at 2. Simple math trick! We call 1/8 and 8 *inverses* because when we multiply them together they equal 1 (see Figures 11-7 and 11-8.)

Cryptographer's favorite math trick

Although our example is not complex enough for them, cryptographers like the math trick with inverses. Here's why. They reason, "If I can multiply the message by 1, the multiplicative identity property guarantees that the answer will equal the original message. And if I can separate that multiplication into two parts—such as 8 and 1/8—then I can disguise the original number (message) after multiplying by one part and come back to the original number (message) after multiplying by the second part."

Multiplicative Inverses:

$8 \times 1/8 = 1$

$3/2 \times 2/3 = 1$

Figure 11-7 A couple of multiplicative inverses.

Multiplicative Inverses and the Identity Property:

$2 \times 8 \times 1/8 = 2$

$98{,}766 \times 3/2 \times 2/3 = 98{,}766$

Figure 11-8 Multiplicative inverses and the identity property.

Simple public/ private key pair: 8 and 1/8

For example, if Alice were the only person on Earth who knew that 1/8 is the multiplicative inverse of 8, she could securely proclaim her public key to be "multiply by 8" (see Table 11-1, first key pair). If Bob wants to send her the message `5,000`, he encrypts it to `40,000` (5,000 x 8). Only Alice knows to multiply the ciphertext `40,000` by 1/8, so only she can quickly recover the plaintext. A slightly more complex key pair (3/2, 2/3) is also shown in Table 11-1.

Table 11-1 Two too-simple key pairs.

Key Pair	Message	Key	Encrypted	Key	Decrypted
8, 1/8	5,000	**8**	40,000	**1/8**	5,000
3/2, 2/3	98,766	**3/2**	148,149	**2/3**	98,766

Obviously, this encryption/decryption pattern is too simplistic to be secure in the real world of Internet- and math-savvy grade schoolers. Cryptographers needed something more complex, and sure enough, they pored over the problem until they found a way.

Division and Remainders: Modular Math

Because most public key cryptographic systems (including RSA, the most widely used) draw on modular math and modular math uses division, let's review some basics.

In grade school division, the answer is the quotient plus the remainder (see Figure 11-9); in modular math, the answer is the remainder only. Note that modular math uses only the whole number remainder and never fractions or negative numbers.

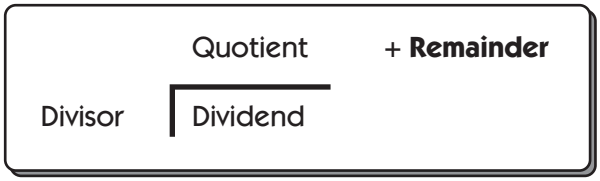


Figure 11-9 The terminology of division.

Clock math is modular math. Example: (10 + 4) mod 12 = 2

People actually do modulo operations every day. For example, a typical (nonmilitary) clock is modulo 12; 10 o'clock plus 4 hours is 2 o'clock. Because you've worked with clock math since grade school, you automatically perform a modular calculation: 10 + 2 gets you back to 0 (12 o'clock), and then you add 2 more. In modular math terms, it's (10 + 4) mod 12 = 2 (see Figure 11-10). Here's another clock math, or modulo, example: 3 o'clock + 25 hours = 4 o'clock, or (3 + 25) mod 12 = 4.

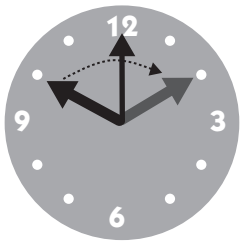
Figure 11-11 shows the similarities between grade school division and modular math. The chart illustrates how modular math uses division to calculate the remainder. The quotient, or the number of times the divisor goes into the dividend, isn't a part of the answer in modular math.

Modulo is usually abbreviated mod, as in 21 mod 10 = 1 (recall that 21 divided by 10 leaves a remainder of 1). The divisor is called the *modulus*, and the remainder is called the *residue*. We'll use the word *modulus* instead of *divisor*. Because the words *residue* and *remainder* are so close, we'll continue to refer to the residue as the remainder.

If you need additional modular math examples to solidify your understanding, see Figure 11-12. There's also an additional modulo lesson in Appendix A.

Because the last step in any modular math problem always finds the remainder, the multiplication table, shown in Figure 11-13, doesn't have a product greater than 10; after multiplication, the modulo operation reduces the product (see Figure 11-13). For example, in both grade school and modular math, 3 x 3 = 9. But grade school math and modulo have different answers for 3 x 4 because modulo 10 reduces 12 to 2. Modular math also reduces 3 x 5, 3 x 6, and so on.

This table also shows the kinds of modular multiplicative inverses that cryptographers use to build public and private keys; for example, 7 x 3 mod 10 = 1.



10 o'clock + 4 hours = 2 o'clock
(10 + 4) mod 12 = 2

Figure 11-10 Clock math uses modulo.

Quotient + **Remainder** (residue)

Divisor (modulus) | Dividend

Grade School Division

Dividend / divisor = quotient + remainder

				Answer		
Dividend	**/**	**Divisor**	**=**	**Quotient**	**+**	**Remainder**
13	/	10	=	1	+	3
23	/	10	=	2	+	3
33	/	10	=	3	+	3
43	/	10	=	4	+	3

Modular Arithmetic

Dividend / modulus = residue (remainder)

		Modulus		Answer		
Dividend		**(Divisor)**	**=**	**Quotient**	**+**	**Remainder**
13	mod	10	=	1	+	3
23	mod	10	=	2	+	3
33	mod	10	=	3	+	3
43	mod	10	=	4	+	3

Figure 11-11 Grade school division compared to modular math.

Same dividend (13) – different modulus	Different dividends – same modulus (10)	Random dividends and random modulus
13 mod 10 = 3	13 mod **10** = 3	13 mod 10 = 3
13 mod 11 = 2	21 mod **10** = 1	14 mod 10 = 4
13 mod 12 = 1	25 mod **10** = 5	14 mod 11 = 3
13 mod 13 = 0	32 mod **10** = 2	15 mod 11 = 4
13 mod 14 = 13	4,567 mod **10** = 7	15 mod 12 = 3
13 mod 15 = 13	1,247 mod **10** = 7	28 mod 12 = 4

If you're not familiar or comfortable with modular math, glance at some of these modulo examples to solidify your understanding. Because modular math is often easier than division, all these examples can be done in your head.

Recall 13 mod 14 = 13 because:

```
     0  remainder  13
14 ⌊13
```

Figure 11-12 Modular math examples.

Modulo Multiplication—
(example: Modulo **10**)

If product < **10**
"normal multiplication"

Example:
3 × 3 = 9 (normal)

If product equals or > **10**
after normal multiplication,
divide by 10 and use remainder

Examples:
3 × 3 = 9 (same as usual)
3 × 4 = 12 (different)
12 / 10 = 1 remainder 2

×	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	0	0	0	0	0	0	0
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	0	2	4	6	8
3	3	6	9	2	5	8	**1**	4	7
4	4	8	2	6	0	4	8	2	6
5	5	0	5	0	5	0	5	0	5
6	6	2	8	4	0	6	2	8	4
7	7	4	**1**	8	5	2	9	6	3
8	8	6	4	2	0	8	6	4	2
9	9	8	7	6	5	4	3	2	1

Figure 11-13 Multiplication modulo 10. The product is always less than 10 because modulo is about remainders and not quotients.

Modular Inverses

Multiplicative inverses are two numbers multiplied together that equal 1.

As you have seen, in grade school math the inverse of any whole number is easily computed; it's simply 1 divided by the integer. For example, the inverse of 8 is 1/8. Modular inverse pairs—two whole numbers multiplied together that equal 1—are the favorite modulo property of cryptographers. We'll see how Alice uses modular inverse pair numbers to force BlackHat to solve a time-consuming puzzle.

Figure 11-14 compares the simple inverse of 8 and the more difficult-to-determine inverse of 3 modulo 10. (Appendix A has more details on finding modular inverse pairs.)

Figure 11-15 shows a simple inverse you learned in grade school as well as a modular inverse. Multiplying any number (1, 2, 3, … 9) by 3 x 7 modulo 10 is the same as multiplying by 1; the original number doesn't change. If you like, instead of separately multiplying by 3 and then 7 you can multiply by 21 (3 x 7), as shown in Figure 11-16. The important point is that multiplying by a modular inverse pair is just like multiplying by 1.

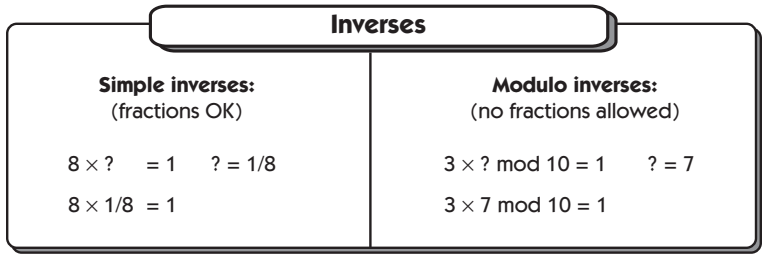


Figure 11-14 Simple and modulo inverses.

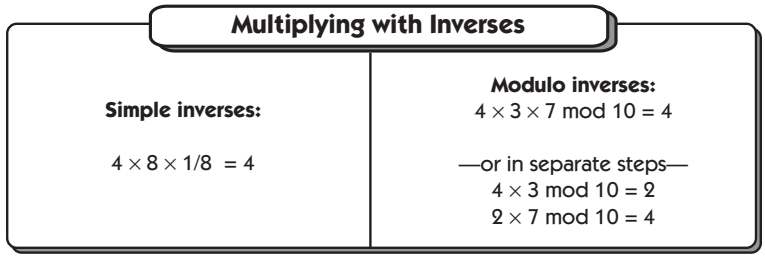


Figure 11-15 Multiplying using simple and modulo inverses.

Sequence	Multiplicative Identity x 21 (3 x 7)	Multiplicative Identity Mod 10 (restored)
0	0	0
1	21	1
3	63	3
5	105	5
6	126	6
7	147	7
8	168	8
9	189	9

Figure 11-16 Multiplying by a modular inverse pair.

Multiplying by a modular inverse pair is just like multiplying by 1.

This is the modular math trick that modern-day cryptographers put to good use. You can make modular inverse pairs that are much more difficult to find than inverses such as 3 and 7 mod 10. Because there are an infinite number of moduli and an infinite number of inverse pairs, cryptographers can choose a pair that, unless you know the secret generating numbers, will be very, very difficult to find.

Now let's separate the process into two parts to create public and private keys. We'll need to use a slightly larger modular inverse pair to see how Alice makes a public/private key pair—in particular, how Alice creates a public key inverse of her private key.

Real-world public key methods use moduli more than 200 digits long.

We used 21 (3 x 7) mod 10 because the numbers are easy to work with. When you use modulo 10, the answers are restricted to 0–9 because mod 10 means remainders after division by 10 must be less than 10. Because bigger moduli have more inverse pairs, cryptographers prefer much bigger numbers. For example, RSA uses moduli more than 200 digits long to make the search even more frustrating for someone who doesn't know the trick numbers. Fortunately, we don't need numbers that big to show how public/private key inverse pairs scramble and restore messages. A modulus of 101 is sufficient to illustrate public key math tricks and still is relatively easy to work with.

Figure 11-17 shows an example of another modular inverse pair (22 x 23 mod 101). If you multiply any number from 1 to 100 by 22 x 23 modulo 101, the result is the original number because 22 x 23 mod 101 = 1; so it's the same

Original Numbers	***22*23** (506)	**Mod 101**
0	0	**0**
1	506	**1**
3	1,518	**3**
5	2,530	**5**
10	5,060	**10**
20	10,120	**20**
40	20,240	**40**

Examples:

1 x 506 = 506. 506 / 101 = 5 R **1**

3 x 506 = 1,518. 1,518 / 101 = 15 R **3**

Figure 11-17 A multiplicative identity using modulo.

as multiplying by 1. (If you still find modulo a bit puzzling, see the section "Modulo Calculations" later in this chapter.)

It's important to see that 22 x 23 mod 101 works as a multiplicative identity just like 8 x 1/8 or 7 x 3 mod 10. Multiply any number by 506 and divide it by 101, and the remainder is the original number. It works every time for all numbers less than 101, the modulus.

By the way, you may have noticed that the column labeled "Original Numbers" in Figure 11-17 is Alice's super-increasing sequence from the beginning of the chapter. (We sneaked that in when you were concentrating on modular inverse pairs.)

Using Modular Inverses to Make a Public Key

Alice keeps the easy numbers for herself and makes difficult numbers for BlackHat.

Recall that Alice (or BlackHat) can quickly and easily figure out any sum made from the super-increasing number sequence (1, 3, 5, 10, 20, 40) first shown in Figure 11-3. These are the numbers she uses to solve her easy problem; it's her private key.

Here's how Alice uses her private key and modular inverses to create her public key. She simply multiplies each of her private key numbers by 23 mod 101, shown in Figure 11-18. That transforms the private key numbers 1, 3, 5, 10, 20, 40 into her public key 23, 69, 14, 28, 56, 11.

Private Key (super-increasing sequence)	*23	Public Key mod 101
1	23	**23**
3	69	**69**
5	115	**14**
10	230	**28**
20	460	**56**
40	920	**11**

Figure 11-18 Manufacturing a public key from a private key using modular inverses.

Putting It All Together

Now let's watch Bob use Alice's public key to encipher his stock picks. Even though BlackHat has a copy of Alice's public key, he can't quickly figure out which numbers Bob chose.

Giving BlackHat a Difficult, Time-Consuming Problem

Alice's private and public keys

Recall the example from the beginning of this chapter, shown again and labeled "Private Key" in Figure 11-19. The figure also includes Alice's newly minted public key, which Alice openly distributes. Her customers use it to request stock reports in the next newsletter. She keeps her private key private.

In the beginning of the chapter Alice openly distributed her super-increasing sequence (private key). Bob and BlackHat made copies. Bob summed `A` (1), `B` (3), and `F` (5) and sent the number `9` to Alice. Although Alice easily figures out that `9` means `A`, `B`, and `F`, so does BlackHat.

Bob makes the sum 106. BlackHat can't quickly figure out which numbers Bob used.

Now Alice openly distributes her public key and does not disclose her private key. Now when Bob selects stock A, B, and F he adds 23 + 69 + 14 and sends the sum `106` to Alice. Even though BlackHat has a copy of Alice's public key, he must spend a long time figuring out which numbers sum to `106`. Because BlackHat doesn't know and can't easily figure out the modular inverse pair Alice used to make her public key, she has successfully forced BlackHat to solve a time-consuming and difficult problem.

If you're not convinced that BlackHat's problem is much more difficult than Alice's, try to figure out which numbers sum to 103. You'll find the answer just before the "Review" section. How does Alice transform Bob's message, 106, to an easy problem?

Name	Private Key (not disclosed)	Public Key (openly distributed) select and sum stock numbers
Amazon.com	1	23
Barnes & Noble	3	69
Ford Motor	5	14
General Motors	10	28
IBM	20	56
Microsoft	40	11

Figure 11-19 Alice's public and private keys.

Trapdoor to the Easy Problem

Alice knows how to transform the difficult problem (finding which public key stock numbers sum to 106) into a much easier one (finding which private key stock numbers sum to 9). As shown in Figure 11-20, Alice multiplies 106 by 22 mod 101 and gets 9. She then easily figures out that 9 can be made only by summing 5, 3, and 1 (which are A, B, and F).

Why does this work? Recall that the modular inverse pair is 23 x 22 mod 101. Figure 11-21 shows that each number in Alice's public key is created by multiplying each number of her private key by 23 mod 101. Figure 11-21 also

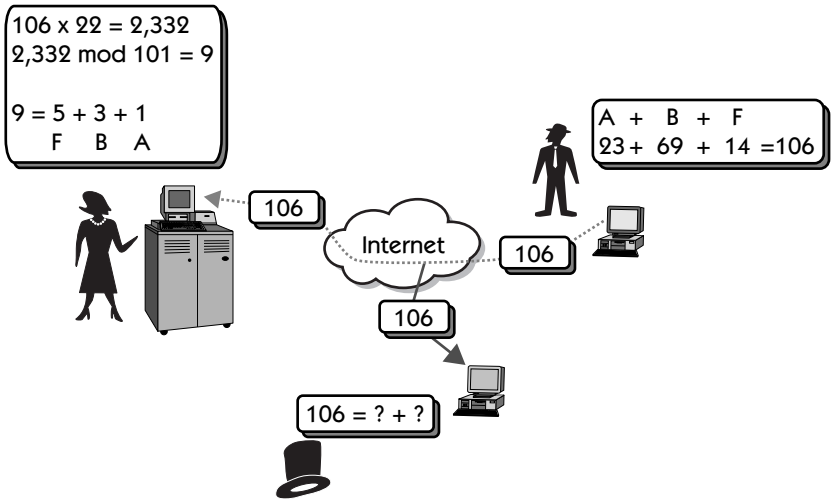


Figure 11-20 BlackHat intercepts but can't figure out Bob's stock requests.

Private Key	Public Key			Private Key
	(Column 1 x 23)	(Column 2 mod 101)	(Column 3 x 22)	(Column 4 mod 101)
1	23	23	506	1
3	69	69	1,518	3
5	115	14	308	5
10	230	28	616	10
20	460	56	1,232	20
40	920	11	242	40

Figure 11-21 Alice manufactures a public key from her private key, and then, to show it's reversible, she converts her public key back to her private key using her modulo inverse pair 23 x 22 mod 101.

shows she can convert each public key number back to its private key by multiplying by 22 mod 10, the other number of this particular modular inverse pair. Alice can use this method to convert public key numbers or sums of public key numbers to corresponding private key numbers, that is, 106 converts to 9.

Again, it's not important to remember the intricacies of this particular cryptographic system. But using modular inverses to effectively multiply by 1 is the cryptographic math trick behind most public key cryptographic systems. Appendix A shows how RSA creates public/private key pairs using modulo inverses.

Knapsack Cryptography

Alice's simplified public key system presented here is based on the first commercial public key system to offer confidentiality. Called the *knapsack*, it was invented by Ralph Merkle and Martin Hellman. The knapsack is no longer a secure system because its design was based on a telltale mathematical pattern that eventually gave it away. The knapsack's weakness, discovered by Adi Shamir (the *S* in RSA), is not in the use of modular inverses but rather is in an underlying part of the knapsack's private key. In fact, modular inverse pairs are used more in RSA than in the knapsack. RSA, probably the most widely used public key cryptographic system, is the subject of Chapter 12.

Modulo Calculations

As promised, all the intermediate calculations of Figure 11-18 are shown in Figure 11-22.

Exercise: Find Which Numbers Sum to 103

Convert the public key sum (103) to the private key sum (103 x 22 mod 101 = 44). Then find which private key numbers sum to 44. The number 44 is uniquely made by summing 40, 3, and 1, which correspond to M, B, and A, respectively. So 103 must be made by adding 23 (A), 69 (B), and 11 (M); see Table 11-2.

Table 11-2 After Alice converts 103 to 44, she knows that Bob picked stocks A, B, and M.

	A	B	F	G	I	M	
public	23	69	14	28	56	11	23 + 69 + 11= 103
private	1	3	5	10	20	40	1 + 3 + 40 = 44

Original Numbers	Modulo Multiplicative Identity *22*23 (506)	Quotient	& Calculations		Mod 101
0	0	0	*101 =	0	**0**
1	506	5	*101 =	505	**1**
3	1,518	15	*101 =	1,515	**3**
5	2,530	25	*101 =	2,525	**5**
10	5,060	50	*101 =	5,050	**10**
20	10,120	100	*101 =	10,100	**20**
40	20,240	200	*101 =	20,200	**40**

The second column shows the number multiplied by 506 (=22*23), (e.g., 1 x 506 = 506).
The quotient column shows how many times that number is divisible by 101, (e.g., 506 / 101 = 5).
The calculations columns show how the remainder is figured, (e.g., 506 – 505 = 1).

Examples:
1 x 506 = 506 506 = 5 * 101 + 1
3 x 506 = 1,518 1,518 = 15 * 101 + 3

Figure 11-22 Detailed intermediate calculations of Figure 11-18. The numbers demonstrate that 22 x 23 mod 101 is like multiplying by 1.

Review

Public key cryptography, although complex, is based on the simple mathematical concept of multiplicative inverses. Multiplicative inverses are two numbers that when multiplied together equal 1 (e.g., 8 and 1/8). In modular mathematics, two whole numbers are inverses if, when they are multiplied together, the answer is 1 (e.g., 7 x 3 mod 10 = 1).

Cryptographers use modular inverse pairs to help create secure public and private keys. Multiplying a message by one of the inverse pairs scrambles (encrypts) it, and multiplying the encrypted message by the other inverse pair recovers (decrypts) it. For example, suppose the message is 4. To scramble it, you use 4 x 7 mod 10 = 8. To recover it, you use 8 x 3 mod 10 = 4.

Almost all public key systems base their security on the difficulty of performing inverse calculations. By using simple math tricks in very complex ways, cryptographers have built public key systems that allow Alice to create and openly distribute her public key—without compromising the security of her private key or messages encrypted with her public key.