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, 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 $0 \leq r < n$. Furthermore, r is equal to $m \bmod n$.*

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

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 $m \bmod n$ to implement a Caeser cipher?

[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 2.1-3 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 $(-m) \bmod n = -(m \bmod n)$ is not necessarily true. Note that -3 mod 7 is 4 also. Furthermore, $-10 + 3 \bmod 7 = 0$, 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      : 335017873005926455916467429642711822266728 11001
reverse, 5rep : 10011827662228117246924764619555546295500378710533
```

As Problem 18 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

$$M = S_A(P_A(M)) = P_A(S_A(M)), \text{ and} \tag{2.2}$$

$$M = S_B(P_B(M)) = P_B(S_B(M)). \tag{2.3}$$

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.

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

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

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

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! Some closely related equations are true as we shall soon see.

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.

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

■

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 2×10^9 or 9×10^{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 11 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] 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

[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.

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 $x = 7$.

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. If instead of encoding by adding a mod n, we encoded by multiplying by a mod n, we would foil the adversary's ability to subtract each two of three messages from each other to get a difference of two unencoded messages. (This doesn't give us a great secret key cryptosystem, even if both a and n are secret, but it does give us a better one.) 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 partner 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 partner 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 the unique solution in Z_{12} to the equation $5 \cdot_{12} x = 11$. Thus in this case we can correctly decode the 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. 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. How many places has each letter been shifted in the Caesar cipher used to encode the message XNQQD RJXXFLJ?

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

4. 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?

5. 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?

6. 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.

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

8. 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.

9. 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}?

10. 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.

11. 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.)

12. 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?

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

14. Prove the equalities for multiplication in Lemma 2.3.

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

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

17. 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.

18. 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.