# Chapter 2

# Cryptography and Number Theory

## 2.1 Cryptography and Modular Arithmetic

### Introduction to Cryptography

For thousands of years people have searched for ways to send messages secretly. There is a story that, in ancient times, a king needed to send a secret message to his general in battle. The king took a servant, shaved his head, and wrote the message on his head. He waited for the servant's hair to grow back and then sent the servant to the general. The general then shaved the servant's head and read the message. If the enemy had captured the servant, they presumably would not have known to shave his head, and the message would have been safe.

*Cryptography* is the study of methods to send and receive secret messages. In general, we have a *sender* who is trying to send a message to a *receiver*. There is also an *adversary*, who wants to steal the message. We are successful if the sender is able to communicate a message to the receiver without the adversary learning what that message was.

Cryptography has remained important over the centuries, used mainly for military and diplomatic communications. Recently, with the advent of the internet and electronic commerce, cryptography has become vital for the functioning of the global economy, and is something that is used by millions of people on a daily basis. Sensitive information such as bank records, credit card reports, passwords, or private communication, is (and should be) *encrypted*—modified in such a way that, hopefully, it is only understandable to people who should be allowed to have access to it, and *undecipherable* to others.

Undecipherability by an adversary is, of course, a difficult goal. No code is completely undecipherable. If there is a printed "codebook," then the adversary can always steal the codebook, and no amount of mathematical sophistication can prevent this possibility. More likely, an adversary may have extremely large amounts of computing power and human resources to devote to trying to crack a code. Thus our notion of security is tied to computing power—a code is only as safe as the amount of computing power needed to break it. If we design codes that seem to need exceptionally large amounts of computing power to break, then we can be relatively confident in their security.

### Private Key Cryptography

Traditional cryptography is known as *private key cryptography*. The sender and receiver agree in advance on a *secret code*, and then send messages using that code. For example, one of the oldest codes is known as a *Caesar cipher*. In this code, the letters of the alphabet are shifted by some fixed amount. Typically, we call the original message the *plaintext* and the encoded text the *ciphertext*. An example of a Caesar cipher would be the following code:

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

Thus if we wanted to send the plaintext message

ONE IF BY LAND AND TWO IF BY SEA ,

we would send the ciphertext

SRI MJ FC PERH ERH XAS MJ FC WIE .

A Caeser cipher is especially easy to implement on a computer using a scheme known as arithmetic mod 26. The symbolism

$$m \bmod n$$

means the remainder we get when we divide $m$ by $n$. A bit more precisely we can give the following definition.

**Definition 2.1** *For integers $m$ and $n$, $m \bmod n$ is the smallest nonnegative integer $r$ such that*

$$m = nq + r \tag{2.1}$$

*for some integer $q$.*

We will refer to the fact that $m \bmod n$ is always well defined as Euclid's division theorem. The proof appears in the next section.[1]

**Theorem 2.1 (Euclid's division theorem)** *For every integer $m$ and positive integer $n$, there exist unique integers $q$ and $r$ such that $m = nq + r$ and $0 \leq r < n$.*

**Exercise 2.1-1** Use The definition of $m \bmod n$ to compute $10 \bmod 7$ and $-10 \bmod 7$. What are $q$ and $r$ in each case? Does $(-m) \bmod n = -(m \bmod n)$?

[1] In an unfortunate historical evolution of terminology, the fact that for every nonnegative integer $m$ and positive integer $n$, there exist unique nonnegative integers $q$ and $r$ such that $m = nq + r$ and $r < n$ is called "Euclid's algorithm." In modern language we would call this "Euclid's Theorem" instead. While it seems obvious that there is such a smallest nonnegative integer $r$ and that there is exactly one such pair $q$, $r$ with $r < n$, a technically complete study would derive these facts from the basic axioms of number theory, just as "obvious" facts of geometry are derived from the basic axioms of geometry. The reasons why mathematicians take the time to derive such obvious facts from basic axioms is so that everyone can understand exactly what we are assuming as the foundations of our subject; as the "rules of the game" in effect.

**Exercise 2.1-2** Using 0 for A, 1 for B, and so on, let the numbers from 0 to 25 stand for the letters of the alphabet. In this way, convert a message to a sequence of strings of numbers. For example SEA becomes 18 4 0. What does (the numerical representation of) this word become if we shift every letter two places to the right? What if we shift every letter 13 places to the right? How can you use the idea of $m$ mod $n$ to implement a Caeser cipher?

**Exercise 2.1-3** Have someone use a Caeser cipher to encode a message of a few words in your favorite natural language, without telling you how far they are shifting the letters of the alphabet. How can you figure out what the message is? Is this something a computer could do quickly?

In Exercise 2.1-1, $10 = 7(1)+3$ and so 10 mod 7 is 3, while $-10 = 7(-2)+4$ and so $-10$ mod 7 is 4. These two calculations show that $(-m)$ mod $n = -(m \text{ mod } n)$ is not necessarily true. Note that $-3$ mod 7 is 4 also. Furthermore, $-10 + 3$ mod $7 = 0$, suggesting that $-10$ is essentially the same as $-3$ when we are considering integers mod 7.

In Exercise 2.1-2, to shift each letter two places to the right, we replace each number $n$ in our message by $(n+2)$ mod 26, so that SEA becomes 20 8 2. To shift 13 places to the right, we replace each number $n$ in our message with $(n + 13)$ mod 26 so that SEA becomes 5 17 13. Similarly to implement a shift of $s$ places, we replace each number $n$ in our message by $(n + s)$ mod 26. Since most computer languages give us simple ways to keep track of strings of numbers and a "mod function," it is easy to implement a Caeser cipher on a computer.

Exercise 2.1-3 considers the complexity of encoding, decoding and cracking a Ceasar cipher. Even by hand, it is easy for the sender to encode the message, and for the receiver to decode the message. The disadvantage of this scheme is that it is also easy for the adversary to just try the 26 different possible Caesar ciphers and decode the message. (It is very likely that only one will decode into plain English.) Of course, there is no reason to use such a simple code; we can use any arbitrary permutation of the alphabet as the ciphertext, e.g.

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

If we encode a short message with a code like this, it would be hard for the adversary to decode it. However, with a message of any reasonable length (greater than about 50 letters), an adversary with a knowledge of the statistics of the English language can easily crack the code. (These codes appear in many newspapers and puzzle books under the name cryptograms. Many people are able to solve these puzzles, which is compelling evidence of the lack of security in such a code.)

We do not have to use simple mappings of letters to letters. For example, our coding algorithm can be to

- take three consecutive letters,
- reverse their order,
- interpret each as a base 26 integer (with A=0; B=1, etc.),
- multiply that number by 37,

- add 95 and then
- convert that number to base 8.

We continue this processing with each block of three consecutive letters. We append the blocks, using either an 8 or a 9 to separate the blocks. When we are done, we reverse the number, and replace each digit 5 by two 5's. Here is an example of this method:

```
plaintext: ONEIFBYLANDTWOIFBYSEA

block and reverse: ENO  BFI   ALY     TDN   IOW     YBF    AES
base 26 integer:  3056  814   310    12935  5794   16255   122
*37 +95 base 8: 335017 73005 26455 1646742 642711 2226672 11001
appended      : 335017873005926455916467429642711822266728110 01
reverse, 5rep : 10011827662228117246924764619555546295500378710533
```

As Problem 20 shows, a receiver who knows the code can decode this message. Furthermore, a casual reader of the message, without knowledge of the encryption algorithm, would have no hope of decoding the message. So it seems that with a complicated enough code, we can have secure cryptography. Unfortunately, there are at least two flaws with this method. The first is that if the adversary learns, somehow, what the code is, then she can easily decode it. Second, if this coding scheme is repeated often enough, and if the adversary has enough time, money and computing power, this code could be broken. In the field of cryptography, some entities have all these resources (such as a government, or a large corporation). The infamous German Enigma code is an example of a much more complicated coding scheme, yet it was broken and this helped the Allies win World War II. (The reader might be interested in looking up more details on this; it helped a lot in breaking the code to have a stolen Enigma machine, though even with the stolen machine, it was not easy to break the code.) In general, any scheme that uses a *codebook*, a secretly agreed upon (possibly complicated) code, suffers from these drawbacks.

### Public-key Cryptosystems

A *public-key* cryptosystem overcomes the problems associated with using a codebook. In a public-key cryptosystem, the sender and receiver (often called *Alice* and *Bob* respectively) don't have to agree in advance on a secret code. In fact, they each publish part of their code in a public directory. Further, an adversary with access to the encoded message and the public directory still cannot decode the message.

More precisely, Alice and Bob will each have two keys, a *public key* and a *secret key*. We will denote Alice's public and secret keys as $KP_A$ and $KS_A$ and Bob's as $KP_B$ and $KS_B$. They each keep their secret keys to themselves, but can publish their public keys and make them available to anyone, including the adversary. While the key published is likely to be a symbol string of some sort, the key is used in some standardized way (we shall see examples soon) to create a function from the set $\mathcal{D}$ of possible messages onto itself. (In complicated cases, the key might be the actual function). We denote the functions associated with $KS_A$, $KP_A$, $KS_B$ and $KP_B$ by

$S_A$, $P_A$, $S_B$, and $P_B$, respectively. We require that the public and secret keys are chosen so that the corresponding functions are inverses of each other, i.e for any message $M \in \mathcal{D}$ we have that

$$M = S_A(P_A(M)) = P_A(S_A(M)), \text{ and} \quad (2.2)$$
$$M = S_B(P_B(M)) = P_B(S_B(M)). \quad (2.3)$$

We also assume that, for Alice, $S_A$ and $P_A$ are easily computable. However, it is essential that *for everyone except Alice, $S_A$ is hard to compute, even if you know $P_A$.* At first glance, this may seem to be an impossible task, Alice creates a function $P_A$, that is public and easy to compute for everyone, yet this function has an inverse, $S_A$, that is hard to compute for everyone except Alice. It is not at all clear how to design such a function. In fact, when the idea for public key cryptography was proposed (by Diffie and Hellman[2]), no one knew of any such functions. The first complete public-key cryptosystem is the now-famous RSA cryptosystem, widely used in many contexts. To understand how such a cryptosystem is possible requires some knowledge of number theory and computational complexity. We will develop the necessary number theory in the next few sections.

Before doing so, let us just assume that we have such a function and see how we can make use of it. If Alice wants to send Bob a message $M$, she takes the following two steps:

1. Alice obtains Bob's public key $P_B$.
2. Alice applies Bob's public key to $M$ to create ciphertext $C = P_B(M)$.

Alice then sends $C$ to Bob. Bob can decode the message by using his secret key to compute $S_B(C)$ which is identical to $S_B(P_B(M))$, which by (2.3) is identical to $M$, the original message. The beauty of the scheme is that even if the adversary has $C$ and knows $P_B$, she cannot decode the message without $S_B$, since $S_B$ is a secret that only Bob has. Even though the adversary knows that $S_B$ is the inverse of $P_B$, the adversary cannot easily compute this inverse.

Since it is difficult, at this point, to describe an example of a public key cryptosystem that is hard to decode, we will give an example of one that is easy to decode. Imagine that our messages are numbers in the range 1 to 999. Then we can imagine that Bob's public key yields the function $P_B$ given by $P_B(M) = rev(1000 - M)$, where $rev()$ is a function that reverses the digits of a number. So to encrypt the message 167, Alice would compute $1000 - 167 = 833$ and then reverse the digits and send Bob $C = 338$. In this case $S_B(C) = 1000 - rev(C)$, and Bob can easily decode. This code is not secure, since if you know $P_B$, you can figure out $S_B$. The challenge is to design a function $P_B$ so that even if you know $P_B$ and $C = P_B(M)$, it is exceptionally difficult to figure out what $M$ is.

### Arithmetic modulo $n$

The RSA encryption scheme is built upon the idea of arithmetic mod $n$, so we introduce this arithmetic now. Our goal is to understand how the basic arithmetic operations, addition, subtraction, multiplication, division, and exponentiation behave when all arithmetic is done mod $n$. As we shall see, some of the operations, such as addition, subtraction and multiplication, are straightforward to understand. Others, such as division and exponentiation, behave very differently than they do for normal arithmetic.

[2]Whitfield Diffie and Martin Hellman. "New directions in cryptography" *IEEE Transactions on Information Theory*, **IT-22**(6) pp 644-654, 1976.

**Exercise 2.1-4** Compute $21 \bmod 9$, $38 \bmod 9$, $(21 \cdot 38) \bmod 9$, $(21 \bmod 9) \cdot (38 \bmod 9)$, $(21 + 38) \bmod 9$, $(21 \bmod 9) + (38 \bmod 9)$.

**Exercise 2.1-5** True or false: $i \bmod n = (i + 2n) \bmod n$; $i \bmod n = (i - 3n) \bmod n$

In Exercise 2.1-4, the point to notice is that

$$(21 \cdot 38) \bmod 9 = (21 \bmod 9)(38 \bmod 9)$$

and

$$(21 + 38) \bmod 9 = (21 \bmod 9) + (38 \bmod 9).$$

These equations are very suggestive, though the general equations that they first suggest aren't true! As we shall soon see, some closely related equations are true.

Exercise 2.1-5 is true in both cases, as adding multiples of $n$ to $i$ does not change the value of $i \bmod n$. In general, we have

**Lemma 2.2** *$i \bmod n = (i + kn) \bmod n$ for any integer $k$.*

**Proof:** By Theorem 2.1, for unique integers $q$ and $r$, with $0 \leq r < n$, we have

$$i = nq + r. \tag{2.4}$$

Adding $kn$ to both sides of Equation 2.4, we obtain

$$i + kn = n(q + k) + r. \tag{2.5}$$

Applying the definition of $i \bmod n$ to Equation 2.4, we have that $r = i \bmod n$ and applying the same definition to Equation 2.5 we have that $r = (i + kn) \bmod n$. The lemma follows. ■

Now we can go back to the equations of Exercise 2.1-4; the correct versions are stated below. Informally, we are showing if we have a computation involving addition and multiplication, and we plan to take the end result mod $n$, then we are free to take any of the intermediate results mod $n$ also.

**Lemma 2.3**

$$\begin{aligned}(i + j) \bmod n &= [i + (j \bmod n)] \bmod n \\ &= [(i \bmod n) + j] \bmod n \\ &= [(i \bmod n) + (j \bmod n)] \bmod n\end{aligned}$$

$$\begin{aligned}(i \cdot j) \bmod n &= [i \cdot (j \bmod n)] \bmod n \\ &= [(i \bmod n) \cdot j] \bmod n \\ &= [(i \bmod n) \cdot (j \bmod n)] \bmod n\end{aligned}$$

**Proof:** We prove the first and last terms in the sequence of equations for plus are equal; the other equalities for plus follow by similar computations. The proofs of the equalities for products are similar.

By Theorem 2.1, we have that for unique integers $q_1$ and $q_2$,

$$i = (i \bmod n) + q_1 n \text{ and } j = (j \bmod n) + q_2 n.$$

Then adding these two equations together mod $n$, and using Lemma 2.2, we obtain

$$\begin{aligned}(i+j) \bmod n &= [(i \bmod n) + q_1 n + (j \bmod n) + q_2 n)] \bmod n \\ &= [(i \bmod n) + (j \bmod n) + n(q_1 + q_2)] \bmod n \\ &= [(i \bmod n) + (j \bmod n)] \bmod n.\end{aligned}$$

■

We now introduce a convenient notation for performing modular arithmetic. We will use the notation $Z_n$ to represent the integers $0, 1, \ldots, n-1$ together with a redefinition of addition, which we denote by $+_n$, and a redefinition of multiplication, which we denote $\cdot_n$. The redefinitions are:

$$i +_n j = (i + j) \bmod n \tag{2.6}$$

$$i \cdot_n j = (i \cdot j) \bmod n \tag{2.7}$$

We will use the expression "$x \in Z_n$" to mean that $x$ is a variable that can take on any of the integral values between 0 and $n-1$. In addition, $x \in Z_n$ is a signal that if we do algebraic operations with $x$, we are will use $+_n$ and $\cdot_n$ rather than the usual addition and multiplication. In ordinary algebra it is traditional to use letters near the beginning of the alphabet to stand for constants; that is, numbers that are fixed throughout our problem and would be known in advance in any one instance of that problem. This allows us to describe the solution to many different variations of a problem all at once. Thus we might say "For all integers $a$ and $b$, there is one and only one integer $x$ that is a solution to the equation $a + x = b$, namely $x = b - a$." We adopt the same system for $Z_n$. When we say "Let $a$ be a member of $Z_n$," we mean the same thing as "Let $a$ be an integer between 0 and $n-1$," but we are also signaling that in equations involving $a$, we will use $+_n$ and $\cdot_n$.

We call these new operations addition mod $n$ and multiplication mod $n$. We must now verify that all the "usual" rules of arithmetic that normally apply to addition and multiplication still apply with $+_n$ and $\cdot_n$. In particular, we wish to verify the commutative, associative and distributive laws.

**Theorem 2.4** *Addition and multiplication mod n satisfy the commutative and associative laws, and multiplication distributes over addition.*

**Proof:** Commutativity follows immediately from the definition and the commutativity of ordinary addition and multiplication. We prove the associative law for addition in the following equations; the other laws follow similarly.

$$\begin{aligned}a +_n (b +_n c) &= (a + (b +_n c)) \bmod n && \text{(Equation 2.6)} \\ &= (a + ((b + c) \bmod n)) \bmod n && \text{(Equation 2.6)}\end{aligned}$$

$$
\begin{aligned}
&= (a + (b + c)) \bmod n && \text{(Lemma 2.3)}\\
&= ((a + b) + c) \bmod n && \text{(Associative law for ordinary sums)}\\
&= ((a + b) \bmod n + c) \bmod n && \text{(Lemma 2.3)}\\
&= ((a +_n b) + c) \bmod n && \text{(Equation 2.6)}\\
&= (a +_n b) +_n c && \text{(Equation 2.6).}
\end{aligned}
$$

■

Notice that $0 +_n i = i$, $1 \cdot_n i = i$, (these equations are called the *additive identity properties* and the *multiplicative identity* properties) and $0 \cdot_n i = 0$, so we can use 0 and 1 in algebraic expressions in $Z_n$ (which we may also refer to as agebraic expressions mod $n$) as we use them in ordinary algebraic expressions. We use $a -_n b$ to stand for $a +_n (-b)$.

We conclude this section by observing that repeated applications of Lemma 2.3 and Theorem 2.4 are useful when computing sums or products mod $n$ in which the numbers are large. For example, suppose you had $m$ integers $x_1, \ldots, x_m$ and you wanted to compute $(\sum_{j=1}^{m} x_i) \bmod n$. One natural way to do so would be to compute the sum, and take the result modulo $n$. However, it is possible that, on the computer that you are using, even though $(\sum_{j=1}^{m} x_i) \bmod n$ is a number that can be stored in an integer, and each $x_i$ can be stored in an integer, $\sum_{j=1}^{m} x_i$ might be too large to be stored in an integer. (Recall that integers are typically stored as 4 or 8 bytes, and thus have a maximum value of roughly $2 \times 10^9$ or $9 \times 10^{18}$.) Lemma 2.3 tells us that if we are computing a result mod $n$, we may do all our calculations in $Z_n$ using $+_n$ and $\cdot_n$, and thus never computing an integer that has significantly more digits than any of the numbers we are working with.

### Cryptography using addition mod $n$

One natural way to use addition of a number $a$ mod $n$ in encryption is first to convert the message to a sequence of digits—say concatenating all the ASCII codes for all the symbols in the message—and then simply add $a$ to the message mod $n$. Thus $P(M) = M +_n a$ and $S(C) = C +_n (-a) = C -_n a$. If $n$ happens to be larger than the message in numerical value, then it is simple for someone who knows $a$ to decode the encrypted message. However an adversary who sees the encrypted message has no special knowledge and so unless $a$ was ill chosen (for example having all or most of the digits be zero would be a silly choice) the adversary who knows what system you are using, even including the value of $n$, but does not know $a$, is essentially reduced to trying all possible $a$ values. (In effect adding $a$ appears to the adversary much like changing digits at random.) Because you use $a$ only once, there is virtually no way for the adversary to collect any data that will aid in guessing $a$. Thus, if only you and your intended recipient know $a$, this kind of encryption is quite secure: guessing $a$ is just as hard as guessing the message.

It is possible that once $n$ has been chosen, you will find you have a message which translates to a larger number than $n$. Normally you would then break the message into segments, each with no more digits than $n$, and send the segments individually. It might seem that as long as you were not sending a large number of segments, it would still be quite difficult for your adversary to guess $a$ by observing the encrypted information. However if your adversary knew $n$ but not $a$ and knew you were adding $a$ mod $n$, he or she could take two messages and subtract them in $Z_n$, thus getting the difference of two unencrypted messages. (In Problem 13 we ask you to explain why, even if your adversary didn't know $n$, but just believed you were adding some secret

number $a$ mod some other secret number $n$, she or he could use three encoded messages to find three differences in the integers, instead of $Z_n$, one of which was the difference of two messages.) This difference could contain valuable information for your adversary.[3] And if your adversary could trick you into sending just one message $z$ that he or she knows, intercepting the message and subtracting $z$ would give your adversary $a$. Thus adding $a$ mod $n$ is not an encoding method you would want to use more than once.

### Cryptography using multiplication mod $n$

We will now explore whether multiplication is a good method for encryption. In particular, we could encrypt by multiplying a message (mod $n$) by a prechosen value $a$. We would then expect to decrypt by "dividing" by $a$. What exactly does division mod $a$ mean? Informally, we think of division as the "inverse" of multiplication, that is, if we take a number $x$, multiply by $a$ and then divide by $a$, we should get back $x$. Clearly, with normal arithmetic, this is the case. However, with modular arithmetic, division is trickier.

**Exercise 2.1-6** One possibility for encryption is to take a message $x$ and compute $a \cdot_n x$, for some value $a$, that the sender and receiver both know. You could then decrypt by doing division by $a$ in $Z_n$ if you knew how to divide in $Z_n$. How well does this work? In particular, consider the following three cases. First, consider $n = 12$ and $a = 4$ and $x = 3$. Second, consider $n = 12$ and $a = 3$ and $x = 6$. Third, consider $n = 12$ and $a = 5$ and $x = 7$. In each case, ask if your recipient, knowing $a$, could figure out what the message $x$ is.

When we encoded a message by adding $a$ in $Z_n$, we could decode the message simply by subtracting $a$ in $Z_n$. However, this method had significant disadvantages, even if our adversary did not know $n$. Suppose that instead of encoding by adding $a$ mod $n$, we encoded by multiplying by $a$ mod $n$. (This doesn't give us a great secret key cryptosystem, but it illustrates some key points.) By analogy, if we encode by multiplying by $a$ in $Z_n$, we would expect to decode by dividing by $a$ in $Z_n$. However, Exercise 2.1-6 shows that division in $Z_n$ doesn't always make very much sense. Suppose your value of $n$ was 12 and the value of $a$ was 4. You send the message 3 as $4 \cdot_{12} 3 = 0$. Thus you send the encoded message 0. Now your recipient sees 0, and says the message might have been 0; after all, $4 \cdot_{12} 0 = 0$. On the other hand, $4 \cdot_{12} 3 = 0$, $4 \cdot_{12} 6 = 0$, and $4 \cdot_{12} 9 = 0$ as well. Thus your recipient has four different choices for the original message, which is almost as bad as having to guess the original message itself!

It might appear that special problems arose because the encoded message was 0, so the next question in Exercise 2.1-6 gives us an encoded message that is not 0. Suppose that $a = 3$ and $n = 12$. Now we encode the message 6 by computing $3 \cdot_{12} 6 = 6$. Straightforward calculation shows that $3 \cdot_{12} 2 = 6$, $3 \cdot_{12} 6 = 6$, and $3 \cdot_{12} 10 = 6$. Thus, the message 6 can be decoded in three possible ways, as 2, 6, or 10.

The final question in Exercise 2.1-6 provides some hope. Let $a = 5$ and $n = 12$. The message is 7 is encoded as $5 \cdot_{12} 7 = 11$. Simple checking of $5 \cdot_{12} 1$, $5 \cdot_{12} 2$, $5 \cdot_{12} 3$, and so on shows that 7 is

[3] If each segement of a message were equally likely to be any number between 0 and $n$, and if any second (or third, etc.) segment were equally likely to follow any first segement, then knowing the difference between two segments would yield no information about the two segments. However, because language is structured and most information is structured, these two conditions are highly unlikely to hold, in which case your adversary could apply structural knowledge to deduce information about your two messages from their difference.

the unique solution in $Z_{12}$ to the equation $5 \cdot_{12} x = 11$. Thus in this case we can correctly decode the message.

One key point that this example shows is that our system of encrypting messages must be one-to-one. That is, each unencrypted message must correspond to a different encrypted message.

As we shall see in the next section, the kinds of problems we had in Exercise 2.1-6 happen only when $a$ and $n$ have a common divisor that is greater than 1. Thus, when $a$ and $n$ have no common factors greater than one, all our receiver needs to know is how to divide by $a$ in $Z_n$, and she can decrypt our message. If you don't now know how to divide by $a$ in $Z_n$, then you can begin to understand the idea of public key cryptography. The message is there for anyone who knows how to divide by $a$ to find, but if nobody but our receiver can divide by $a$, we can tell everyone what $a$ and $n$ are and our messages will still be secret. That is the second point our system illustrates. If we have some knowledge that nobody else has, such as how to divide by $a$ mod $n$, then we have a possible public key cryptosystem. As we shall soon see, dividing by $a$ is not particularly difficult, so a better trick is needed for public key cryptography to work.

### Important Concepts, Formulas, and Theorems

1. *Cryptography* is the study of methods to send and receive secret messages.
   (a) The *sender* wants to send a message to a *receiver*.
   (b) The *adversary* wants to steal the message.
   (c) In *private key cryptography*, the sender and receiver agree in advance on a *secret code*, and then send messages using that code.
   (d) In *public key cryptography*, the encoding method can be published. Each person has a *public key* used to encrypt messages and a *secret key* used to encrypt an encrypted message.
   (e) The original message is called the *plaintext*.
   (f) The encoded text is called the *ciphertext*.
   (g) A *Caesar cipher* is one in which each letter of the alphabet is shifted by a fixed amount.
2. *Euclid's Division Theorem.* For every integer $m$ and positive integer $n$, there exist unique integers $q$ and $r$ such that $m = nq + r$ and $0 \leq r < n$. By definition, $r$ is equal to $m \bmod n$.
3. *Adding multiples of $n$ does not change values mod $n$.* That is, $i \bmod n = (i + kn) \bmod n$ for any integer $k$.
4. *Mods (by $n$) can be taken anywhere in calculation, so long as we take the final result mod $n$.*

$$\begin{aligned}(i+j) \bmod n &= [i + (j \bmod n)] \bmod n \\ &= [(i \bmod n) + j] \bmod n \\ &= [(i \bmod n) + (j \bmod n)] \bmod n\end{aligned}$$

$$\begin{aligned}(i \cdot j) \bmod n &= [i \cdot (j \bmod n)] \bmod n \\ &= [(i \bmod n) \cdot j] \bmod n \\ &= [(i \bmod n) \cdot (j \bmod n)] \bmod n\end{aligned}$$

5. *Commutative, associative and distributive laws.* Addition and multiplication mod $n$ satisfy the commutative and associative laws, and multiplication distributes over addition.

6. $Z_n$. We use the notation $Z_n$ to represent the integers $0, 1, \ldots, n-1$ together with a redefinition of addition, which we denote by $+_n$, and a redefinition of multiplication, which we denote $\cdot_n$. The redefinitions are:

$$\begin{aligned} i +_n j &= (i + j) \bmod n \\ i \cdot_n j &= (i \cdot j) \bmod n \end{aligned}$$

We use the expression "$x \in Z_n$" to mean that $x$ is a variable that can take on any of the integral values between 0 and $n-1$, and that in algebraic expressions involving $x$ we will use $+_n$ and $\cdot_n$. We use the expression $a \in Z_n$ to mean that $a$ is a constant between 0 and $n-1$, and in algebraic expressions involving $a$ we will use $+_n$ and $\cdot_n$.

## Problems

1. What is 14 mod 9? What is $-1$ mod 9? What is $-11$ mod 9?

2. Encrypt the message HERE IS A MESSAGE using a Caeser cipher in which each letter is shifted three places to the right.

3. Encrypt the message HERE IS A MESSAGE using a Caeser cipher in which each letter is shifted three places to the left.

4. How many places has each letter been shifted in the Caesar cipher used to encode the message XNQQD RJXXFLJ?

5. What is $16 +_{23} 18$? What is $16 \cdot_{23} 18$?

6. A short message has been encoded by converting it to an integer by replacing each "a" by 1, each "b" by 2, and so on, and concatenating the integers. The result had six or fewer digits. An unknown number $a$ was added to the message mod 913,647, giving 618,232. Without the knowledge of $a$, what can you say about the message? With the knowledge of $a$, what could you say about the message?

7. What would it mean to say there is an integer $x$ equal to $\frac{1}{4}$ mod 9? If it is meaningful to say there is such an integer, what is it? Is there an integer equal to $\frac{1}{3}$ mod 9? If so, what is it?

8. By multiplying a number $x$ times 487 in $Z_{30031}$ we obtain 13008. If you know how to find the number $x$, do so. If not, explain why the problem seems difficult to do by hand.

9. Write down the addition table for $+_7$ addition. Why is the table symmetric? Why does every number appear in every row?

10. It is straightforward to solve, for $x$, any equation of the form

$$x +_n a = b$$

in $Z_n$, and to see that the result will be a unique value of $x$. On the other hand, we saw that 0, 3, 6, and 9 are all solutions to the equation

$$4 \cdot_{12} x = 0.$$

a) Are there any integral values of $a$ and $b$, with $1 \leq a, b < 12$, for which the equation $a \cdot_{12} x = b$ does not have any solutions in $Z_{12}$? If there are, give one set of values for $a$ and $b$. If there are not, explain how you know this.

b) Are there any integers $a$, with $1 < a < 12$ such that for every integral value of $b$, $1 \leq b < 12$, the equation $a \cdot_{12} x = b$ has a solution? If so, give one and explain why it works. If not, explain how you know this.

11. Does every equation of the form $a \cdot_n x = b$, with $a, b \in Z_n$ have a solution in $Z_5$? in $Z_7$? in $Z_9$? in $Z_{11}$?

12. Recall that if a prime number divides a product of two integers, then it divides one of the factors.

a) Use this to show that as $b$ runs though the integers from 0 to $p-1$, with $p$ prime, the products $a \cdot_p b$ are all different (for each fixed choice of $a$ between 1 and $p-1$).

b) Explain why every integer greater than 0 and less than $p$ has a unique multiplicative inverse in $Z_p$, if $p$ is prime.

13. Explain why, if you were encoding messages $x_1$, $x_2$, and $x_3$ to obtain $y_1$, $y_2$ and $y_3$ by adding $a$ mod $n$, your adversary would know that at least one of the differences $y_1 - y_2$, $y_1 - y_3$ or $y_2 - y_3$ taken in the integers, not in $Z_n$, would be the difference of two unencoded messages. (Note: we are not saying that your adversary would know which of the three was such a difference.)

14. Modular arithmetic is used in generating pseudo-random numbers. One basic algorithm (still widely used) is *linear congruential random number generation*. The following piece of code generates a sequence of numbers that may appear random to the unaware user.

```
(1)  set seed to a random value
(2)  x = seed
(3)  Repeat
(4)        x = (ax + b) mod n
(5)        print x
(6)  Until x = seed
```

Execute the loop by hand for $a = 3$, $b = 7$, $n = 11$ and $seed = 0$. How "random" are these random numbers?

15. Write down the $\cdot_7$ multiplication table for $Z_7$.

16. Prove the equalities for multiplication in Lemma 2.3.

17. State and prove the associative law for $\cdot_n$ multiplication.

18. State and prove the distributive law for $\cdot_n$ multiplication over $+_n$ addition.

19. Write pseudocode to take $m$ integers $x_1, x_2, \ldots, x_m$, and an integer $n$, and return $\Pi_i^m x_i \bmod n$. Be careful about *overflow*; in this context, being careful about overflow means that at no point should you ever compute a value that is greater than $n^2$.

20. Write pseudocode to decode a message that has been encoded using the algorithm

    - take three consecutive letters,
    - reverse their order,
    - interpret each as a base 26 integer (with A=0; B=1, etc.),
    - multiply that number by 37,
    - add 95 and then
    - convert that number to base 8.

    Continue this processing with each block of three consecutive letters. Append the blocks, using either an 8 or a 9 to separate the blocks. Finally, reverse the number, and replace each digit 5 by two 5's.

## 2.2 Inverses and GCDs

### Solutions to Equations and Inverses mod $n$

In the last section we explored multiplication in $Z_n$. We saw in the special case with $n = 12$ and $a = 4$ that if we used multiplication by $a$ in $Z_n$ to encrypt a message, then our receiver would need to be able to solve, for $x$, the equation $4 \cdot_n x = b$ in order to decode a received message $b$. We saw that if the encrypted message was 0, then there were four possible values for $x$. More generally, Exercise 2.1-6 and some of the problems in the last section show that for certain values of $n$, $a$, and $b$, equations of the form $a \cdot_n x = b$ have a unique solution, while for other values of $n$, $a$, and $b$, the equation could have no solutions, or more than one solution.

To decide whether an equation of the form $a \cdot_n x = b$ has a unique solution in $Z_n$, it helps know whether $a$ has a *multiplicative inverse* in $Z_n$, that is, whether there is another number $a'$ such that $a' \cdot_n a = 1$. For example, in $Z_9$, the inverse of 2 is 5 because $2 \cdot_9 5 = 1$. On the other hand, 3 does not have an inverse in $Z_9$, because the equation $3 \cdot_9 x = 1$ does not have a solution. (This can be verified by checking the 9 possible values for $x$.) If $a$ does have an inverse $a'$, then we can find a solution to the equation

$$a \cdot_n x = b \ .$$

To do so, we multiply both sides of the equation by $a'$, obtaining

$$a' \cdot_n (a \cdot_n x) = a' \cdot_n b.$$

By the associative law, this gives us

$$(a' \cdot_n a) \cdot_n x = a' \cdot_n b.$$

But $a' \cdot_n a = 1$ by definition so we have that

$$x = a' \cdot_n b \ .$$

Since this computation is valid for any $x$ that satisfies the equation, we conclude that the only $x$ that satisfies the equation is $a' \cdot_n b$. We summarize this discussion in the following lemma.

**Lemma 2.5** *Suppose $a$ has a multiplicative inverse $a'$ in $Z_n$. Then for any $b \in Z_n$, the equation*

$$a \cdot_n x = b$$

*has the unique solution*

$$x = a' \cdot_n b \ .$$

Note that this lemma holds for *any* value of $b \in Z_n$.

This lemma tells us that whether or not a number has an inverse mod $n$ is important for the solution of modular equations. We therefore wish to understand exactly when a member of $Z_n$ has an inverse.

### Inverses mod $n$

We will consider some of the examples related to Problem 11 of the last section.

**Exercise 2.2-1** Determine whether every element $a$ of $Z_n$ has an inverse for $n$= 5, 6, 7, 8, and 9.

**Exercise 2.2-2** If an element of $Z_n$ has a multiplicative inverse, can it have two different multiplicative inverses?

For $Z_5$, we can determine by multiplying each pair of nonzero members of $Z_5$ that the following table gives multiplicative inverses for each element $a$ of $Z_5$. For example, the products $2 \cdot_5 1 = 2$, $2 \cdot_5 2 = 4$, $2 \cdot_5 3 = 1$, and $2 \cdot_5 4 = 3$ tell us that 3 is the unique multiplicative inverse for 2 in $Z_5$. This is the reason we put 3 below 2 in the table. One can make the same kinds of computations with 3 or 4 instead of 2 on the left side of the products to get the rest of the table.

| a | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $a'$ | 1 | 3 | 2 | 4 |

For $Z_7$, we have similarly the table

| a | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $a'$ | 1 | 4 | 5 | 2 | 3 | 6 |

.

For $Z_9$, we have already said that $3 \cdot_9 x = 1$ does not have a solution, so by Lemma 2.5, 3 does not have an inverse. (Notice how we are using the Lemma. The Lemma says that if 3 had an inverse, then the equation $3 \cdot_9 x = 1$ *would* have a solution, and this would contradict the fact that $3 \cdot_9 x = 1$ does not have a solution. Thus assuming that 3 had an inverse would lead us to a contradiction. Therefore 3 has no multiplicative inverse.)

This computation is a special case of the following corollary[4] to Lemma 2.5.

**Corollary 2.6** *Suppose there is a $b$ in $Z_n$ such that the equation*

$$a \cdot_n x = b$$

*does not have a solution. Then $a$ does not have a multiplicative inverse in $Z_n$.*

**Proof:** Suppose that $a \cdot_n x = b$ has no solution. Suppose further that $a$ does have a multiplicative inverse $a'$ in $Z_n$. Then by Lemma 2.5, $x = a'b$ is a solution to the equation $a \cdot_n x = b$. This contradicts the hypothesis given in the corollary that the equation does not have a solution. Thus some supposition we made above must be incorrect. One of the assumptions, namely that $a \cdot_n x = b$ has no solution was the hypothesis given to us in the statement of the corollary. The only other supposition we made was that $a$ has an inverse $a'$ in $Z_n$. Thus this supposition must be incorrect as it led to the contradiction. Therefore, it must be case that $a$ does not have a multiplicative inverse in $Z_n$. ■

Our proof of the corollary is a classical example of the use of contradiction in a proof. The principle of *proof by contradiction* is the following.

[4]In the next section we shall see that this corollary is actually equivalent to Lemma to Lemma 2.5.

**Principle 2.1 (Proof by contradiction)** *If by assuming a statement we want to prove is false, we are lead to a contradiction, then the statement we are trying to prove must be true.*

We can actually give more information than Exercise 1 asks for. You can check that the table below shows an X for the elements of $Z_9$ that do not have inverses and gives an inverse for each element that has one

| a | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| $a'$ | 1 | 5 | X | 7 | 2 | X | 4 | 8 |

.

In $Z_6$, 1 has an inverse, namely 1, but the equations

$$2 \cdot_6 1 = 2, \quad 2 \cdot_6 2 = 4, \quad 2 \cdot_6 3 = 0, \quad 2 \cdot_6 4 = 2, \quad 2 \cdot_6 5 = 4$$

tell us that 2 does not have an inverse. Less directly, but with less work, we see that the equation $2 \cdot_6 x = 3$ has no solution because $2x$ will always be even, so $2x$ mod 6 will always be even. Then Corollary 2.6 tells us that 2 has no inverse. Once again, we give a table that shows exactly which elements of $Z_6$ have inverses.

| a | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $a'$ | 1 | X | X | X | 5 |

A similar set of equations shows that 2 does not have an inverse in $Z_8$. The following table shows which elements of $Z_8$ have inverses.

| a | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $a'$ | 1 | X | 3 | X | 5 | X | 7 |

.

We see that every nonzero element in $Z_5$ and $Z_7$ does have a multiplicative inverse, but in $Z_6$, $Z_8$, and $Z_9$, some elements do not have a multiplicative inverse. Notice that 5 and 7 are prime, while 6, 8, and 9 are not. Further notice that the elements in $Z_n$ that do not have a multiplicative inverse are exactly those that share a common factor with $n$.

We showed that 2 has exactly one inverse in $Z_5$ by checking each multiple of 2 in $Z_5$ and showing that exactly one multiple of 2 equals 1. In fact, for any element that has an inverse in $Z_5$, $Z_6$, $Z_7$, $Z_8$, and $Z_9$, you can check in the same way that it has exactly one inverse. We explain why in a theorem.

**Theorem 2.7** *If an element of $Z_n$ has a multiplicative inverse, then it has exactly one inverse.*

**Proof:** Suppose that an element $a$ of $Z_n$ has an inverse $a'$. Suppose that $a^*$ is also an inverse of $a$. Then $a'$ is a solution to $a \cdot_n x = 1$ and $a^*$ is a solution to $a \cdot_n x = 1$. But by Lemma 2.5, the equation $a \cdot_n x = 1$ has a unique solution. Therefore $a' = a^*$. ■

Just as we use $a^{-1}$ to denote the inverse of $a$ in the real numbers, we use $a^{-1}$ to denote the unique inverse of $a$ in $Z_n$ when $a$ has an inverse. Now we can say precisely what we mean by division in $Z_n$. We will define what we mean by *dividing* a member of $Z_n$ by $a$ only in the case that $a$ has an inverse $a^{-1}$ mod $n$. In this case dividing $b$ by $a$ mod $n$ is defined to be same as multiplying $b$ by $a^{-1}$ mod $n$. We were led to our discussion of inverses because of their role in solving equations. We observed that in our examples, an element of $Z_n$ that has an inverse mod $n$ has no factors greater than 1 in common with $n$. This is a statement about $a$ and $n$ as integers with ordinary multiplication rather than multiplication mod $n$. Thus to prove that $a$ has an inverse mod $n$ if and only if $a$ and $n$ have no common factors other than 1 and -1, we have to convert the equation $a \cdot_n x = 1$ into an equation involving ordinary multiplication.

### Converting Modular Equations to Normal Equations

We can re-express the equation

$$a \cdot_n x = 1$$

as

$$ax \bmod n = 1.$$

But the definition of $ax \bmod n$ is that it is the remainder $r$ we get when we write $ax = qn + r$, with $0 \leq r < n$. This means that $ax \bmod n = 1$ if and only if there is an integer $q$ with $ax = qn + 1$, or

$$ax - qn = 1. \tag{2.8}$$

Thus we have shown

**Lemma 2.8** *The equation*

$$a \cdot_n x = 1$$

*has a solution in $Z_n$ if and only if there exist integers $x$ and $y$ such that*

$$ax + ny = 1.$$

**Proof:** We simply take $y = -q$. ■

We make the change from $-q$ to $y$ for two reasons. First, if you read a number theory book, you are more likely to see the equation with $y$ in this context. Second, to solve this equation, we must find both $x$ and $y$, and so using a letter near the end of the alphabet in place of $-q$ emphasizes that this is a variable for which we need to solve.

It appears that we have made our work harder, not easier, as we have converted the problem of solving (in $Z_n$) the equation $a \cdot_n x = 1$, an equation with just one variable $x$ (that could only have $n-1$ different values), to a problem of solving Equation 2.8, which has two variables, $x$ and $y$. Further, in this second equation, $x$ and $y$ can take on any integer values, even negative values.

However, this equation will prove to be exactly what we need to prove that $a$ has an inverse mod $n$ if and only if $a$ and $n$ have no common factors larger than one.

### Greatest Common Divisors (GCD)

**Exercise 2.2-3** Suppose that $a$ and $n$ are integers such that $ax+ny = 1$, for some integers $x$ and $y$. What does that tell us about being able to find a (multiplicative) inverse for $a \pmod n$? In this situation, if $a$ has an inverse in $Z_n$, what is it?

**Exercise 2.2-4** If $ax+ny = 1$ for integers $x$ and $y$, can $a$ and $n$ have any common divisors other than 1 and -1?

In Exercise 2.2-3, since by Lemma 2.8, the equation $a \cdot_n x = 1$ has a solution in $Z_n$ if and only if there exist integers $x$ and $y$ such that $ax + ny = 1$, we can can conclude that

**Theorem 2.9** *A number $a$ has a multiplicative inverse in $Z_n$ if and only if there are integers $x$ and $y$ such that $ax + ny = 1$.*

We answer the rest of Exercise 2.2-3 with a corollary.

**Corollary 2.10** *If $a \in Z_n$ and $x$ and $y$ are integers such that $ax+ny = 1$, then the multiplicative inverse of $a$ in $Z_n$ is $x \bmod n$.*

**Proof:** Since $n \cdot_n y = 0$ in $Z_n$, we have $a \cdot_n x = 1$ in $Z_n$ and therefore $x$ is the inverse of $a$ in $Z_n$. ■

Now let's consider Exercise 2.2-4. If $a$ and $n$ have a common divisor $k$, then there must exist integers $s$ and $q$ such that

$$a = sk$$

and

$$n = qk \ .$$

Substituting these into $ax + ny = 1$, we obtain

$$\begin{aligned} 1 &= ax + ny \\ &= skx + qky \\ &= k(sx + qy). \end{aligned}$$

But then $k$ is a divisor of 1. Since the only integer divisors of 1 are $\pm 1$, we must have $k = \pm 1$. Therefore $a$ and $n$ can have no common divisors other than 1 and -1.

In general, the **greatest common divisor** of two numbers $j$ and $k$ is the largest number $d$ that is a factor of both $j$ and $k$.[5] We denote the greatest common divisor of $j$ and $k$ by $\gcd(j, k)$.

We can now restate Exercise 2.2-4 as follows:

**Lemma 2.11** *Given $a$ and $n$, if there exist integers $x$ and $y$ such that $ax + ny = 1$ then $\gcd(a, n) = 1$.*

If we combine Theorem 2.9 and Lemma 2.11, we see that that if $a$ has a multiplicative inverse mod $n$, then $\gcd(a, n) = 1$. It is natural to ask whether the statement that "if $\gcd a, n = 1$, then $a$ has a multiplicative inverse" is true as well.[6] If so, this would give us a way to test whether $a$ has a multiplicative inverse mod $n$ by computing the greatest common divisor of $a$ and $n$. For this purpose we would need an algorithm to find $\gcd(a, n)$. It turns out that there is such an algorithm, and a byproduct of the algorithm is a proof of our conjectured converse statement! When two integers $j$ and $k$ have $\gcd(j, k) = 1$, we say that $j$ and $k$ are *relatively prime*.

### Euclid's Division Theorem

One of the important tools in understanding greatest common divisors is Euclid's Division Theorem, a result which has already been important to us in defining what we mean by $m \bmod n$. While it appears obvious, as do some other theorems in number theory, it follows from simpler principles of number theory, and the proof helps us understand how the greatest common divisor

[5] There is one common factor of $j$ and $k$ for sure, namely 1. No common factor can be larger than the smaller of $j$ and $k$ in absolute value, and so there must be a largest common factor.

[6] Notice that this statement is *not* equivalent to the statement in the lemma. This statement is what is called the "converse" of the lemma; we will explain the idea of converse statements more in Chapter 3.

algorithm works. Thus we restate it and present a proof here. Our proof uses the method of proof by contradiction, which you first saw in Corollary 2.6. Notice that we are assuming $m$ is nonnegative which we didn't assume in our earlier statement of Euclid's Division Theorem, Theorem 2.1. In Problem 16 we will explore how we can remove this additional assumption.

**Theorem 2.12 (Euclid's Division Theorem, restricted version)** *For every nonnegative integer $m$ and positive integer $n$, there exist unique integers $q$ and $r$ such that $m = nq + r$ and $0 \leq r < n$. By definition, $r$ is equal to $m \bmod n$.*

**Proof:** To prove this theorem, assume instead, for purposes of contradiction, that it is false. Among all pairs $(m, n)$ that make it false, choose the smallest $m$ that makes it false. We cannot have $m < n$ because then the statement would be true with $q = 0$ and $r = m$, and we cannot have $m = n$ because then the statement is true with $q = 1$ and $r = 0$. This means $m - n$ is a positive number smaller than $m$. We assumed that $m$ was the smallest value that made the theorem false, and so the theorem must be true for the pair $(m - n, n)$. Therefore, there must exist a $q'$ and $r'$ such that

$$m - n = q'n + r', \text{ with } 0 \leq r' < n.$$

Thus $m = (q' + 1)n + r'$. Now, by setting $q = q' + 1$ and $r = r'$, we can satisfy the theorem for the pair $(m, n)$, contradicting the assumption that the statement is false. Thus the only possibility is that the statement is true. ■

The proof technique used here is a special case of proof by contradiction. We call the technique *proof by smallest counterexample*. In this method, we assume, as in all proofs by contradiction, that the theorem is false. This implies that there must be a *counterexample* which does not satisfy the conditions of the theorem. In this case that counterexample would consist of numbers $m$ and $n$ such that *no* integers $q$ and $r$ exist which satisfy $m = qn + r$. Further, if there are counterexamples, then there must be one having the smallest $m$. We assume we have chosen a counter example with such a smallest $m$. the we reason that if such an $m$ exists, then every example wit a smaller $m$ satisfies the conclusion of the theorem. If we can then use a smaller true example to show that our supposedly false example is true as well, we have created a contradiction. The only thing this can contradict is our assumption that the theorem was false. Therefore this assumption has to be invalid, and the theorem has to be true. As we will see in Chapter 4.1, this method is closely related to a proof method called *proof by induction* and to recursive algorithms. In essence, the proof of Theorem 2.1 describes a recursive program to find $q$ and $r$ in the theorem above so that $0 \leq r < n$.

**Exercise 2.2-5** Suppose that $k = jq + r$ as in Euclid's Division Theorem. Is there a relationship between $\gcd(j, k)$ and $\gcd(r, j)$?

In this exercise, if $r = 0$, then $\gcd(r, j)$ is $j$, because any number is a divisor of zero. But this is the GCD of $k$ and $j$ as well since in this case $k = jq$. The answer to the remainder of Exercise 2.2-5 appears in the following lemma.

**Lemma 2.13** *If $j$, $k$, $q$, and $r$ are positive integers such that $k = jq + r$ then*

$$\gcd(j, k) = \gcd(r, j). \tag{2.9}$$

**Proof:** In order to prove that both sides of Equation 2.9 are equal, we will show that they have exactly the same set of factors. That is, we will first show that if $d$ is a factor of the left-hand side, then it is a factor of the right-hand side. Second, we will show that if $d$ is a factor of the right-hand side, then it is a factor of the left-hand side.

If $d$ is a factor of $\gcd(j,k)$ then it is a factor of both $j$ and $k$. There must be integers $i_1$ and $i_2$ so that $k = i_1 d$ and $j = i_2 d$. Thus $d$ is also a factor of

$$\begin{aligned} r &= k - jq \\ &= i_1 d - i_2 dq \\ &= (i_1 - i_2 q)d \ . \end{aligned}$$

Since $d$ is a factor of $j$ (by supposition) and $r$ (by the equation above), it must be a factor of $\gcd(r,j)$.

Similarly, if $d$ is a factor of $\gcd(r,j)$, it is a factor of $j$ and $r$, and we can write $j = i_3 d$ and $r = i_4 d$. Therefore,

$$\begin{aligned} k &= jq + r \\ &= i_3 dq + i_4 d \\ &= (i_3 q + i_4)d \ , \end{aligned}$$

and $d$ is a factor of $k$ and therefore of $\gcd(j,k)$.

Since $\gcd(j,k)$ has the same factors as $\gcd(r,j)$ they must be equal. ■

While we did not need to assume $r < j$ in order to prove the lemma, Theorem 2.1 tells us we may assume $r < j$. The assumption in the lemma that $j$, $q$ and $r$ are positive implies that $j < k$. Thus this lemma reduces our problem of finding $\gcd(j,k)$ to the simpler (in a recursive sense) problem of finding $\gcd(r,j)$.

### The GCD Algorithm

**Exercise 2.2-6** Using Lemma 2.13, write a recursive algorithm to find $\gcd(j,k)$, given that $j \leq k$. Use it (by hand) to find the GCD of 24 and 14 and the GCD of 252 and 189.

Our algorithm for Exercise 2.2-6 is based on Lemma 2.13 and the observation that if $k = jq$, for any $q$, then $j = \gcd(j,k)$. We first write $k = jq + r$ in the usual way. If $r = 0$, then we return $j$ as the greatest common divisor. Otherwise, we apply our algorithm to find the greatest common divisor of $j$ and $r$. Finally, we return the result as the greatest common divisor of $j$ and $k$.

To find

$$\gcd(14, 24)$$

we write

$$24 = 14(1) + 10.$$

In this case $k = 24$, $j = 14$, $q = 1$ and $r = 10$. Thus we can apply Lemma 2.13 and conclude that

$$\gcd(14, 24) = \gcd(10, 14).$$

We therefore continue our computation of $\gcd(10, 14)$, by writing $14 = 10 \cdot 1 + 4$, and have that

$$\gcd(10, 14) = \gcd(4, 10).$$

Now,

$$10 = 4 \cdot 2 + 2,$$

and so

$$\gcd(4, 10) = \gcd(2, 4).$$

Now

$$4 = 2 \cdot 2 + 0,$$

so that now $k = 4$, $j = 2$, $q = 2$, and $r = 0$. In this case our algorithm tells us that our current value of $j$ is the GCD of the original $j$ and $k$. This step is the base case of our recursive algorithm. Thus we have that

$$\gcd(14, 24) = \gcd(2, 4) = 2.$$

While the numbers are larger, it turns out to be even easier to find the GCD of 252 and 189. We write

$$252 = 189 \cdot 1 + 63,$$

so that $\gcd(189, 252) = \gcd(63, 189)$, and

$$189 = 63 \cdot 3 + 0.$$

This tells us that $\gcd(189, 252) = \gcd(189, 63) = 63$.

### Extended GCD algorithm

By analyzing our process in a bit more detail, we will be able to return not only the greatest common divisor, but also numbers $x$ and $y$ such that $\gcd(j, k) = jx + ky$. This will solve the problem we have been working on, because it will prove that if $\gcd(a, n) = 1$, then there are integers $x$ and $y$ such that $ax + ny = 1$. Further it will tell us how to find $x$, and therefore the multiplicative inverse of $a$.

In the case that $k = jq$ and we want to return $j$ as our greatest common divisor, we also want to return 1 for the value of $x$ and 0 for the value of $y$. Suppose we are now in the case that that $k = jq + r$ with $0 < r < j$ (that is, the case that $k \neq jq$). Then we recursively compute $\gcd(r, j)$ and in the process get an $x'$ and a $y'$ such that $\gcd(r, j) = rx' + jy'$. Since $r = k - jq$, we get by substitution that

$$\gcd(r, j) = (k - jq)x' + jy' = kx' + j(y' - qx').$$

Thus when we return $\gcd(r, j)$ as $\gcd(j, k)$, we want to return the value of $x'$ as $y$ and and the value of $y' - qx'$ as $x$.

We will refer to the process we just described as "Euclid's extended GCD algorithm."

**Exercise 2.2-7** Apply Euclid's extended GCD algorithm to find numbers $x$ and $y$ such that the GCD of 14 and 24 is $14x + 24y$.

For our discussion of Exercise 2.2-7 we give pseudocode for the extended GCD algorithm. While we expressed the algorithm more concisely earlier by using recursion, we will give an iterative version that is longer but can make the computational process clearer. Instead of using the variables $q, j, k, r, x$ and $y$, we will use six arrays, where $q[i]$ is the value of $q$ computed on the $i$th iteration, and so forth. We will use the index zero for the input values, that is $j[0]$ and $k[0]$ will be the numbers whose gcd we wish to compute. Eventually $x[0]$ and $y[0]$ will become the $x$ and $y$ we want.

(In Line 0 we are using the notation $\lfloor x \rfloor$ to stand for the floor of $x$, the largest integer less than or equal to $x$.)

```
gcd(j, k)
// assume that j < k
(1)  i = 0; k[i] = k; j[i] = j
(2)  Repeat
(3)        q[i] = ⌊k[i]/j[i]⌋
(4)        r[i] = k[i] − q[i]j[i]
(5)        k[i + 1] = j[i]; j[i + 1] = r[i]
(6)        i = i + 1
(7)  Until  (r[i − 1] = 0)
// we have found the value of the gcd, now we compute the x and y
(8)  i = i − 1
(9)  gcd = j[i]
(10) y[i] = 0; x[i] = 1
(11) i = i − 1
(12) While (i ≥ 0)
(13)       y[i] = x[i + 1]
(14)       x[i] = y[i + 1] − q[i]x[i + 1]
(15)       i = i − 1
(16) Return gcd
(17) Return x
(18) Return y
```

We show the details of how this algorithm applies to $\gcd(24, 14)$ in Table 2.1. In a row, the $q[i]$ and $r[i]$ values are computed from the $j[i]$ and $k[i]$ values. Then the $j[i]$ and $r[i]$ are passed down to the next row as $k[i+1]$ and $j[i+1]$ respectively. This process continues until we finally reach a case where $k[i] = q[i]j[i]$ and we can answer $j[i]$ for the gcd. We can then begin computing $x[i]$ and $y[i]$. In the row with $i = 3$, we have that $x[i] = 0$ and $y[i] = 1$. Then, as $i$ decreases, we compute $x[i]$ and $y[i]$ for a row by setting $y[i]$ to $x[i+1]$ and $x[i]$ to $y[i+1] - q[i]x[i+1]$. We note that in every row, we have the property that $j[i]x[i] + k[i]y[i] = \gcd(j, k)$.

We summarize Euclid's extended GCD algorithm in the following theorem:

**Theorem 2.14** *Given two integers $j$ and $k$, Euclid's extended GCD algorithm computes* $\gcd(j, k)$ *and two integers $x$ and $y$ such that* $\gcd(j, k) = jx + ky$ .

We now use Eculid's extended GCD algorithm to extend Lemma 2.11.

| $i$ | $j[i]$ | $k[i]$ | $q[i]$ | $r[i]$ | $x[i]$ | $y[i]$ |
|---|---|---|---|---|---|---|
| 0 | 14 | 24 | 1 | 10 | | |
| 1 | 10 | 14 | 1 | 4 | | |
| 2 | 4 | 10 | 2 | 2 | | |
| 3 | 2 | 4 | 2 | 0 | 1 | 0 |
| 2 | 4 | 10 | 2 | 2 | $-2$ | 1 |
| 1 | 10 | 14 | 1 | 4 | 3 | $-2$ |
| 0 | 14 | 24 | 1 | 10 | $-5$ | 3 |

$\gcd = 2$
$x = -5$
$y = 3$

Table 2.1: The computation of $\gcd(14, 24)$ by algorithm $\gcd(j, k)$.

**Theorem 2.15** *Two positive integers $j$ and $k$ have greatest common divisor 1 (and thus are relatively prime) if and only if there are integers $x$ and $y$ such that $jx + ky$=1.*

**Proof:** The statement that if there are integers $x$ and $y$ such that $jx+ky = 1$, then $\gcd(j, k) = 1$ is proved in Lemma 2.11. In other words, $\gcd(j, k) = 1$ if there are integers $x$ and $y$ such that $jx + ky = 1$.

On the other hand, we just showed, by Euclid's extended GCD algorithm, that given positive integers $j$ and $k$, there are integers $x$ and $y$ such that $\gcd(j, k) = jx+ky$. Therefore, $\gcd(j, k) = 1$ only if there are integers $x$ and $y$ such that $jx + ky = 1$. ■

Combining Lemma 2.8 and Theorem 2.15, we obtain:

**Corollary 2.16** *For any positive integer $n$, an element $a$ of $Z_n$ has a multiplicative inverse if and only if* $\gcd(a, n) = 1$.

Using the fact that if $n$ is prime, $\gcd(a, n) = 1$ for all non-zero $a \in Z_n$, we obtain

**Corollary 2.17** *For any prime $p$, every non-zero element $a$ of $Z_p$ has an inverse.*

### Computing Inverses

Not only does Euclid's extended GCD algorithm tell us if an inverse exists, but, just as we saw in Exercise 2.2-3 it computes it for us. Combining Exercise 2.2-3 with Theorem 2.15, we get

**Corollary 2.18** *If an element $a$ of $Z_n$ has an inverse, we can compute it by running Euclid's extended GCD algorithm to determine integers $x$ and $y$ so that $ax + ny = 1$. Then the inverse of $a$ in $Z_n$ is $x \bmod n$.*

For completeness, we now give pseudocode which determines whether an element $a$ in $Z_n$ has an inverse and computes the inverse if it exists:

```
inverse(a, n)
(1)  Run procedure gcd(a, n) to obtain gcd(a, n), x and y
(2)  if gcd(a, n) = 1
(3)       return x
(4)  else
(5)       print ‘‘no inverse exists’’
```

The correctness of the algorithm follows immediately from the fact that $\gcd(a, n) = ax + ny$, so if $\gcd(a, n) = 1$, $ax \bmod n$ must be equal to 1.

## Important Concepts, Formulas, and Theorems

1. *Multiplicative inverse.* $a'$ is a multiplicative inverse of $a$ in $Z_n$ if $a \cdot_n a' = 1$. If $a$ has a multiplicative inverse, then it has a unique multiplicative inverse which we denote by $a^{-1}$.

2. *An important way to solve modular equations.* Suppose $a$ has a multiplicative inverse mod n, and this inverse is $a^{-1}$. Then for any $b \in Z_n$, the unique solution to the equation

$$a \cdot_n x = b$$

   is

$$x = a^{-1} \cdot_n b \ .$$

3. *Converting modular to regular equations.* The equation

$$a \cdot_n x = 1$$

   has a solution in $Z_n$ if and only if there exist integers $x$ and $y$ such that

$$ax + ny = 1 \ .$$

4. *When do inverses exist in $Z_n$?* A number $a$ has a multiplicative inverse in $Z_n$ if and only if there are integers $x$ and $y$ such that $ax + ny = 1$.

5. *Greatest common divisor (GCD).* The *greatest common divisor* of two numbers $j$ and $k$ is the largest number $d$ that is a factor of both $j$ and $k$.

6. *Relatively prime.* When two numbers, $j$ and $k$ have $\gcd(j, k) = 1$, we say that $j$ and $k$ are relatively prime.

7. *Connecting inverses to GCD.* Given $a$ and $n$, if there exist integers $x$ and $y$ such that $ax + ny = 1$ then $\gcd(a, n) = 1$.

8. *GCD recursion lemma.* If $j$, $k$, $q$, and $r$ are positive integers such that $k = jq + r$ then $\gcd(j, k) = \gcd(r, j)$.

9. *Euclid's GCD algorithm.* Given two numbers $j$ and $k$, this algorithm returns $\gcd(j, k)$.

10. *Euclid's extended GCD algorithm.* Given two numbers $j$ and $k$, this algorithm returns $\gcd(j, k)$, and two integers $x$ and $y$ such that $\gcd(j, k) = jx + ky$.

11. *Relating GCD of 1 to Euclid's extended GCD algorithm.* Two positive integers $j$ and $k$ have greatest common divisor 1 if and only if there are integers $x$ and $y$ such that $jx + ky$=1. One of the integers $x$ and $y$ could be negative.

12. *Restatement for $Z_n$.* $\gcd(a, n) = 1$ if and only if there are integers $x$ and $y$ such that $ax + ny = 1$.

13. *Condition for multiplicative inverse in $Z_n$* For any positive integer $n$, an element $a$ of $Z_n$ has an inverse if and only if $\gcd(a, n) = 1$.

14. *Multiplicative inverses in $Z_p$, $p$ prime* For any prime $p$, every non-zero element $a$ of $Z_p$ has a multiplicative inverse.

15. *A way to solve some modular equations $a \cdot_n x = b$.* Use Euclid's extended GCD algorithm to compute $a^{-1}$ (if it exists), and multiply both sides of the equation by $a^{-1}$. (If $a$ has no inverse, the equation might or might not have a solution.)

## Problems

1. If $a \cdot 133 - m \cdot 277 = 1$, does this guarantee that $a$ has an inverse mod $m$? If so, what is it? If not, why not?

2. If $a \cdot 133 - 2m \cdot 277 = 1$, does this guarantee that $a$ has an inverse mod $m$? If so, what is it? If not, why not?

3. Determine whether every nonzero element of $Z_n$ has a multiplicative inverse for $n = 10$ and $n = 11$.

4. How many elements $a$ are there such that $a \cdot_{31} 22 = 1$? How many elements $a$ are there such that $a \cdot_{10} 2 = 1$?

5. Given an element $b$ in $Z_n$, what can you say in general about the possible number of elements $a$ such that $a \cdot_n b = 1$ in $Z_n$?

6. If $a \cdot 133 - m \cdot 277 = 1$, what can you say about all possible common divisors of $a$ and $m$?

7. Compute the GCD of 210 and 126 by using Euclid's GCD algorithm.

8. If $k = jq + r$ as in Euclid's Division Theorem, is there a relationship between $\gcd(q, k)$ and $\gcd(r, q)$. If so, what is it?

9. Bob and Alice want to choose a key they can use for cryptography, but all they have to communicate is a bugged phone line. Bob proposes that they each choose a secret number, $a$ for Alice and $b$ for Bob. They also choose, over the phone, a prime number $p$ with more digits than any key they want to use, and one more number $q$. Bob will send Alice $bq$ mod $p$, and Alice will send Bob $aq$ mod $p$. Their key (which they will keep secret) will then be $abq$ mod $p$. (Here we don't worry about the details of how they use their key, only with how they choose it.) As Bob explains, their wire tapper will know $p$, $q$, $aq$ mod $p$, and $bq$ mod $p$, but will not know $a$ or $b$, so their key should be safe.

   Is this scheme safe, that is can the wiretapper compute $abq$ mod $p$? If so, how does she do it?

Alice says "You know, the scheme sounds good, but wouldn't it be more complicated for the wire tapper if I send you $q^a \bmod p$, you send me $q^b \pmod p$ and we use $q^{ab} \bmod p$ as our key?" In this case can you think of a way for the wire tapper to compute $q^{ab} \bmod p$? If so, how can you do it? If not, what is the stumbling block? (It is fine for the stumbling block to be that you don't know how to compute something; you don't need to prove that you can't compute it.)

10. Write pseudocode for a recursive version of the extended GCD algorithm.

11. Run Euclid's extended GCD algorithm to compute $\gcd(576, 486)$. Show all the steps.

12. Use Euclid's extended GCD algorithm to compute the multiplicative inverse of 16 modulo 103.

13. Solve the equation $16 \cdot_{103} x = 21$ in $Z_{103}$.

14. Which elements of $Z_{35}$ do not have multiplicative inverses in $Z_{35}$?

15. If $k = jq + r$ as in Euclid's Division Theorem, is there a relationship between $\gcd(j, k)$ and $\gcd(r, k)$. If so, what is it?

16. Notice that if $m$ is negative, then $-m$ is positive, so that by Theorem 2.12 $-m = qn + r$, where $0 \leq r < n$. This gives us $m = -qn - r$. If $r = 0$, then we have written $m = q'n + r'$, where $0 \leq r' \leq n$ and $q' = -q$. However if $r > 0$, we cannot take $r' = -r$ and have $0 \leq r' < n$. Notice, though, that since since we have already finished the case $r = 0$ we may assume that $0 \leq n - r < n$. This suggests that if we were to take $r'$ to be $n - r$, we might be able to find a $q'$ so that $m = q'n + r'$ with $0 \leq r' \leq n$, which would let us conclude that Euclid's Division Theorem is valid for negative values $m$ as well as nonnegative values $m$. Find a $q'$ that works and explain how you have extended Euclid's Division Theorem from the version in Theorem 2.12 to the version in Theorem 2.1.

17. The Fibonacci numbers $F_i$ are defined as follows:

$$F_i = \begin{cases} 1 & \text{if } i \text{ is 1 or 2} \\ F_{i-1} + F_{i-2} & \text{otherwise.} \end{cases}$$

What happens when you run Euclid's extended GCD algorithm on $F_i$ and $F_{i-1}$? (We are asking about the execution of the algorithm, not just the answer.)

18. Write (and run on several different inputs) a program to implement Euclid's extended GCD algorithm. Be sure to return $x$ and $y$ in addition to the GCD. About how many times does your program have to make a recursive call to itself? What does that say about how long we should expect it to run as we increase the size of the $j$ and $k$ whose GCD we are computing?

19. The least common multiple of two positive integers $x$ and $y$ is the smallest positive integer $z$ such that $z$ is an integer multiple of both $x$ and $y$. Give a formula for the least common multiple that involves the GCD.

20. Write pseudocode that given integers $a$, $b$ and $n$ in $Z_n$, either computes an $x$ such that $a \cdot_n x = b$ or concludes that no such $x$ exists.

21. Give an example of an equation of the form $a \cdot_n x = b$ that has a solution even though $a$ and $n$ are not relatively prime, or show that no such equation exists.

22. Either find an equation of the form $a \cdot_n x = b$ in $Z_n$ that has a unique solution even though $a$ and $n$ are not relatively prime, or prove that no such equation exists. In other words, you are either to prove the statement that if $a \cdot_n x = b$ has a unique solution in $Z_n$, then $a$ and $n$ are relatively prime or to find a counter example.

## 2.3 The RSA Cryptosystem

### Exponentiation mod $n$

In the previous sections, we have considered encryption using modular addition and multiplication, and have seen the shortcomings of both. In this section, we will consider using exponentiation for encryption, and will show that it provides a much greater level of security.

The idea behind RSA encryption is *exponentiation* in $Z_n$. By Lemma 2.3, if $a \in Z_n$,

$$a^j \bmod n = \underbrace{a \cdot_n a \cdot_n \cdots \cdot_n a}_{j \text{ factors}}. \tag{2.10}$$

In other words $a^j \bmod n$ is the product in $Z_n$ of $j$ factors, each equal to $a$.

### The Rules of Exponents

Lemma 2.3 and the rules of exponents for the integers tell us that

**Lemma 2.19** *For any $a \in Z_n$, and any nonnegative integers $i$ and $j$,*

$$(a^i \bmod n) \cdot_n (a^j \bmod n) = a^{i+j} \bmod n \tag{2.11}$$

*and*

$$(a^i \bmod n)^j \bmod n = a^{ij} \bmod n. \tag{2.12}$$

**Exercise 2.3-1** Compute the powers of 2 mod 7. What do you observe? Now compute the powers of 3 mod 7. What do you observe?

**Exercise 2.3-2** Compute the sixth powers of the nonzero elements of $Z_7$. What do you observe?

**Exercise 2.3-3** Compute the numbers $1 \cdot_7 2$, $2 \cdot_7 2$, $3 \cdot_7 2$, $4 \cdot_7 2$, $5 \cdot_7 2$, and $6 \cdot_7 2$. What do you observe? Now compute the numbers $1 \cdot_7 3$, $2 \cdot_7 3$, $3 \cdot_7 3$, $4 \cdot_7 3$, $5 \cdot_7 3$, and $6 \cdot_7 3$. What do you observe?

**Exercise 2.3-4** Suppose we choose an arbitrary nonzero number $a$ between 1 and 6. Are the numbers $1 \cdot_7 a$, $2 \cdot_7 a$, $3 \cdot_7 a$, $4 \cdot_7 a$, $5 \cdot_7 a$, and $6 \cdot_7 a$ all different? Why or why not?

In Exercise 2.3-1, we have that

$$\begin{aligned}
2^0 \bmod 7 &= 1 \\
2^1 \bmod 7 &= 2 \\
2^2 \bmod 7 &= 4 \\
2^3 \bmod 7 &= 1 \\
2^4 \bmod 7 &= 2 \\
2^5 \bmod 7 &= 4 \\
2^6 \bmod 7 &= 1 \\
2^7 \bmod 7 &= 2 \\
2^8 \bmod 7 &= 4.
\end{aligned}$$

Continuing, we see that the powers of 2 will cycle through the list of three values $1, 2, 4$ again and again. Performing the same computation for 3, we have

$$\begin{aligned} 3^0 \bmod 7 &= 1 \\ 3^1 \bmod 7 &= 3 \\ 3^2 \bmod 7 &= 2 \\ 3^3 \bmod 7 &= 6 \\ 3^4 \bmod 7 &= 4 \\ 3^5 \bmod 7 &= 5 \\ 3^6 \bmod 7 &= 1 \\ 3^7 \bmod 7 &= 3 \\ 3^8 \bmod 7 &= 2. \end{aligned}$$

In this case, we will cycle through the list of six values $1, 3, 2, 6, 4, 5$ again and again.

Now observe that in $Z_7$, $2^6 = 1$ and $3^6 = 1$. This suggests an answer to Exercise 2.3-2. Is it the case that $a^6 \bmod 7 = 1$ for all $a \in Z_7$? We can compute that $1^6 \bmod 7 = 1$, and

$$\begin{aligned} 4^6 \bmod 7 &= (2 \cdot_7 2)^6 \bmod 7 \\ &= (2^6 \cdot_7 2^6) \bmod 7 \\ &= (1 \cdot_7 1) \bmod 7 \\ &= 1. \end{aligned}$$

What about $5^6$? Notice that $3^5 = 5$ in $Z_7$ by the computations we made above. Using Equation 2.12 twice, this gives us

$$\begin{aligned} 5^6 \bmod 7 &= (3^5)^6 \bmod 7 \\ &= 3^{5 \cdot 6} \bmod 7 \\ &= 3^{6 \cdot 5} \bmod 7 \\ &= (3^6)^5 = 1^5 = 1 \end{aligned}$$

in $Z_7$. Finally, since $-1 \bmod 7 = 6$, Lemma 2.3 tells us that $6^6 \bmod 7 = (-1)^6 \bmod 7 = 1$. Thus the sixth power of each element of $Z_7$ is 1.

In Exercise 2.3-3 we see that

$$\begin{aligned} 1 \cdot_7 2 &= 1 \cdot 2 \bmod 7 &= 2 \\ 2 \cdot_7 2 &= 2 \cdot 2 \bmod 7 &= 4 \\ 3 \cdot_7 2 &= 3 \cdot 2 \bmod 7 &= 6 \\ 4 \cdot_7 2 &= 4 \cdot 2 \bmod 7 &= 1 \\ 5 \cdot_7 2 &= 5 \cdot 2 \bmod 7 &= 3 \\ 6 \cdot_7 2 &= 6 \cdot 2 \bmod 7 &= 5. \end{aligned}$$

Thus these numbers are a permutation of the set $\{1, 2, 3, 4, 5, 6\}$. Similarly,

$$\begin{aligned} 1 \cdot_7 3 &= 1 \cdot 3 \bmod 7 &= 3 \\ 2 \cdot_7 3 &= 2 \cdot 3 \bmod 7 &= 6 \end{aligned}$$

$$\begin{aligned} 3 \cdot_7 3 &= 3 \cdot 3 \bmod 7 &= 2 \\ 4 \cdot_7 3 &= 4 \cdot 3 \bmod 7 &= 5 \\ 5 \cdot_7 3 &= 5 \cdot 3 \bmod 7 &= 1 \\ 6 \cdot_7 3 &= 6 \cdot 3 \bmod 7 &= 4. \end{aligned}$$

Again we get a permutation of $\{1, 2, 3, 4, 5, 6\}$.

In Exercise 2.3-4 we are asked whether this is always the case. Notice that since 7 is a prime, by Corollary 2.17, each nonzero number between 1 and 6 has a mod 7 multiplicative inverse $a^{-1}$. Thus if $i$ and $j$ were integers in $Z_7$ with $i \cdot_7 a = j \cdot_7 a$, we multiply mod 7 on the right by $a^{-1}$ to get

$$(i \cdot_7 a) \cdot_7 a^{-1} = (j \cdot_7 a) \cdot_7 a^{-1}.$$

After using the associative law we get

$$i \cdot_7 (a \cdot_7 a^{-1}) = j \cdot_7 (a \cdot_7 a^{-1}). \tag{2.13}$$

Since $a \cdot_7 a^{-1} = 1$, Equation 2.13 simply becomes $i = j$. Thus, we have shown that the only way for $i \cdot_7 a$ to equal $j \cdot_7 a$ is for $i$ to equal $j$. Therefore, all the values $i \cdot_7 a$ for $i = 1, 2, 3, 4, 5, 6$ must be different. Since we have six different values, all between 1 and 6, we have that the values $ia$ for $i = 1, 2, 3, 4, 5, 6$ are a permutation of $\{1, 2, 3, 4, 5, 6\}$.

As you can see, the only fact we used in our analysis of Exercise 2.3-4 is that if $p$ is a prime, then any number between 1 and $p-1$ has a multiplicative inverse in $Z_p$. In other words, we have really proved the following lemma.

**Lemma 2.20** *Let $p$ be a prime number. For any fixed nonzero number $a$ in $Z_p$, the numbers $(1 \cdot a) \bmod p$, $(2 \cdot a) \bmod p$, ..., $((p-1) \cdot a) \bmod p$, are a permutation of the set $\{1, 2, \cdots, p-1\}$.*

With this lemma in hand, we can prove a famous theorem that explains the phenomenon we saw in Exercise 2.3-2.

### Fermat's Little Theorem

**Theorem 2.21** *(Fermat's Little Theorem). Let $p$ be a prime number. Then $a^{p-1} \bmod p = 1$ in $Z_p$ for each nonzero $a$ in $Z_p$.*

**Proof:** Since $p$ is a prime, Lemma 2.20 tells us that the numbers $1 \cdot_p a$, $2 \cdot_p a$, ..., $(p-1) \cdot_p a$, are a permutation of the set $\{1, 2, \cdots, p-1\}$. But then

$$1 \cdot_p 2 \cdot_p \cdots \cdot_p (p-1) = (1 \cdot_p a) \cdot_p (2 \cdot_p a) \cdot_p \cdots \cdot_p ((p-1) \cdot_p a).$$

Using the commutative and associative laws for multiplication in $Z_p$ and Equation 2.10, we get

$$1 \cdot_p 2 \cdot_p \cdots \cdot_p (p-1) = 1 \cdot_p 2 \cdot_p \cdots \cdot_p (p-1) \cdot_p (a^{p-1} \bmod p).$$

Now we multiply both sides of the equation by the multiplicative inverses in $Z_p$ of 2, 3, ..., $p-1$ and the left hand side of our equation becomes 1 and the right hand side becomes $a^{p-1} \bmod p$. But this is exactly the conclusion of our theorem. ■

**Corollary 2.22** *(Fermat's Little Theorem, version 2) For every positive integer $a$, and prime $p$, if $a$ is not a multiple of $p$,*

$$a^{p-1} \bmod p = 1.$$

**Proof:** This is a direct application of Lemma 2.3, because if we replace $a$ by $a \bmod p$, then Theorem 2.21 applies.

### The RSA Cryptosystem

Fermat's Little Theorem is at the heart of the RSA cryptosystem, a system that allows Bob to tell the world a way that they can encode a message to send to him so that he and only he can read it. In other words, even though he tells everyone how to encode the message, nobody except Bob has a significant chance of figuring out what the message is from looking at the encoded message. What Bob is giving out is called a "one-way function." This is a function $f$ that has an inverse $f^{-1}$, but even though $y = f(x)$ is reasonably easy to compute, nobody but Bob (who has some extra information that he keeps secret) can compute $f^{-1}(y)$. Thus when Alice wants to send a message $x$ to Bob, she computes $f(x)$ and sends it to Bob, who uses his secret information to compute $f^{-1}(f(x)) = x$.

In the RSA cryptosystem Bob chooses two prime numbers $p$ and $q$ (which in practice each have at least a hundred digits) and computes the number $n = pq$. He also chooses a number $e \neq 1$ which need not have a large number of digits but is relatively prime to $(p-1)(q-1)$, so that it has an inverse $d$ in $Z_{(p-1)(q-1)}$, and he computes $d = e^{-1} \bmod (p-1)(q-1)$. Bob publishes $e$ and $n$. The number $e$ is called his *public key*. The number $d$ is called his *private key*.

To summarize what we just said, here is a pseudocode outline of what Bob does:

```
Bob's RSA key choice algorithm
(1)  Choose 2 large prime numbers p and q
(2)  n = pq
(3)  Choose e ≠ 1 so that e is relatively prime to (p − 1)(q − 1)
(4)  Compute d = e^{-1} mod (p − 1)(q − 1).
(5)  Publish e and n.
(6)  Keep d secret.
```

People who want to send a message $x$ to Bob compute $y = x^e \bmod n$ and send that to him instead. (We assume $x$ has fewer digits than $n$ so that it is in $Z_n$. If not, the sender has to break the message into blocks of size less than the number of digits of $n$ and send each block individually.)

To decode the message, Bob will compute $z = y^d \bmod n$.

We summarize this process in pseudocode below:

```
Alice-send-message-to-Bob(x)
Alice does:
(1)       Read the public directory for Bob's keys e and n.
(2)       Compute y = x^e mod n
```

(3) Send $y$ to Bob
Bob does:
(4) Receive $y$ from Alice
(5) Compute $z = y^d \bmod n$, using secret key $d$
(6) Read $z$

Each step in these algorithms can be computed using methods from this chapter. In Section 2.4, we will deal with computational issues in more detail.

In order to show that the RSA cryptosystem works, that is, that it allows us to encode and then correctly decode messages, we must show that $z = x$. In other words, we must show that, when Bob decodes, he gets back the original message. In order to show that the RSA cryptosystem is secure, we must argue that an eavesdropper, who knows $n$, $e$, and $y$, but does not know $p$, $q$ or $d$, can not easily compute $x$.

**Exercise 2.3-5** To show that the RSA cryptosystem works, we will first show a simpler fact. Why is

$$y^d \bmod p = x \bmod p?$$

Does this tell us what $x$ is?

Plugging in the value of $y$, we have

$$y^d \bmod p = x^{ed} \bmod p. \tag{2.14}$$

But, in Line 4 we chose $e$ and $d$ so that $e \cdot_m d = 1$, where $m = (p-1)(q-1)$. In other words,

$$ed \bmod (p-1)(q-1) = 1.$$

Therefore, for some integer $k$,

$$ed = k(p-1)(q-1) + 1.$$

Plugging this into Equation (2.14), we obtain

$$\begin{aligned} x^{ed} \bmod p &= x^{k(p-1)(q-1)+1} \bmod p \\ &= x^{(k(q-1))(p-1)} x \bmod p. \end{aligned} \tag{2.15}$$

But for any number $a$ which is not a multiple of $p$, $a^{p-1} \bmod p = 1$ by Fermat's Little Theorem (Theorem 2.22). We could simplify equation 2.15 by applying Fermat's Little Theorem to $x^{k(q-1)}$, as you will see below. However we can only do this when $x^{k(q-1)}$ is not a multiple of $p$. This gives us two cases, the case in which $x^{k(q-1)}$ is not a multiple of $p$ (we'll call this case 1) and the case in which $x^{k(q-1)}$ is a multiple of $p$ (we'll call this case 2). In case 1, we apply Equation 2.12 and Fermat's Little Theorem with $a$ equal to $x^{k(q-1)}$, and we have that

$$\begin{aligned} x^{(k(q-1))(p-1)} \bmod p &= \left(x^{k(q-1)}\right)^{(p-1)} \bmod p \\ &= 1. \end{aligned} \tag{2.16}$$

Combining equations 2.14, 2.15 and 2.17, we have that

$$y^d \bmod p = x^{k(q-1)(p-1)} x \bmod p = 1 \cdot x \bmod p = x \bmod p,$$

and hence $y^d \bmod p = x \bmod p$.

We still have to deal with case 2, the case in which $x^{k(q-1)}$ is a multiple of $p$. In this case $x$ is a multiple of $p$ as well since $x$ is an integer and $p$ is prime. Thus $x \bmod p = 0$. Combining Equations 2.14 and 2.15 with Lemma 2.3, we get

$$y^d \bmod p = (x^{k(q-1)(p-1)} \bmod p)(x \bmod p) = 0 = x \bmod p.$$

Hence in this case as well, we have $y^d \bmod p = x \bmod p$.

While this will turn out to be useful information, it does not tell us what $x$ is, however, because $x$ may or may not equal $x \bmod p$.

The same reasoning shows us that $y^d \bmod q = x \bmod q$. What remains is to show what these two facts tell us about $y^d \bmod pq = y \bmod n$, which is what Bob computes.

Notice that by Lemma 2.3 we have proved that

$$(y^d - x) \bmod p = 0 \tag{2.17}$$

and

$$(y^d - x) \bmod q = 0. \tag{2.18}$$

**Exercise 2.3-6** Write down an equation using only integers and addition, subtraction and multiplication in the integers, but perhaps more letters, that is equivalent to Equation 2.17, which says that $(y^d - x) \bmod p = 0$. (Do not use mods.)

**Exercise 2.3-7** Write down an equation using only integers and addition, subtraction and multiplication in the integers, but perhaps more letters, that is equivalent to Equation 2.18, which says that $(y^d - x) \bmod q = 0$. (Do not use mods.)

**Exercise 2.3-8** If a number is a multiple of a prime $p$ and a different prime $q$, then what else is it a multiple of? What does this tell us about $y^d$ and $x$?

The statement that $y^d - x \bmod p = 0$ is equivalent to the statement that $y^d - x = ip$ for some integer $i$. The statement that $y^d - x \bmod q = 0$ is equivalent to the statement that $y^d - x = jq$ for some integer $j$. If something is a multiple of the prime $p$ and the prime $q$, then it is a multiple of $pq$. Thus $(y^d - x) \bmod pq = 0$. Lemma 2.3 tells us that $(y^d - x) \bmod pq = (y^d \bmod pq - x) \bmod pq = 0$. But $x$ and $y^d \bmod pq$ are both integers between 0 and $pq - 1$, so their difference is between $-(pq - 1)$ and $pq - 1$. The only integer between these two values that is 0 mod $pq$ is zero itself. Thus $(y^d \bmod pq) - x = 0$. In other words,

$$\begin{aligned} x &= y^d \bmod pq \\ &= y^d \bmod n \ , \end{aligned}$$

which means that Bob will in fact get the correct answer.

**Theorem 2.23** *(Rivest, Shamir, and Adleman) The RSA procedure for encoding and decoding messages works correctly.*

**Proof:** Proved above.■

One might ask, given that Bob published $e$ and $n$, and messages are encrypted by computing $x^e \bmod n$, why can't any adversary who learns $x^e \bmod n$ just compute $e$th roots mod $n$ and break the code? At present, nobody knows a quick scheme for computing $e$th roots mod $n$, for an arbitrary $n$. Someone who does not know $p$ and $q$ cannot duplicate Bob's work and discover $x$. Thus, as far as we know, modular exponentiation is an example of a one-way function.

### The Chinese Remainder Theorem

The method we used to do the last step of the proof of Theorem 2.23 also proves a theorem known as the "Chinese Remainder Theorem."

**Exercise 2.3-9** For each number in $x \in Z_{15}$, write down $x \bmod 3$ and $x \bmod 5$. Is $x$ uniquely determined by these values? Can you explain why?

| $x$ | $x \bmod 3$ | $x \bmod 5$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 0 | 3 |
| 4 | 1 | 4 |
| 5 | 2 | 0 |
| 6 | 0 | 1 |
| 7 | 1 | 2 |
| 8 | 2 | 3 |
| 9 | 0 | 4 |
| 10 | 1 | 0 |
| 11 | 2 | 1 |
| 12 | 0 | 2 |
| 13 | 1 | 3 |
| 14 | 2 | 4 |

Table 2.2: The values of $x \bmod 3$ and $x \bmod 5$ for each $x$ between zero and 14.

As we see from Table 2.2, each of the $3 \cdot 5 = 15$ pairs $(i, j)$ of integers $i, j$ with $0 \leq i \leq 2$ and $0 \leq j \leq 4$ occurs exactly once as $x$ ranges through the fifteen integers from 0 to 14. Thus the function $f$ given by $f(x) = (x \bmod 3, x \bmod 5)$ is a one-to-one function from a fifteen element set to a fifteen element set, so each $x$ is uniquely determined by its pair of remainders.

The Chinese Remainder Theorem tells us that this observation always holds.

**Theorem 2.24** *(Chinese Remainder Theorem) If $m$ and $n$ are relatively prime integers and $a \in Z_m$ and $b \in Z_n$, then the equations*

$$x \bmod m = a \tag{2.19}$$

$$x \bmod n = b \tag{2.20}$$

*have one and only one solution for an integer $x$ between 0 and $mn - 1$.*

**Proof:** If we show that as $x$ ranges over the integers from 0 to $mn - 1$, then the ordered pairs $(x \bmod m, x \bmod n)$ are all different, then we will have shown that the function given by $f(x) = (x \bmod m, x \bmod n)$ is a one to one function from an $mn$ element set to an $mn$ element

set, so it is onto as well.[7] In other words, we will have shown that each pair of equations 2.19 and 2.20 has one and only one solution.

In order to show that $f$ is one-to-one, we must show that if $x$ and $y$ are different numbers between 0 and $mn-1$, then $f(x)$ and $f(y)$ are different. To do so, assume instead that we have an $x$ and $y$ with $f(x) = f(y)$. Then $x \bmod m = y \bmod m$ and $x \bmod n = y \bmod n$, so that $(x-y) \bmod m = 0$ and $(x-y) \bmod n = 0$. That is, $x-y$ is a multiple of both $m$ and $n$. Then as we show in Problem 11 in the problems at the end of this section, $x-y$ is a multiple of $mn$; that is, $x-y = dmn$ for some integer $d$. Since we assumed $x$ and $y$ were different, this means $x$ and $y$ cannot both be between 0 and $mn-1$ because their difference is $mn$ or more. This contradicts our hypothesis that $x$ and $y$ were different numbers between 0 and $mn-1$, so our assumption must be incorrect; that is $f$ must be one-to-one. This completes the proof of the theorem. ■

## Important Concepts, Formulas, and Theorems

1. *Exponentiation in $Z_n$.* For $a \in Z_n$, and a positive integer $j$:

$$a^j \bmod n = \underbrace{a \cdot_n a \cdot_n \cdots \cdot_n a}_{j \text{ factors}} .$$

2. *Rules of exponents.* For any $a \in Z_n$, and any nonnegative integers $i$ and $j$,

$$(a^i \bmod n) \cdot_n (a^j \bmod n) = a^{i+j} \bmod n$$

and

$$(a^i \bmod n)^j \bmod n = a^{ij} \bmod n.$$

3. *Multiplication by a fixed nonzero $a$ in $Z_p$ is a permutation.* Let $p$ be a prime number. For any fixed nonzero number $a$ in $Z_p$, the numbers $(1 \cdot a) \bmod p$, $(2 \cdot a) \bmod p$, ..., $((p-1) \cdot a) \bmod p$, are a permutation of the set $\{1, 2, \cdots, p-1\}$.

4. *Fermat's Little Theorem.* Let $p$ be a prime number. Then $a^{p-1} \bmod p = 1$ for each nonzero $a$ in $Z_p$.

5. *Fermat's Little Theorem, version 2.* For every positive integer $a$ and prime $p$, if $a$ is not a multiple of $p$, then

$$a^{p-1} \bmod p = 1.$$

6. *RSA cryptosystem.* (The first implementation of a public-key cryptosystem) In the RSA cryptosystem Bob chooses two prime numbers $p$ and $q$ (which in practice each have at least a hundred digits) and computes the number $n = pq$. He also chooses a number $e \neq 1$ which need not have a large number of digits but is relatively prime to $(p-1)(q-1)$, so that it has an inverse $d$, and he computes $d = e^{-1} \bmod (p-1)(q-1)$. Bob publishes $e$ and $n$. To send a message $x$ to Bob, Alice sends $y = x^e \bmod n$. Bob decodes by computing $y^d \bmod n$.

[7] If the function weren't onto, then because the number of pairs is the same as the number of possible $x$-values, two $x$ values would have to map to the same pair, so the function wouldn't be one-to-one after all.

7. *Chinese Remainder Theorem.* If $m$ and $n$ are relatively prime integers and $a \in Z_m$ and $b \in Z_n$, then the equations

$$\begin{aligned} x \bmod m &= a \\ x \bmod n &= b \end{aligned}$$

have one and only one solution for an integer $x$ between 0 and $mn - 1$.

## Problems

1. Compute the powers of 4 in $Z_7$. Compute the powers of 4 in $Z_{10}$. What is the most striking similarity? What is the most striking difference?

2. Compute the numbers $1 \cdot_{11} 5$, $2 \cdot_{11} 5$, $3 \cdot_{11} 5$, ..., $10 \cdot_{11} 5$. Do you get a permutation of the set $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$? Would you get a permutation of the set $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ if you used another nonzero member of of $Z_{11}$ in place of 5?

3. Compute the fourth power mod 5 of each element of $Z_5$. What do you observe? What general principle explains this observation?

4. The numbers 29 and 43 are primes. What is $(29-1)(43-1)$? What is $199 \cdot 1111$ in $Z_{1176}$? What is $(23^{1111})^{199}$ in $Z_{29}$? In $Z_{43}$? In $Z_{1247}$?

5. The numbers 29 and 43 are primes. What is $(29-1)(43-1)$? What is $199 \cdot 1111$ in $Z_{1176}$? What is $(105^{1111})^{199}$ in $Z_{29}$? In $Z_{43}$? In $Z_{1247}$? How does this answer the second question in Exercise 2.3-5?

6. How many solutions with $x$ between 0 and 34 are there to the system of equations

$$\begin{aligned} x \bmod 5 &= 4 \\ x \bmod 7 &= 5? \end{aligned}$$

What are these solutions?

7. Compute each of the following. Show or explain your work, and do not use a calculator or computer.

   (a) $15^{96}$ in $Z_{97}$

   (b) $67^{72}$ in $Z_{73}$

   (c) $67^{73}$ in $Z_{73}$

8. Show that in $Z_p$, with $p$ prime, if $a^i \bmod p = 1$, then $a^n \bmod p = a^{n \bmod i} \bmod p$.

9. Show that there are $p^2 - p$ elements with multiplicative inverses in $Z_{p^2}$ when $p$ is prime. If $x$ has a multiplicative inverse in $Z_p^2$, what is $x^{p^2-p} \bmod p^2$? Is the same statement true for an element without an inverse? (Working out an example might help here.) Can you find something (interesting) that is true about $x^{p^2-p}$ when $x$ does not have an inverse?

10. How many elements have multiplicative inverses in $Z_{pq}$ when $p$ and $q$ are primes?

11. In the paragraph preceding the proof of Theorem 2.23 we said that if a number is a multiple of the prime $p$ and the prime $q$, then it is a multiple of $pq$. We will see how that is proved here.

    (a) What equation in the integers does Euclid's extended GCD algorithm solve for us when $m$ and $n$ are relatively prime?

    (b) Suppose that $m$ and $n$ are relatively prime and that $k$ is a multiple of each one of them; that is, $k = bm$ and $k = cn$ for integers $b$ and $c$. If you multiply both sides of the equation in part (a) by $k$, you get an equation expressing $k$ as a sum of two products. By making appropriate substitutions in these terms, you can show that $k$ is a multiple of $mn$. Do so. Does this justify the assertion we made in the paragraph preceding the proof of Theorem 2.23?

12. The relation of "congruence modulo $n$" is the relation $\equiv$ defined by $x \equiv y \bmod n$ if and only if $x \bmod n = y \bmod n$.

    (a) Show that congruence modulo $n$ is an equivalence relation by showing that it defines a partition of the integers into equivalence classes.

    (b) Show that congruence modulo $n$ is an equivalence relation by showing that it is reflexive, symmetric, and transitive.

    (c) Express the Chinese Remainder theorem in the notation of congruence modulo $n$.

13. Write and implement code to do RSA encryption and decryption. Use it to send a message to someone else in the class. (You may use smaller numbers than are usually used in implementing the RSA algorithm for the sake of efficiency. In other words, you may choose your numbers so that your computer can multiply them without overflow.)

14. For some non-zero $a \in Z_p$, where $p$ is prime, consider the set

$$S = \{a^0 \bmod p,\ a^1 \bmod p,\ a^2 \bmod p,\ \ldots,\ a^{p-2} \bmod p,\ a^{p-1} \bmod p\},$$

    and let $s = |S|$. Show that $s$ is always a factor of $p - 1$.

15. Show that if $x^{n-1} \bmod n = 1$ for all integers $x$ that are not multiples of $n$, then $n$ is prime. (The slightly weaker statement that $x^{n-1} \bmod n = 1$ for all $x$ relatively prime to $n$, does not imply that $n$ is prime. There is a famous family of numbers called Carmichael numbers that are counterexamples.[8])

[8] See, for example, Cormen, Leiserson, Rivest, and Stein, cited earlier.

## 2.4 Details of the RSA Cryptosystem

In this section, we deal with some issues related to implementing the RSA cryptosystem: exponentiating large numbers, finding primes, and factoring.

### Practical Aspects of Exponentiation mod $n$

Suppose you are going to raise a 100 digit number $a$ to the $10^{120}$th power modulo a 200 digit integer $n$. Note that the exponent is a 121 digit number.

**Exercise 2.4-1** Propose an algorithm to compute $a^{10^{120}} \bmod n$, where $a$ is a 100 digit number and $n$ is a 200 digit number.

**Exercise 2.4-2** What can we say about how long this algorithm would take on a computer that can do one infinite precision arithmetic operation in constant time?

**Exercise 2.4-3** What can we say about how long this would take on a computer that can multiply integers in time proportional to the product of the number of digits in the two numbers, i.e. multiplying an $x$-digit number by a $y$-digit number takes roughly $xy$ time?

Notice that if we form the sequence $a$, $a^2$, $a^3$, $a^4$, $a^5$, $a^6$, $a^7$, $a^8$, $a^9$, $a^{10}$, $a^{11}$ we are modeling the process of forming $a^{11}$ by successively multiplying by $a$. If, on the other hand, we form the sequence $a$, $a^2$, $a^4$, $a^8$, $a^{16}$, $a^{32}$, $a^{64}$, $a^{128}$, $a^{256}$, $a^{512}$, $a^{1024}$, we are modeling the process of successive squaring, and in the same number of multiplications we are able to get $a$ raised to a four digit number. Each time we square we double the exponent, so every ten steps or so we will add three to the number of digits of the exponent. Thus in a bit under 400 multiplications, we will get $a^{10^{120}}$. This suggests that our algorithm should be to square $a$ some number of times until the result is almost $a^{10^{120}}$, and then multiply by some smaller powers of $a$ until we get exactly what we want. More precisely, we square $a$ and continue squaring the result until we get the largest $a^{2^{k_1}}$ such that $2^{k_1}$ is less than $10^{120}$, then multiply $a^{2^{k_1}}$ by the largest $a^{2^{k_2}}$ such that $2^{k_1} + 2^{k_2}$ is less than $10^{120}$ and so on until we have

$$10^{120} = 2^{k_1} + 2^{k_2} + \cdots + 2^{k_r}$$

for some integer $r$. (Can you connect this with the binary representation of $10^{120}$?) Then we get

$$a^{10^{120}} = a^{2^{k_1}} a^{2^{k_2}} \cdots a^{2^{k_r}}.$$

Notice that all these powers of $a$ have been computed in the process of discovering $k_1$. Thus it makes sense to save them as you compute them.

To be more concrete, let's see how to compute $a^{43}$. We may write $43 = 32 + 8 + 2 + 1$, and thus

$$a^{43} = a^{2^5} a^{2^3} a^{2^1} a^{2^0} \ . \tag{2.21}$$

So, we first compute $a^{2^0}, a^{2^1}, a^{2^2}, a^{2^3}, a^{2^4}, a^{2^5}$, using 5 multiplications. Then we can compute $a^{43}$, via equation 2.21, using 3 additional multiplications. This saves a large number of multiplications.

On a machine that could do infinite precision arithmetic in constant time, we would need about $\log_2(10^{120})$ steps to compute all the powers $a^{2^i}$, and perhaps equally many steps to do the multiplications of the appropriate powers. At the end we could take the result mod $n$. Thus the length of time it would take to do these computations would be more or less $2\log_2(10^{120}) = 240\log_2 10$ times the time needed to do one operation. (Since $\log_2 10$ is about 3.33, it will take at most 800 times the amount of time for one operation to compute $a^{10^{120}}$. )

You may not be used to thinking about how large the numbers get when you are doing computation. Computers hold fairly large numbers (4-byte integers in the range roughly $-2^{31}$ to $2^{31}$ are typical), and this suffices for most purposes. Because of the way computer hardware works, as long as numbers fit into one 4-byte integer, the time to do simple arithmetic operations doesn't depend on the value of the numbers involved. (A standard way to say this is to say that the time to do a simple arithmetic operation is constant.) However, when we talk about numbers that are much larger than $2^{31}$, we have to take special care to implement our arithmetic operations correctly, and also we have to be aware that operations are slower.

Since $2^{10} = 1024$, we have that $2^{31}$ is twice as big as $2^{30} = (2^{10})^3 = (1024)^3$ and so is somewhat more than two billion, or $2 \cdot 10^9$. In particular, it is less than $10^{10}$. Since $10^{120}$ is a one followed by 120 zeros, raising a positive integer other than one to the $10^{120}$th power takes us completely out of the realm of the numbers that we are used to making exact computations with. For example, $10^{(10^{120})}$ has 119 more zeros following the 1 in the exponent than does $10^{10}$.

It is accurate to assume that when multiplying large numbers, the time it takes is roughly proportional to the product of the number of digits in each. If we computed our 100 digit number to the $10^{120}$th power, we would be computing a number with more than $10^{120}$ digits. We clearly do **not** want to be doing computation on such numbers, as our computer cannot even store such a number!

Fortunately, since the number we are computing will ultimately be taken modulo some 200 digit number, we can make all our computations modulo that number. (See Lemma 2.3.) By doing so, we ensure that the two numbers we are multiplying together have at most 200 digits, and so the time needed for the problem proposed in Exercise 2.4-1 would be a proportionality constant times 40,000 times $\log_2(10^{120})$ times the time needed for a basic operation plus the time needed to figure out which powers of $a$ are multiplied together, which would be quite small in comparison.

This algorithm, on 200 digit numbers, could be on the order of a million times slower than on simple integers.[9] This is a noticeable effect and if you use or write an encryption program, you can see this effect when you run it. However, we can still typically do this calculation in less than a second, a small price to pay for secure communication.

### How long does it take to use the RSA Algorithm?

Encoding and decoding messages according to the RSA algorithm requires many calculations. How long will all this arithmetic take? Let's assume for now that Bob has already chosen $p$, $q$, $e$, and $d$, and so he knows $n$ as well. When Alice wants to send Bob the message $x$, she

[9] If we assume that we can multiply four digit integers exactly but not five digit numbers exactly, then efficiently multiplying two 200 digit numbers is like multiplying 50 integers times 50 integers, or 2500 products, and $\log_2(10^{120}) \approx \log_2((2^10)^{40} = \log_2(2^{400}) = 400$, so we would have something like million steps, each equivalent to multiplying two integers, in executing our algorithm.

sends $x^e \bmod n$. By our analyses in Exercise 2.4-2 and Exercise 2.4-3 we see that this amount of time is more or less proportional to $\log_2 e$, which is itself proportional to the number of digits of $e$, though the first constant of proportionality depends on the method our computer uses to multiply numbers. Since $e$ has no more than 200 digits, this should not be too time consuming for Alice if she has a reasonable computer. (On the other hand, if she wants to send a message consisting of many segments of 200 digits each, she might want to use the RSA system to send a key for another simpler (secret key) system, and then use that simpler system for the message.)

It takes Bob a similar amount of time to decode, as he has to take the message to the $d$th power, mod $n$.

We commented already that nobody knows a fast way to find $x$ from $x^e \bmod n$. In fact, nobody knows that there isn't a fast way either, which means that it is possible that the RSA cryptosystem could be broken some time in the future. We also don't know whether extracting $e$th roots mod $n$ is in the class of #P-complete problems, an important family of problems with the property that a reasonably fast solution of any one of them will lead to a reasonably fast solution of any of them. We do know that extracting $e$th roots is no harder than these problems, but it may be easier.

However, here someone is not restricted to extracting roots to discover $x$. Someone who knows $n$ and knows that Bob is using the RSA system, could presumably factor $n$, discover $p$ and $q$, use the extended GCD algorithm to compute $d$ and then decode all of Bob's messages. However, nobody knows how to factor integers quickly either. Again, we don't know if factoring is #P-complete, but we do know that it is no harder than the #P-complete problems. Thus here is a second possible way around the RSA system. However, enough people have worked on the factoring problem that most compputer scientists are confident that it is in fact difficult, in which case the RSA system is safe, as long as we use keys that are long enough.

### How hard is factoring?

**Exercise 2.4-4** Factor 225,413. (The idea is to try to do this without resorting to computers, but if you give up by hand and calculator, using a computer is fine.)

With current technology, keys with roughly 100 digits are not that hard to crack. In other words, people can factor numbers that are roughly 100 digits long, using methods that are a little more sophisticated than the obvious approach of trying all possible divisors. However, when the numbers get long, say over 120 digits, they become very hard to factor. The record, as of the year 2000, for factoring is a roughly 155-digit number. To factor this number, thousands of computers around the world were used, and it took several months. So given the current technology, RSA with a 200 digit key seems to be very secure.

### Finding large primes

There is one more issue to consider in implementing the RSA system for Bob. We said that Bob chooses two primes of about a hundred digits each. But how does he choose them? It follows from some celebrated work on the density of prime numbers that if we were to choose a number $m$ at random, and check about $\log_e(m)$ numbers around $m$ for primality, we would expect that one of these numbers was prime. Thus if we have a reasonably quick way to check whether a

number is prime, we shouldn't have to guess too many numbers, even with a hundred or so digits, before we find one we can show is prime.

However, we have just mentioned that nobody knows a quick way to find any or all factors of a number. The standard way of proving a number is prime is by showing that it and 1 are its only factors. For the same reasons that factoring is hard, the simple approach to primality testing, test all possible divisors, is much too slow. If we did not have a faster way to check whether a number is prime, the RSA system would be useless.

In August of 2002, Agrawal, Kayal and Saxena announced an algorithm for testing whether an integer $n$ is prime which they can show takes no more than the twelveth power of the number of digits of $n$ to determine whether $n$ is prime, and in practice seems to take significantly less time. While the algorithm requires more than the background we are able to provide in this book, its description and the proof that it works in the specified time uses only results that one might find in an undergraduate abstract algebra course and an undergraduate number theory course! The central theme of the algorithm is the use of a variation of Fermat's Little Theorem.

In 1976 Miller[10] was able to use Fermat's Little Theorem to show that if a conjecture called the "Extended Reiman Hypothesis" was true, then an algorithm he developed would determine whether a number $n$ was prime in a time bounded above by a polynomial in the number of digits of $n$. In 1980 Rabin[11] modified Miller's method to get one that would determine in polynomial time whether a number was prime without the extra hypothesis, but with a probability of error that could be made as small a positive number as one might desire, but not zero. We describe the general idea behind all of these advances in the context of what people now call the Miller-Rabin primality test. As of the writing of this book, variations on this kind of algorithm are used to provide primes for cryptography.

We know, by Fermat's Little Theorem, that in $Z_p$ with $p$ prime, $x^{p-1} \bmod p = 1$ for every $x$ between 1 and $p-1$. What about $x^{m-1}$, in $Z_m$, when $m$ is not prime?

**Exercise 2.4-5** Suppose $x$ is a member of $Z_m$ that has no multiplicative inverse. Is it possible that $x^{n-1} \bmod n = 1$?

We answer the question of the exercise in our next lemma.

**Lemma 2.25** *Let $m$ be a non-prime, and let $x$ be a number in $Z_m$ which has no multiplicative inverse. Then $x^{m-1} \bmod m \neq 1$.*

**Proof:** Assume, for the purpose of contradiction, that

$$x^{m-1} \bmod m = 1.$$

Then

$$x \cdot x^{m-2} \bmod m = 1.$$

But then $x^{m-2} \bmod m$ is the inverse of $x$ in $Z_m$, which contradicts the fact that $x$ has no multiplicative inverse. Thus it must be the case that $x^{m-1} \bmod m \neq 1$. ■

[10] G.L. Miller. "Riemann's Hypothesis and tests for primality," J. Computer and Systems Science **13**, 1976, pp 300-317.

[11] M. O. Rabin. "Probabilistic algorithm for testing primality." Journal of Number Theory, **12**, 1980. pp 128-138.

This distinction between primes and non-primes gives the idea for an algorithm. Suppose we have some number $m$, and are not sure whether it is prime or not. We can run the following algorithm:

```
(1)  PrimeTest(m)
(2)  choose a random number x, 2 ≤ x ≤ m − 1.
(3)  compute y = x^{m−1} mod m
(4)  if (y = 1)
(5)       output `` m might be prime''
(6)  else
(7)       output ``m is definitely not prime''
```

Note the asymmetry here. If $y \neq 1$, then $m$ is definitely not prime, and we are done. On the other hand, if $y = 1$, then the $m$ might be prime, and we probably want to do some other calculations. In fact, we can repeat the algorithm Primetest(m) for $t$ times, with a different random number $x$ each time. If on any of the $t$ runs, the algorithm outputs "$m$ is definitely not prime", then the number $m$ is definitely not prime, as we have an $x$ for which $x^{m-1} \neq 1$. On the other hand, if on all $t$ runs, the algorithm Primetest(m) outputs "$m$ might be prime", then, with reasonable certainty, we can say that the number $m$ is prime. This is actually an example of a *randomized algorithm*; we will be studying these in greater detail later in the course. For now, let's informally see how likely it is that we make a mistake.

We can see that the chance of making a mistake depends on, for a particular non-prime $m$, exactly how many numbers $a$ have the property that $a^{m-1} = 1$. If the answer is that very few do, then our algorithm is very likely to give the correct answer. On the other hand, if the answer is most of them, then we are more likely to give an incorrect answer.

In Exercise 12 at the end of the section, you will show that the number of elements in $Z_m$ without inverses is at least $\sqrt{m}$. In fact, even many numbers that do have inverses will also fail the test $x^{m-1} = 1$. For example, in $Z_{12}$ only 1 passes the test while in $Z_{15}$ only 1 and 14 pass the test. ($Z_{12}$ really is not typical; can you explain why? See Problem 13 at the end of this section for a hint.)

In fact, the Miller-Rabin algorithm modifies the test slightly (in a way that we won't describe here[12]) so that for any non-prime $m$, at least half of the possible values we could choose for $x$ will fail the modified test and hence will show that $m$ is composite. As we will see when we learn about probability, this implies that if we repeat the test $t$ times, and assert that an $x$ which passes these $t$ tests is prime, the probability of being wrong is actually $2^{-t}$. So, if we repeat the test 10 times, we have only about a 1 in a thousand chance of making a mistake, and if we repeat it 100 times, we have only about a 1 in $2^{100}$ (a little less than one in a nonillion) chance of making a mistake!

Numbers we have chosen by this algorithm are sometimes called *pseudoprimes*. They are called this because they are very likely to be prime. In practice, pseudoprimes are used instead of primes in implementations of the RSA cryptosystem. The worst that can happen when a pseudoprime is not prime is that a message may be garbled; in this case we know that our pseudoprime is not really prime, and choose new pseudoprimes and ask our sender to send the

[12]See, for example, Cormen, Leiserson, Rivest and Stein, Introduction to Algorithms, McGraw Hill/MIT Press, 2002

message again. (Note that we do not change $p$ and $q$ with each use of the system; unless we were to receive a garbled message, we would have no reason to change them.)

A number theory theorem called the Prime Number Theorem tells us that if we check about $\log_e n$ numbers near $n$ we can expect one of them to be prime. A $d$ digit number is at least $10^{d-1}$ and less than $10^d$, so its natural logarithm is between $(d-1)\log_e 10$ and $d\log_e 10$. If we want to find a $d$ digit prime, we can take any $d$ digit number and test about $d\log_e 10$ numbers near it for primality, and it is reasonable for us to expect that one of them will turn out to be prime. The number $\log_e 10$ is 2.3 to two decimal places. Thus it does not take a really large amount of time to find two prime numbers with a hundred (or so) digits each.

## Important Concepts, Formulas, and Theorems

1. *Exponentiation.* To perform exponentiation mod $n$ efficiently, we use repeated squaring, and take mods after each arithmetic operation.

2. *Security of RSA.* The security of RSA rests on the fact that no one has developed an efficient algorithm for factoring, or for finding $x$, given $x^e \bmod n$.

3. *Fermat's Little Theorem does not hold for composites.* Let $m$ b e a non-prime, and let $x$ be a number in $Z_n$ which has no multiplicative inverse. Then $x^{m-1} \bmod m \neq 1$.

4. *Testing numbers for primality.* The randomized Miller-Rabin algorithm will tell you almost surely if a given number is prime.

5. *Finding prime numbers.* By applying the randomized Miller-Rabin to $d\ln 10$ (which is about $2.3d$) numbers with $d$ digits, you can expect to find at least one that is prime.

## Problems

1. What is $3^{1024}$ in $Z_7$? (This is a straightforward problem to do by hand.)

2. Suppose we have computed $a^2$, $a^4$, $a^8$, $a^{16}$ and $a^{32}$. What is the most efficient way for us to compute $a^{53}$?

3. A gigabyte is one billion bytes; a terabyte is one trillion bytes. A byte is eight bits, each a zero or a 1. Since $2^{10} = 1024$, which is about 1000, we can store about three digits (any number between 0 and 999) in ten bits. About how many decimal digits could we store in a five gigabytes of memory? About how many decimal digits could we store in five terabytes of memory? How does this compare to the number $10^{120}$? To do this problem it is reasonable to continue to assume that 1024 is about 1000.

4. Find all numbers $a$ different from 1 and $-1$ (which is the same as 8) such that $a^8 \bmod 9 = 1$.

5. Use a spreadsheet, programmable calculator or computer to find all numbers $a$ different from 1 and $-1 \bmod 33 = 32$ with $a^{32} \bmod 33 = 1$. (This problem is relatively straightforward to do with a spreadsheet that can compute mods and will let you "fill in" rows and columns with formulas. However you do have to know how to use the spreadsheet in this way to make it strightforward!)

6. How many digits does the $10^{120}$th power of $10^{100}$ have?

7. If $a$ is a 100 digit number, is the number of digits of $a^{10^{120}}$ closer to $10^{120}$ or $10^{240}$? Is it a lot closer? Does the answer depend on what $a$ actually is rather than the number of digits it has?

8. Explain what our outline of the solution to Exercise 2.4-1 has to do with the binary representation of $10^{120}$.

9. Give careful pseudocode to compute $a^x \bmod n$. Make your algorithm as efficient as possible. You may use right shift n in your algorithm.

10. Suppose we want to compute $a^{e_1 e_2 \cdots e_m} \bmod n$. Discuss whether it makes sense to reduce the exponents mod $n$ as we compute their product. In particular, what rule of exponents would allow us to do this, and do you think this rule of exponents makes sense?

11. Number theorists use $\varphi(n)$ to stand for the number of elements of $Z_n$ that have inverses. Suppose we want to compute $a^{e_1 e_2 \cdots e_m} \bmod n$. Would it make sense for us to reduce our exponents mod $\varphi(n)$ as we compute their product? Why?

12. Show that if $m$ is not prime, then at least $\sqrt{m}$ elements of $Z_m$ do not have multiplicative inverses.

13. Show that in $Z_{p+1}$, where $p$ is prime, only one element passes the primality test $x^{m-1} = 1 \pmod m$. (In this case, $m = p + 1$.)

14. Suppose for RSA, $p = 11$, $q = 19$, and $e = 7$. What is the value of $d$? Show how to encrypt the message 100, and then show how to decrypt the resulting message.

15. Suppose for applying RSA, $p = 11$, $q = 23$, and $e = 13$. What is the value of $d$? Show how to encrypt the message 100 and then how to decrypt the resulting message.

16. A digital signature is a way to securely sign a document. That is, it is a way to put your "signature" on a document so that anyone reading it knows that it is you who have signed it, but no one else can "forge" your signature. The document itself may be public; it is your signature that we are trying to protect. Digital signatures are, in a way, the opposite of encryption, as if Bob wants to sign a message, he first applies his signature to it (think of this as encryption) and then the rest of the world can easily read it (think of this as decryption). Explain, in detail, how to achieve digital signatures, using ideas similar to those used for RSA. In particular, anyone who has the document and has your signature of the document (and knows your public key) should be able to determine that you signed it.