Cryptography Module (title TBD, perhaps Lqwurgxfwlrq wr Fubswrjudskb)

Authors: Jim Kupetz and Steven J. Miller (sjm1@williams.edu)

The purpose of this module is to introduce students and teachers to the theory and practice of cryptography. As this is a vast subject with numerous applications, any introduction necessarily has to be brief and must concentrate on a few jewels of the subject. More material and exercises are included than can be covered in a week in order to allow teachers flexibility in terms of topics and depths. In particular, the homework problems range from straightforward calculations to some very challenging exercises included for advanced students.

Cryptographic systems and ideas covered:

1. Caesar ciphers (Day 1): Simple to state and implement, they are one of the earliest cipher schemes. These work by shifting each letter of the alphabet a fixed amount to get the encoded letter, noting that we need to wrap around (so Z shifted by two is B).
2. Substitution ciphers (Day 2): These are a natural generalization of Caesar ciphers, where instead of shifting each letter by the same amount we randomly shuffle all 26 letters.
3. Frequency analysis (Day 2): Using letter frequencies we can easily break substitution ciphers.
4. RSA (Day 3): For years RSA was the gold standard for encryption. It is significantly harder to implement than substitution ciphers, but it is easily done with modern computers.
5. Trapdoor problems (Day 3): These are problems that are easy to solve if you know an extra piece of information, but believed to be hard otherwise. Many cryptography systems are based on such problems.
6. Digital signatures (Day 3): Just because a letter is signed `Alice' does not mean it comes from Alice. It is imperative that the identity of the sender be verified. If not, e-commerce collapses, as we need to make sure our funds go to the correct merchant.
7. Disinformation (Day 5): Usually we think of encrypting information to prevent others from knowing the message; however, there are many situations where it is desirable to mislead.

Mathematics covered:

1. Clock arithmetic (Day 1): Clock (or modulo) arithmetic generalizes our notion of addition on a standard clock. This topic is of immense use, both in understanding classical ciphers (such as the Caesar cipher) and powerful modern ones such as RSA.
2. Dirichlet's Pidgeon-hole (or Box) Principle (Day 2): This method is often used to prove something happens, although frequently it is non-constructive.
3. Complexity Analysis (Day 2): Estimating how long a calculation will take. This is especially important in cryptography, as it gives a sense of how secure a code is.
4. Algorithms (Day 3): step by step procedures to solve a problem.
5. Fast exponentiation (Days 3 and 4): Frequently there are significantly better ways to attack a problem than the `obvious' brute force approach one is used to. A terrific example involves computing high powers of numbers; while initially it appears we need n-1 multiplications to find x^n, using fast exponentiation we can do this with at most 2 $\log_2(n)$ multiplications. This is an enormous savings, and makes systems such as RSA practical. A key component of this algorithm

is the binary expansion of a number, which is a lot more convenient than the standard decimal expansion.

6. Prime numbers (Day 3): The primes are the building blocks of the integers, as every integer can be written as a product of primes. They also play a key role in the RSA system.
7. Euclidean and Generalized Euclidean Algorithm (Days 3 and 4): The details of these are not covered due to lack of time, but on-line implementations are mentioned.

The major pre-requisite for this module is a comfort with elementary algebra. Many of the problems and topics involve taking standard algebra concepts students have seen and generalizing these to a new and somewhat unfamiliar setting. Such generalizations occur all the time in mathematics, and one of the goals of these lectures is to help introduce students to this process.

Cryptography is almost surely nearly as old as communicating, as there are many reasons to wish certain information to be secure. Thus many of the issues we face today have been faced and the source of study for hundreds, if not thousands, of years. Cryptography is thus simultaneously an old subject and a very dynamic modern pursuit. It is hard to go an entire day without seeing some occurrence or application of the subject, ranging from national defense to e-commerce and on-line transactions.

In order to keep the module accessible to a large audience, some important mathematics has deliberately been omitted from the lectures. The most prominent omission concerns the Euclidean algorithm to find greatest common divisors. While this is a key input in RSA, it can easily be used as a black box result, and students who wish to know more can read online (note this would make an excellent choice for supplementary material for Day 5, if additional topics are needed). It might be worthwhile putting this omission in a broader context. For example, students are familiar with the square-root function and the square-root key on a calculator. With a little thought, they might come up with an algorithm to compute these (keep adding digits and correct if the guess is too high or too low); however, this algorithm is slow and unwieldy. There are great methods to find square-roots. An excellent one is Newton's method, which generates the sequence $x_{n+1} = (1/2)(x_n + a/x_n)$. Taking $x_0 = a$ gives a sequence which rapidly converges to $\sqrt{a}$. Note that students don't need to understand how the square-root key works to use calculators to find square-roots. The *concept* of the square-root is fairly elementary; the difficulty is the practical implementation. So too is it with the Euclidean algorithm. Most students should be able to grasp what the greatest common divisor of x and y is, and understand how to find it through a brute force attack (try all possible numbers less than x and y). Thus, while it is nice to know *how* to find the greatest common divisor efficiently, at this level it should be fine to accept the *existence* of such an algorithm and move on to the rest of RSA. This point should be emphasized to the students so they can put the decisions of what material to include and what to omit in the proper context.

Related to the above discussion is the notion of efficiency. It is not enough to be able to do a calculation; often these calculations must be done in real-time to be useful. This is a major theme of the module. The primary example is fast exponentiation. We have chosen to include full details, making this our representative example of efficiency. The results of this method are absolutely staggering, and should be emphasized to students. The very simple idea of using binary expansions to help with exponentiation can reduce problems from having to do 10^{200} multiplications to less than 500! These numbers should be put into perspective (such a calculation is done earlier in the module, converting such numbers into language involving the age of the universe).

The amount of material that can be covered in a week will vary greatly depending on the background of the students and their comfort with mathematics. So long as the students have some basic algebra, all of the material from the first three `days’ should be coverable in a week long unit, with material drawn from `days’ four and five as time permits.

Lqwurgxfwlrq wr Fubswrjudskb

(Introduction to Cryptography)

Jim Kupetz and Steven J Miller

Day 1: Wkh Fdhvdu Flskhu (The Caesar Cipher)

Teacher's Note: Depending on your class and your teaching style, you may want to introduce Lesson 1 with Activity 1 or introduce the Caesar cipher first and then try the activity. It is the authors' intention that the students take from the activity that it is easy to encode and decode with the key, but not as easy to decode without it. Some students may reach the conclusion that they can decode, as long as they can discover the key. This is also a good conclusion because it *reinforces* the idea that a stronger encryption method is required to keep information safe.

Activity 1: Cut Handout 1 into slips of paper for the students. The teacher can either assign slips (and therefore roles) or can randomly pass out slips. Each slip will have a group number, a role and a code sheet. There are four possible roles; coder, decoder, messenger and spy. Messengers only have code sheets to make it difficult for students to tell the difference between spies and messengers. Each coder should also get a "quote sheet" to choose a quote to encode. After encoding the quote, hand the messages (not the quote, only the encoded part) to the messengers, who will take it to the decoders. Each of the messengers will have one minute to try to spy on the message by decoding the message. Messenger 3's slip of paper has the decoding key, making messenger 3 the spy. It should be easy for messenger 3 to decode the message.

Teacher Note: Although the page of papers we pre-generated made Messenger 3 the spy, the teacher may want to change it around for different sections of the same course, or even make multiple spies by giving more than one messenger the key.

Quotes to be encoded: **(or create your own slogans using your school's motto, lines from the alma mater, or your personal favorite quotes.)**

Pure mathematics is, in its way, the poetry of logical ideas. - Albert Einstein

The human mind has never invented a labor-saving machine equal to algebra.

A mathematician is a device for turning coffee into theorems. - Paul Erdos

You may be an engineer if your idea of good interpersonal communication means getting the decimal point in the right place. (http://www.quotegarden.com/math.html)

Mathematics is the art of giving the same name to different things. - Henri Poincare

Television is something the Russians invented to destroy American education. - Paul Erdos

Rest satisfied with doing well, and leave others to talk of you as they will. - Pythagoras (http://www.brainyquote.com)

The quotes above are long, and will take time to encode and decode. Here are some shorter suggestions:

The Eagle has landed. (Neil Armstrong)

Fortune favors the brave. (Virgil)

A joke is a very serious thing. (Winston Churchill)

My life is my message. (Ghandi)

There is no next time. It is now or never. (Unknown)

Be yourself. Everyone else is taken. (Anonymous)

Failure is success in progression. (Albert Einstein)

Luke, I am your father. (Darth Vader)

Do, or do not. There is no try. (Master Yoda)

Handout 1 – I've Got a Secret, but I'll Share it with You

ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 1	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 2
EFGHIJKLMNOPQRSTUVWXYZABCD	Coder	MNOPQRSTUVWXYZABCDEFGHIJKL	Coder
EFGHIJKLMNOPQRSTUVWXYZABCD	Group 1	MNOPQRSTUVWXYZABCDEFGHIJKL	Group 2
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Decoder	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Decoder
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 1	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 2
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Messenger	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Messenger
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 3	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 4
HIJKLMNOPQRSTUVWXYZABCDEFG	Coder	OPQRSTUVWXYZABCDEFGHIJKLMN	Coder
HIJKLMNOPQRSTUVWXYZABCDEFG	Group 3	OPQRSTUVWXYZABCDEFGHIJKLMN	Group 4
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Decoder	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Decoder
HIJKLMNOPQRSTUVWXYZABCDEFG	Group 3	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 4
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Messenger	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Messenger
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 5	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 6
QRSTUVWXYZABCDEFGHIJKLMNOP	Coder	UVWXYZABCDEFGHIJKLMNOPQRST	Coder
QRSTUVWXYZABCDEFGHIJKLMNOP	Group 5	UVWXYZABCDEFGHIJKLMNOPQRST	Group 6
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Decoder	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Decoder
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 5	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 6
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Messenger	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Messenger
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 7	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 8
IJKLMNOPQRSTUVWXYZABCDEFGH	Coder	RSTUVWXYZABCDEFGHIJKLMNOPQ	Coder
IJKLMNOPQRSTUVWXYZABCDEFGH	Group 7	RSTUVWXYZABCDEFGHIJKLMNOPQ	Group 8
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Decoder	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Decoder
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 7	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 8
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Messenger	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Messenger
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 9	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 10
WXYZABCDEFGHIJKLMNOPQRSTUV	Coder	DEFGHIJKLMNOPQRSTUVWXYZABC	Coder
WXYZABCDEFGHIJKLMNOPQRSTUV	Group 9	DEFGHIJKLMNOPQRSTUVWXYZABC	Group 10
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Decoder	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Decoder
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 9	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 10
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Messenger	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Messenger
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 11	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 12
KLMNOPQRSTUVWXYZABCDEFGHIJ	Coder	FGHIJKLMNOPQRSTUVWXYZABCDE	Coder
KLMNOPQRSTUVWXYZABCDEFGHIJ	Group 11	FGHIJKLMNOPQRSTUVWXYZABCDE	Group 12
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Decoder	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Decoder
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 11	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Group 12
ABCDEFGHIJKLMNOPQRSTUVWXYZ	Messenger	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Messenger

Julius Caesar is known for many things. He conquered Gaul (France), invaded Britain, and became dictator of Rome. Numerous words for king come from his name (czar in Russian and kaisar in German), and for many years the king of diamonds was given his name. As important as all these contributions are, for us there's a more important one: the Caesar cipher in cryptography.

It's very important to be able to quickly and securely send messages. One early method used by the Greeks was to shave a slave's hair, write the message down, wait for the hair to grow back, and then send the slave on his way. This method is impractical for many reasons. One reason is the efficiency of the method. It takes far too long to send a message. Caesar instead used the following encryption scheme: shift each letter forward in the alphabet by 3. This can be easily represented in a table:

<Production note: following lines in a monospaced font>

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
D 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

</mono font>

The early part of the table is clear: "A" is written as "D", "B" is written as "E", "C" is written as "F" and so on. The difficulty comes toward the end of the alphabet. "W" is shifted forward by 3 to become "Z", but shifting "X" by 2 hits "Z", but at that point, there are no letters after "Z." The solution is to wrap around to the beginning, so after "Z" comes "A", resulting in "W" being written as "A", "Y" as "B" and "Z" as "C."

Of course, there's nothing special about the number 3. Shifting by 4 gives the following table for coding.

<Production note: following lines in a monospaced font>

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
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

Shift of 4 Coding Table

</mono font>

To decode a message written using this system, reverse the order of the lines. For example the previous encoding lines become:

<Production note: following lines in a monospaced font>

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
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

Shift of 4 Decoding Table

</mono font>

Example 1: Coding and decoding a simple message.

Using the "Shift of 4 Coding Table"
<Production note: following lines in a monospaced font>
"A simple message to be sent." becomes
"E wmqtpi qiwweki xs fi wirx."
</mono font>

Then by using the "Shift of 4 Decoding Table"
<Production note: following lines in a monospaced font>
"E wmqtpi qiwweki xs fi wirx." once again becomes
"A simple message to be sent."

</mono font>

There are 26 possible shifts: 0, 1, 2, and so on up to a shift of 25. Although shifts greater than 25 are legitimate, it should be clear that a shift of 26 is the same as a shift of 0. Similarly, a shift of 27 is the same as a shift of 1, and so on. Shifting letters to the left is also possible. However, any shift to the left can be replaced by a shift to the right. What is the right shift equivalent of a shift of 3 to the left? A left shift of three is equivalent to a right shift of 23.

Even though a coded message appears as nonsense, it's not too hard to decipher once the cipher method is known. The reason is that there are only 25 possibilities, and they're easy to check. Start decoding a message using one of the keys. If the result makes no sense, stop and use the next key. If the result makes sense, continue decoding using the same key until the message is completely decoded.
To set up the decoding table takes some time, but it is a one-time cost. That is, once the table is set up, it does not need to be recalculated in any way. A spy can carry around the table (or with very little effort recreate it on the spot) and easily decode any message sent using the Caesar Cipher with any shift.
<Production note: following lines in a monospaced font>

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

General Decoding Table for Caesar cipher showing all possible shifts

</mono font>

Any message encrypted by a Caesar cipher can be decrypted in at most 26 tries (realistically, at most 25, as it's unlikely a shift of 0 would be used).

While the Caesar cipher satisfies one of the primary goals of a cryptosystem, namely that it's easy to encrypt and decrypt, it suffers a serious flaw: it's far too simple to break it. There were many improvements and modifications to the Caesar cipher over the years. For centuries, these mostly involved alphabet codes, where one letter is replaced by another and there are rules of varying difficulty explaining which letters replace which.

Starting in the 19th century, however, some better schemes were developed. Our goal in this unit is to describe one of the most important encryption schemes to date, the **RSA algorithm,** and the mathematics behind it. Although it isn't used as much as it was, for many years it was *the* gold standard for encryption. It is relatively easy to implement, but believed to be very hard to crack.

Before we can describe RSA encryption, we need to discuss some of the mathematics involved. Fortunately we've already seen it, both in the Caesar cipher and in our daily routines. If someone tells you 10 plus 5 is 3, you would be right to question their mathematical abilities. However, if they say "it's 10 o'clock now, and in 5 hours it'll be 3 o'clock," you should not have a problem with that statement. Therefore you should have no problem with their first statement *if* the proper context is known. On a 12 hour clock, 10 + 5 **is** 3. When counting on a clock what happens after 12 is reached? At that point, we wrap around and start again at 1 o'clock. It takes 2 hours to get from 10 to 12, and then another hour to 1, another to 2, and finally one more makes it 3 o'clock. It may help to unravel the clock, repeating the values:

Computational Thinking Alert: Congruences

For many problems, it's important to realize what properties affect the answer, and what do not. For example, when Galileo dropped different balls to see which would land first, what mattered to him was their shape and not their color. We say two objects are *congruent* or *equivalent* if they agree on the properties that matter for the problem; thus they may disagree elsewhere. This is a very important perspective, as it allows us to focus on what truly matters and not be distracted by irrelevancies for the given problem. Note, however, that it is not always easy to figure out which are the properties that matter.

1 2 3 4 5 6 7 8 9 10 11 12 1 2 3 4 5 6 7 8 9 10 11 12 1 2 3 4 5 6 7 …

Notice this is exactly what happened with the alphabet in the Caesar cipher, where three letters after X is A. In homage to its origins on a clock, what follows is called **clock arithmetic.** Its fancier, more mathematical name is **modular arithmetic**. A clock has twelve hours, but modular arithmetic allows the "clock" to have as many hours as desired. We denote the number of hours on a general clock by *n*. This terminology is useful later as it allows us to discuss different clocks with different numbers of hours all at once. Let's look at some examples of properties on a 12 hour clock and then extend that to an *n* hour clock.

<<Insert a 24 hour clock diagram here. Label it figure 1.>>

Congruence or equivalence: On a 12 hour clock, two numbers are **congruent** if they differ by a multiple of 12. For example, 15 is congruent to 3 on the clock because there is an integer *k* (in this case, 1) such that 15 - k ∗12 = 3. Similarly 3 is congruent to 5 as 3 – (-1) ∗ 12 = 15. This example illustrates two important facts: if *x* is congruent to *y* then *y* is congruent to *x*, and *k* may be positive or negative.

We can see in Figure 1 that 3 and 15 are in the same position on the clock. In a similar way, 37 is congruent to 1 on the 12 hour clock, as their difference 37 – 1 = 36 = 3 ∗ 12 is a multiple of 12. In order to be able to describe these relationships concisely and quickly, we refer to the number of hours on the clock as the **modulus**. Thus our example is 37 equals 1 *modulo* 12, and we write it as 37 = 1 *mod* 12. In general, *x = y mod n* means *n* divides *x-y*.

Modular arithmetic has some expected properties, and some strange ones too. If *x = y mod n* and *a = b mod n*, then
(1) *x + a = y + b mod n,*
(2) *x – a = y – b mod n*, and
(3) *x a = y b mod n*.

The first claim is proven in the side bar. The others are left to the reader.

Proof that x + a = y + b mod n:
As x = y mod n and a = b mod n, there must be integers k and j such that
1) x = y + kn
2) a = b + jn.
Adding equations 1) and 2) gives
3) x + a = y + kn + b+ jn.
Rearranging terms and factoring n gives
4) x + a = y + b + (k+j)n,
Therefore x+a and y+b differ by a factor of n, and are thus congruent mod n.

What strange properties does modular arithmetic have? Just because *a ∗ x = a ∗ y mod n* it isn't necessarily true that *x = y*. For example, if *a = 4*, *x = 3*, *y = 6* and *n = 12*, then *4 ∗ 3 = 4 ∗ 6 modulo 12*, since *12 = 24 – 12*, but clearly *3* is not the same as *6*. Notice 3 and 6 are neither equal nor congruent modulo 12. Keep this in mind as a warning: while many properties are as you'd expect, there are some differences! It's very important to build up the theory carefully and rigorously to prevent making mistakes.

Another property important to cryptography is that any integer *x* is congruent to one and only one number in the set {0, 1, 2, …, n-1}. This unique number is written as *x mod n*. The algorithm for finding this number is in the sidebar.

Computational Thinking Alert
Algorithm for finding the unique value *x mod n* for x positive.
1. If x < n, then x is the number. Stop.
2. Subtract n from x.
3. If x-n < n, then x-n is the number. Stop.
4. Else let x take the value x-n and return to step 3.

Another convenient property is the modular multiplication property: *The product of two numbers mod* n *is the product of each of the mod* n *numbers mod* n. This sounds confusing as a statement, but is easier to understand through an example:

$$(17 * 33) \bmod 12 \equiv \big((17 \bmod 12) * (33 \bmod 12)\big) \bmod 12.$$

The left hand side is 561 modulo 12. As 561 = 46 ∗ 12 + 9, the left hand side is 9. Similarly the right hand side is 5 ∗ 9 modulo 12; as 45 is 3 ∗ 12 + 9, we see the right hand side is also 9.

While we only proved the claim for one example, a similar argument handles the general case. This result has important consequences: a relatively small number can always be used for multiplications in modular arithmetic by reducing by the modulus before multiplying.

It turns out that in RSA enormous powers of integers must be calculated, and thus this property is needed for exponentiation. Imagine *x mod n* is *y*, so there is some *k* with $x = y + kn$. What's x^2? We have

$$x^2 = (y + kn)^2 = y^2 + 2kn + k^2n^2 = y^2 + (2k+k^2n)n.$$

This makes it clear that $x^2 = y^2 \bmod n$, as they differ by a multiple of *n*.

What about x^3? Well, $x^3 = x\,x^2$ and since x^2 equals y^2 plus a multiple of n, we find

$$x^3 = x\,x^2 = (y + kn)(y^2 + ln) = y^3 + y^2kn+ y^2ln+kln^2 = y^3 + (y^2k+y^2l+kln)n.$$

This shows that $x^3 = y^3$ + a multiple of *n*. It's distracting to include the true value of the multiple, since multiples of *n* vanish when looking at what the expression is congruent to modulo *n*.

The pattern continues, and $x^m = y^m$ modulo *n*, or equivalently x^m is congruent to $(x \bmod n)^m \bmod n$.
One last modular arithmetic property is necessary; modular inverses. You have seen multiplicative inverses in earlier math classes. If *x* is a number, its multiplicative inverse is the number *y* such that $xy = 1$. The multiplicative inverse of *x* is written *1/x* or x^{-1}. In standard arithmetic, every number has a multiplicative inverse except for 0, and the inverse is always unique.

The situation is drastically different in modular arithmetic. We say *y* is the multiplicative inverse mod *n* to *x* if $x * y = 1 \bmod n$. Consider a clock with 12 hours. For some numbers we can easily find the multiplicative modular inverse: 1 x 1 = 1 mod *n*, so 1 is the multiplicative inverse of 1. This is the same for real numbers, as well. However, in modular arithmetic 5 ∗ 5 equals 1 mod 12. Further 7 ∗ 7 and 11 ∗ 11 are also equal to 1 mod 12. This is very different than real numbers, as here many different numbers are their own inverse! Other numbers don't have a multiplicative inverse mod 12. For example, there is no inverse to 2 mod 12. There are a few ways to show this. One way is to check all the possible numbers.

2 ∗ 1 = 2 mod 12,	2 ∗ 2 = 4 mod 12,	2 ∗ 3 = 6 mod 12,	2 ∗ 4 = 8 mod 12,
2 ∗ 5 = 10 mod 12,	2 ∗ 6 = 0 mod 12,	2 ∗ 7 = 2 mod 12,	2 ∗ 8 = 4 mod 12,
2 ∗ 9 = 6 mod 12,	2 ∗ 10 = 8 mod 12,	2 ∗ 11 = 10 mod 12,	2 ∗ 12 = 0 mod 12.

Even though there were four numbers on the clock that are their own multiplicative inverse, there is no number on the clock that is an inverse of 2. We can see this by noting that any number times 2 is even, and the reduction of any even number modulo 12 is never be odd. Thus we can't find a number *x* such that $x * 2$ is congruent to 1 modulo 12. Therefore 2 does not have an inverse modulo 12.

Homework problems:

#1. Assume $x = y \bmod n$ and $a = b \bmod n$. Show that
(1) $x - a = y - b \bmod n$, and
(2) $x\,a = y\,b \bmod n$.

Solution:

(1) If $x = y \bmod n$ then there is an integer k such that $x = y + k * n$, while if $a = b \bmod n$ there must be an integer m with $a = b + m * n$. If we subtract the second equality from the first, we get $x - a = (y + k * n) - (b + m * n)$. Regrouping, we find $x - a = y - b + (k * n + m * n)$. Notice, however, that $k * n + m * n$ is just $(k+m)n$, which is a multiple of n. We have thus shown that $x - a$ differs from $y - b$ by a multiple of n, and this is the definition of what it means for $x - a$ to equal $y - b$ modulo n.

(2) As in part (1), we have integers k and m such that $x = y + k * n$ and $a = b + m * n$. If we multiply these two equations we find $x\,a = (y + k * n)(b + m * n)$. Expanding the right hand side by using FOIL, we see $x * a = y * b + y * m * n + k * n * b + k * n * m * n$. Notice we may pull an n out of the last three terms on the right, getting $x * a = y * b + (y * m + k * b + k * n * m)n$. Thus $x * a$ and $y * b$ differ from each other by a multiple of n, proving the claim.

#2. Show by brute force calculation that 15^3 is congruent to 2^3 modulo 12. The point of this calculation is to check that x^3 mod n equals $(x \bmod n)^3$ mod n.

Solution: $15^3 = 15 * 15 * 15$, which is 3375, while $2^3 = 8$. Reducing 3375 modulo 12 gives 8, as 3375 is $281*12 + 8$.

#3. Calculate 15^{10} mod 12 by continually reducing. Note 15^{10} is 576650390625, but by doing mod arithmetic, none of the intermediate numbers is ever greater than 144!

#4. Consider a clock with 10 hours. Which numbers have multiplicative inverses modulo 10?

Solution: The invertible numbers are 1, 3, 7 and 9 as $1 * 1 = 1$, $3 * 7 = 21$ (which is 1 modulo 10), $7 * 3 = 21$ (which again is 1 modulo 10), and $9 * 9 = 81$ (which is 1 modulo 10). The numbers 0, 2, 4, 5, 6 and 8 are not invertible modulo 10, as each share a factor with 10.

#5. Consider a clock with 3 hours. Which numbers have a multiplicative inverse modulo 3? Repeat for 5, 7, and 11 hours. Notice these are all primes. Make a guess as to what happens if you have a clock with a prime number of hours. (Remember a positive integer m > 1 is prime if the only integers dividing m are itself and 1.)

Solution: The invertible numbers modulo 3 are 1 and 2, as $1 * 1$ and $2 * 2$ are both 1 modulo 3; 0 is clearly not invertible. For a clock with 5 hours, we again find all the non-zero numbers are invertible, as $1 * 1$, $2 * 3$, $3 * 2$ and $4 * 4$ are all 1 modulo 5. If there are seven hours, again all the non-zero numbers are invertible, since $1 * 1$, $2 * 4$, $3 * 5$, $4 * 2$, $5 * 3$ and $6 * 6$ are all 1 modulo 7. Finally, when there are 11 hours we again have all the non-zero numbers are invertible, as $1 * 1$, $2 * 6$, $3 * 4$, $4 * 3$, $5 * 9$, $6 * 2$, $7 * 8$, $8 * 7$, $9 * 5$ and $10 * 10$ are all 1 modulo 11. It is natural to guess that if we have a clock with a prime number of hours, then all the non-zero numbers are invertible modulo that prime; it can be shown that this guess is correct.

#6. Look up other early methods of cryptography, such as the Greek scytale and Egyptian hieroglyphics and write a short paragraph on each.

Solution: There are lots of good sources for early methods of cryptography. See for example

- http://en.wikipedia.org/wiki/Caesar_cipher
- http://en.wikipedia.org/wiki/Atbash
- http://en.wikipedia.org/wiki/Scytale
- http://en.wikipedia.org/wiki/Polybius#Cryptography
- http://en.wikipedia.org/wiki/Steganography
- http://en.wikipedia.org/wiki/Egyptian_hieroglyphs

#7. After reading this chapter, a friend decides to send you a message using the Caesar cipher, but he's worried about how easy it is for a spy to break the code. Therefore, he decides to use two shifts. He first shifts the alphabet by 3 and encrypts, and then shifts the alphabet by 7 and encrypts the encrypted message. He then sends you the doubly encrypted message. Did your friend increase the security of the message? Why or why not?

Solution: Our friend did not increase the security. The effect of first shifting by 3 and then shifting by 7 is equivalent to just immediately shifting by 10. Thus, the effect of two Caesar ciphers is just another Caesar cipher.

#8. The message KVUVAAYBZAIYBABZ was encrypted with a Caesar cipher. What was the shift, and what does the message say?

Solution: It says DONOTTRUSTBRUTUS, or after spacing it out, DO NOT TRUST BRUTUS. The shift is 19.

Day 2: Substitution Ciphers

The Caesar cipher only fills half of the wish list for a good method of encryption. It's very fast and easy to use, but it's very insecure. Fortunately, the 25 (or 26) Caesar ciphers fit into a larger, more general family called "substitution ciphers." This is a common theme in mathematics – a specific example is part of a bigger picture, and if the first instance that you work with does not provide you with what you need, perhaps a related element has the desired properties.

Computational Thinking Alert: The Pigeon-Hole Principle

Many people find the statement of the pigeon-hole principle obvious, and wonder about the value of isolating it out and giving it a fancy name. The reason is that sometimes the most powerful and useful ideas are the simplest, and it's worth calling attention to them. There are many problems which seem hard but, if you look at them the right way, have simple solutions.

The Caesar cipher works by swapping each letter of the alphabet with another. No two letters go to the same letter, nor is any letter missed. Both of these properties are essential. If two letters were encrypted to the same letter, there would be no way to determine which was meant and decryption is impossible. Thus each letter is the encryption of exactly one letter. Why? Imagine no letter encrypts to Z. Then the 26 letters of the alphabet encrypt to at most 25 letters. It's impossible for the 26 letters to encrypt to distinct letters (as we must have for decryption to be possible), as these 26 are going into 25 possibilities and thus at least one possibility must be hit twice (if each possibility is hit at most once, that accounts for at most 25 letters, and we need to assign values to all 26 letters of the alphabet). As a good exercise, draw a picture illustrating what we've just seen. Try to assign the 26 letters of the alphabet to the first 25, and you'll see one letter must be hit twice.

This concept is called the **Pigeon-hole Principle** (or the Box Principle, or Dirichlet's Box Principle). In its most entertaining form, it says the following: if there are have *n+1* pigeons and each pigeon is placed in one of *n* boxes, then at least one box must have at least 2 pigeons. Why? Imagine each box has at most 1 pigeon. As there are only *n* boxes, this can account for at most *n* pigeons, but there are *n+1* pigeons.

This simple principle turns out to be an indispensible tool in higher mathematics, unfortunately, it is not constructive. This means that while it is known at least one box has at least 2 pigeons, it does not say *which* box, only that there is at least one. The alphabet encryption can be recast along these lines. Imagine no letter encrypts to *Z*. This would be a problem in an encryption, since no message including the letter *Z* could be sent. There are 26 letters (these are the pigeons) that must be sent to the 25 remaining letters (these are the boxes). Thus, no matter what choice is made, at least two letters are encrypted to the same letter. Again, it is not known *which* letters are sent to the same letter, but it *must* happen. This demonstrates the power of this perspective. There are an incredibly large number of ways to assign the letters, but there is no need to investigate further any assignment where *Z* is missed. Of course, similar reasoning applies to the other letters, so all letters will be used in all of the encryptions.

Armed with these observations, is it possible to come up with a new encryption method? The problem with the Caeser cipher is once it is known how *one* letter encrypts, the others are known as well. For instance, if *A* goes to *D*, then the shift is 3, so *B* must go to E. As an alternative, take *A* and choose one of the 26 letters of the alphabet to send it to. There are 26 possible choices for *A*. For example, let *A* go to *Q*. There are now 25 choices remaining for *B*, since it can't be sent to the same letter we sent *A* to, but everything else is a valid choice. To continue the example, let *B* go to *H*. So far, there are $26 * 25 = 650$ possible pairs of distinct letters to send (*A*, *B*) to. Continuing, there are 24 remaining choices for *C*. Let *C* go to *P*. There are $26 * 25 * 24 = 15{,}600$ possible choices so far and only one of these choices is what we have; namely (*A*, *B*, *C*) goes to (*Q*, *H*, *P*). This method of choosing a key creates a ***permutation cipher***, as in the end all that has been done is permuting the alphabet. That is, it is just being written it in a different order. To finish the example:
<Production note: following lines in a monospaced font>

```
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
Q H P A S T G C Z R F U I B J E V O K D Y N M L X W
```
<End monospaced font>

Notice how rapidly the possibilities grow. There were only 26 possible Caeser ciphers (and one of those would not ever be used!). The number of possible permutation ciphers is $26 * 25 * 24 * 23 * 22 * 21 * 20 * 19 * 18 * 17 * 16 * 15 * 14 * 13 * 12 * 11 * 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1$, or 403,291,461,126,605,635,584,000,000. That's

Computational Thinking Alert
Computational Complexity – It's not helpful to have an algorithm which solves a problem, if the method takes so long to solve that the result that the irrelevant by the time the algorithm finishes. Although there are many "brute force" algorithms that are fine for small calculations, more finesse is needed for calculations involving large data sets.

approximately 4.03 x 10^{26} in scientific notation. This is a lot more options than the standard Caesar cipher! All the possible Caeser cipher codes can be written on half a page, but it would not be possible to write all of the possible permutation ciphers using all of the printed material in the world!

Note to the teacher: If your students are familiar with factorial notation, you can consider giving them related questions later. For example, how many ciphers have at least one letter unchanged? Or, how many have exactly one unchanged (find the number that leave two fixed…).

To get a sense of how truly large this number is, imagine the following example. Everyone in the world (we'll round up and say there are 10 billion, or 10^{10} people) decide to write down all the possible permutation ciphers. Assume each person can write one cipher per second, which is extremely fast writing! How long would it take them to write all the possibilities? The answer is approximately

$$4.03 * \ 10^{10}\text{ciphers} \ \times \ \frac{1 \text{ second}}{10^{10}\text{ciphers}} = 4.03 \times 10^{16} \text{ seconds.}$$

There are 31,557,600 seconds in a year (60 seconds per minute times 60 minutes per hour times 24 hours per day times 365.25 days per year). To make the arithmetic easy, round up and say there are $4 * 10^7$ seconds in a year. So, it would take about

$$4.03 \ \times 10^{16} \text{ seconds } \times \frac{1 \text{ year}}{4 \ \times 10^7 \text{ seconds}} \approx 1 \ \times 10^9 \text{ years}$$

to write down all the ciphers. That's a billion years! There are thus so many possibilities here that surely this is a secure cryptography method. The Caesar cipher only had 25 or 26 options, so someone intercepting a message could try them all in a short time. In the case of the substitution cipher, it would take billions of years, with all of the people in the world trying one possibility per second.

WolframAlpha bills itself as a "computational knowledge engine." The site "http://www.wolframalpha.com/" is free to use, after registration, but will send advertisements to users. If it violates your school's policies to have students register, or to accept random advertisements, it is possible to use a graphing calculator, Wolfram Mathematica or a spreadsheet program to do any of the actions described in this module. Each of the problems which use WolframAlpha will be described in such a way that translating to another system will be well within the means of a typical high school classroom. In the following lessons, the term "random" will be used, even though we mean "pseudorandom." Although numbers generated by computational methods are not truly random, they are close enough for our needs to mean the same thing. You may want to point this out to your students depending on their level of sophistication with mathematics and computers. There are now Wolfram Alpha apps for iOS and Android operating systems, so these examples can be worked out on a smartphone or tablet PC. Be aware that capitalization matters, and the commands need to be typed as stated.

To get a random permutation of the letters of the alphabet for a permutation cipher, type RandomSample[CharacterRange["A", "Z"]] into WolframAlpha. (CharacterRange["A", "Z"] tells the computer to generate a list of characters from "A" to "Z" and RandomSample tells the

computer to create a random ordering (or permutation) of that list. Type the random list from Wolfram under the letters of the alphabet in order, and you will have a coding key similar to the ones used for the Caesar cipher, though with far greater complexity.

Activity 2
Have each student generate a coding/decoding key using WolframAlpha. They should then code a message and trade coded messages, but not keys, to see if someone else in the class can break the code. After giving the students five minutes, see if anyone has broken someone else's coded message. If the messages are short, it should be difficult to do so you might tell them to have at least 30 words. As the length of the messages increases, the likelihood of breaking the code increases. After they have, most likely unsuccessfully, tried to decode the messages, the coders can give the key to their partners to decode and see that, with the key, it is just as easy to decode as the Caesar cipher.

It was previously stated that it would take a billion years to try all of the possibilities of the permutation cipher if all of the people in the world were trying one possible solution per second. This would lead you to believe that it is very hard to crack the cipher. If the only way you knew to solve quadratic equations was by factoring, and you were required to find the roots of $x^2 – 1701\, x + 717200$, it will be difficult to find them. Almost surely, anything you check will fail, but you know there are two roots.

It is easy to create a complex looking polynomial if you know what you want the roots to be. For example let's take $\frac{1701+\sqrt{24601}}{2}$ and $\frac{1701-\sqrt{24601}}{2}$ as the roots. The polynomial with these as the two roots is $\left(x-\frac{1701+\sqrt{24601}}{2}\right)\left(x-\frac{1701-\sqrt{24601}}{2}\right)$, which equals $x^2 – 1701\, x + 717200$, the equation in the previous paragraph. What this shows is that even if you don't know the quadratic formula, you can create a complicated looking polynomial where *you* know the roots. Thinking that these roots could be used as passwords to an encryption scheme and giving the roots to your friends, you might think that no one else could figure them out. However, you'd be wrong. The quadratic formula *does* exist and gives a very easy way to find the roots. What you thought was secure turns out to be trivial to crack. The roots of $ax^2 + bx + c = 0$ are $\frac{-b\pm\sqrt{b^2-4ac}}{2a}$, and all someone needs to do is substitute for a, b and c.

Computational Thinking Alert:

We have taken a problem that seemed difficult to solve and, by using a formula, made it easy. The danger in creating encryption algorithms is that when creating a scheme that seems complex to solve by the methods we are used to, there may be someone who sees the solution as trivial because of additional facts known only to them.

What does finding roots of a polynomial have to do with the permutation cipher? Just because something looks hard to crack doesn't mean it is. It looks like a permutation cipher should be hard to crack because there are so many possible keys, but there are some very simple observations that help break the message. The problem is the message is a direct swap of letters from English sentences. In the example above, every time there is an *A* in the message it is replaced with a *Q*. More importantly, each *E* becomes an *S* and each *T* becomes a *D*. Certain letters are more common than others in English text. In the table below (taken from Henry Beker and Fred Piper, Cipher Systems: The Protection of Communications, Wiley-Interscience, 1982, page 397) are the frequencies. It's important to note that other sources will give slightly different frequencies (the same letter frequencies in Shakespeare's writings should not be expected in Hollywood scripts).

Letter	Frequency
a	8.17%
b	1.49%
c	2.78%
d	4.25%
e	12.70%
f	2.23%
g	2.02%
h	6.09%
i	6.97%
j	0.15%
k	0.77%
l	4.03%
m	2.41%
n	6.75%
o	7.51%
p	1.93%
q	0.10%
r	5.99%
s	6.33%
t	9.06%
u	2.76%
v	0.98%
w	2.36%
x	0.15%
y	1.97%
z	0.07%

Letter frequencies in the English language.

It turns out this simple frequency analysis is powerful enough to break long messages easily. How easily? You can find scrambled messages like these in many daily newspapers. These aren't aimed at mathematical superstars, but are for consumption by the general populace. It takes some people very little time to crack these codes using their knowledge of the frequency of letters in English, the placement of punctuation and length of common words.

For example, imagine intercepting the message:

IH JXH PHTPSH TB JXH MFVJHR QJDJHQ, VF TYRHY JT BTYU D UTYH PHYBHNJ MFVTF, HQJDWSVQX KMQJVNH, VFQMYH RTUHQJVN JYDFGMVSVJL, PYTEVRH BTY JXH NTUUTF RHBHFNH, PYTUTJH JXH CHFHYDS IHSBDYH, DFR QHNMYH JXH WSHQQVFCQ TB SVWHYJL JT TMYQHSEHQ DFR TMY PTQJHYVJL, RT TYRDVF DFR HQJDWSVQX JXVQ NTFQJVJMJVTF BTY JXH MFVJHR QJDJHQ TB DUHYVND.

To demonstrate how to decrypt this type of code, as letters are determined, we will write the letters we know (decrypted) in lowercase, and leave the unknown (encrypted) letters in uppercase.

Let's assume the message was encrypted with a permutation cipher and try to look for common letters. A little bit of searching shows that *H* occurs 39 times in the 268 characters in the message, for about 14.5%; other popular letters are 29 *J*'s (about 10.8%) and 14 *D*'s (about 5.2%). If you have programming expertise, you should write a simple program to find the frequencies of the different letters; a bit of web searching will reveal many sites that have applets for this (for example, http://www.wiley.com/college/mat/gilbert139343/java/java11_s.html).

Looking at the letter frequency table, we should be fairly confident about guessing that *H* is really *E*, and somewhat confident that *J* is really *T*. After that, it gets a bit harder.

Of course, the frequency table is not the only available tool. There are some common sense tricks to prune the analysis. The spacing is a great aid. Notice that in the first line there is a one letter word, *D*. There are only two one letter words in English: *a* and *I*. Thus, *D* is either *a* or *i*. We also notice that *JXH* occurs six times in the text. It's quite likely that *JXH* is *the*. If that's correct, then *JT* has to be *to*. Trying *D* as *a*, *J* as *t*, *X* as *h*, *H* as *e* and *T* as *o* gives:

Ie the PeoPSe oB the MFVteR QtateQ, VF oYReY to BoYU a UoYe PeYBeNt MFVoF, eQtaWSVQh KMQtVNe, VFQMYe RoUeQtVN tYaFGMVSVtL, PYoEVRe BoY the NoUUoF ReBeFNe, PYoUote the CeFeYaS IeSBaYe, aFR QeNMYe the WSeQQVFCQ oB SVWeYtL to oMYQeSEeQ aFR oMY PoQteYVtL, Ro oYRaVF aFR eQtaWSVQh thVQ NoFQtVtMtVoF BoY the MFVteR QtateQ oB aUeYVNa.

Focusing next on the word *oB*, the only possibilities for *B* are *f*, *n*, *r* or *y*. There are other words to focus on, as well. A great choice is *QtateQ*, which has to be *states*. Using that guess yields

Ie the PeoPSe oB the MFVteR states, VF oYReY to BoYU a UoYe PeYBeNt MFVoF, estaWSVsh KMstVNe, VFsMYe RoUestVN tYaFGMVSVtL, PYoEVRe BoY the NoUUoF ReBeFNe, PYoUote the CeFeYaS IeSBaYe, aFR seNMYe the WSessVFCs oB SVWeYtL to oMYseSEes aFR oMY PosteYVtL, Ro oYRaVF aFR estaWSVsh thVs NoFstVtMtVoF BoY the MFVteR states oB aUeYVNa.

The rest of the decryption is left to the reader (likely candidates to use for the next guess are *Ie* and *estaWSVsh*).

The point of this exercise is to highlight how careful one must be in cryptography. At first glance, it seemed like permutation ciphers would easily provide all the security we need. After all, there were over 4.03 x 10^{26} possible ciphers. It was already calculated that it would take over a billion years to try all the ciphers, even if 10 billion a second could be tested! However, in less than 5 minutes the code can be cracked. This is a valuable lesson. Just because something *seems* hard, it doesn't mean it *is* hard. A much better encryption scheme is needed if information is to be secure.

Homework Problems:
#1. It's surprisingly involved to count the number of permutations that leave at least one letter fixed. It's not as easy as finding out how many leave the letter *A* fixed and then multiplying by 26, due to double counting. That said, find how many permutations leave the letter *A* fixed (so *A* is sent to *A*), and how many leave *A* and *B* fixed.

Solution: There are 25! Or 15,511,210,043,330,985,984,000,000 permutations that leave the letter A fixed. The reason is that we have no choice in what A is sent to, and then can permute the remaining 25 letters any way. If both A and B are fixed the answer is 24!, or 620,448,401,733,239,439,360,000.

#2. How many permutations of the alphabet send *A* to *B*, *B* to *D*, *C* to *Z*, *D* to *C* and *Z* to *A*? How many are there that send *A* to *B*, *B* to *C*, *C* to *Z*, *D* to *G*, *G* to *Z* and *Z* to *A*?

Solution: We have fixed what A, B, C, D and Z go to, and have complete freedom for the remaining 21 letters. Thus the answer is 21!, or 51,090,942,171,709,440,000. In the second part, we fix what A, B, C, D, G and Z are sent to; there are thus 20! or 2,432,902,008,176,640,000 options.

#3. Break the alphabet into two halves: {*A, B, …, M*} and {*N, O, …, Z*}. How many permutations are there that shuffle the first half to the first half, and the second half to the second half? This means that *A* can be sent to *A*, *B*, …, or *M*, but not to *N*, *O*, …, or *Z*.

Solution: There are 13! ways to permute the first 13 letters amongst themselves, and similarly 13! ways to permute the second half of the alphabet. The answer is thus $13! * 13! = 38{,}775{,}788{,}043{,}632{,}640{,}000$.

#4. Consider a four letter alphabet, {*A, B, C, D*}. How many permutations are there? Write them all down. How many have at least one letter fixed? For the brave: redo this problem for 5 and for 6 letter alphabets. Make a guess as to the percentage that have at least one letter fixed (it involves 1/e, where e is the base of the natural logarithm).

Solution: There are 4! or 24 permutations. They are
{A, B, C, D}, {A, B, D, C}, {A, C, B, D}, {A, C, D, B}, {A, D, B, C},
{A, D, C, B}, {B, A, C, D}, {B, A, D, C}, {B, C, A, D}, {B, C, D, A},
{B, D, A, C}, {B, D, C, A}, {C, A, B, D}, {C, A, D, B}, {C, B, A, D},
{C, B, D, A}, {C, D, A, B}, {C, D, B, A}, {D, A, B, C}, {D, A, C, B},
{D, B, A, C}, {D, B, C, A}, {D, C, A, B}, {D, C, B, A}.

We highlight the ones with at least one letter fixed. If you go to http://www.wolframalpha.com/ and type Permutations[{A, B, C, D, E}], WolframAlpha will output all the permutations of 5 letters (to get six letters, just add an F after the E).

{A, B, C, D}, {A, B, D, C}, {A, C, B, D}, {A, C, D, B}, {A, D, B, C}, {A, D, C, B}, {B, A, C, D}, {B, A, D, C}, **{B, C, A, D},** {B, C, D, A}, {B, D, A, C}, {B, D, C, A}, **{C, A, B, D},** {C, A, D, B}, **{C, B, A, D}, {C, B, D, A},** {C, D, A, B}, {C, D, B, A}, {D, A, B, C}, **{D, A, C, B}, {D, B, A, C}, {D, B, C, A},** {D, C, A, B}, {D, C, B, A}.

Thus 14 out of the 24, or about 58.3%, have at least one letter fixed.

#5. Use WolframAlpha to generate a random permutation of the alphabet. Do you think the following was randomly chosen: P O I U Y T R E W Q A S D F G H J K L M N B V C X Z? What about G L C E H M R K Y P X O S W I N J T U B V F Z D A Q? Why or why not?

Solution: Type RandomSample[CharacterRange["A","Z"]] into WolframAlpha to get a random permutation. It is unlikely the first string above is a truly randomly chosen string (though anything is possible), as it is reading the letters on the standard keyboard in a snaking pattern, starting at the P, moving to the Q, then down to A, then over to L, down to M and finally ending at Z. The second string looks far more random.

#6. The following is a nice application of both the Pigeon-hole Principle *and* quadratic polynomials. Consider an army with 10 generals. One wants a security system such that any three of them can determine the code to launch nuclear missiles, but no two of them can. It is possible to devise such a system by using a quadratic polynomial, such as $ax^2 + bx + c$. To launch the missiles, one must input (a,b,c). One cannot just tell each general one of a, b, or c (as then it is possible that some subset of three generals won't know a, b and c); however, if a general knows two of (a,b,c), then some sets of two generals can launch the missiles! What information should be given to the generals so that any three can find a,b and c but no two can? What about the general situation with n generals and any m generals can launch (but no set of m-1) can?

Solution: Give each general a point on the parabola. That way any three can get together and recover the curve, as three points uniquely determine a parabola, but no set of two generals can. If we want any set of M generals to be able to launch, we need to give each person a point on a curve y = f(x), with f(x) a polynomial of degree M-1.

#7. Find the letter frequencies of some early English works, such as Shakespeare, Chaucer or Spencer, and compare to the current letter frequencies.

Solution: These are available with a little web searching. For example, http://math.ucsd.edu/~crypto/Projects/FeliksDushtsky/letter-freq-compute.pdf

#8. Look at any page of this module, find the letter frequencies, and compare them to the values quoted in the text.

Solution: There are lots of programs online that will count letter frequencies. See for example
http://www.csgnetwork.com/documentanalystcalc.html or
http://www.characterfrequencyanalyzer.com/english/index.php
http://www.wiley.com/college/mat/gilbert139343/java/java11_s.html

#9. So far we have ignored punctuation marks as well as spaces. Estimate what percentage of characters are letters and what percentage is a space by looking at any moderately long piece of writing.

Solution: Note some of the letter frequency counters will also count special symbols; see for example
http://www.characterfrequencyanalyzer.com/english/tfa_results.php

#10. When Morse invented the telegraph, he also had to invent a way to send messages. At that time, he could only use dots and dashes. He needed to assign symbols to letters and wanted to have the faster symbols to transmit go with the more frequently used letters. Look up Morse code, and see how well he did.

Solution: Morse code is available online at http://en.wikipedia.org/wiki/Morse_code

#11. Decipher the following passage, which was encrypted with a substitution cipher. The passage has no apostrophes and punctuation marks, but spacing remains.
ITPR UR YTP WQMJVP QK TMOFR PXPRYV UY SPWQOPV RPWPVVFJE KQJ QRP ZPQZGP YQ HUVVQGXP YTP ZQGUYUWFG SFRHV ITUWT TFXP WQRRPWYPH YTPO IUYT FRQYTPJ FRH YQ FVVMOP FOQRC YTP ZQIPJV QK YTP PFJYT YTP VPZFJFYP FRH PLMFG VYFYUQR YQ ITUWT YTP GFIV QK RFYMJP FRH QK RFYMJPV CQH PRYUYGP YTPO F HPWPRY JPVZPWY YQ YTP QZURUQRV QK OFRDURH JPLMUJPV YTFY YTPE VTQMGH HPWGFJP YTP WFMVPV ITUWT UOZPG YTPO YQ YTP VPZFJFYUQR.

Solution: It's the first sentence of the Declaration of Independence: When in the Course of human events it becomes necessary for one people to dissolve the political bands which have connected them with another and to assume among the powers of the earth the separate and equal station to which the Laws of Nature and of Natures God entitle them a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation.

#12. Encrypt a message using an alphabet cipher and exchange with a classmate. Try to decrypt their message. You should keep in the spacing and punctuation, and have at least 100 characters of text. A great website to use is http://25yearsofprogramming.com/fun/ciphers.htm. It will encrypt messages for you using a substitution cipher (though it won't tell you what cipher it's using!).

Solution: The above website is a great aid in problems like this.

#13. Substitution ciphers are not secure because of frequency analysis, however, there are so many possibilities that it seems a shame to completely discard the method. Before and during World War II, the Germans developed an encryption scheme based on substitution ciphers, but with several added layers of complications to increase security. The result was Enigma, which the Germans believed provided them with easy and secure communication. Fortunately for the Allies, there were flaws in how the Germans used Enigma, and their codes were compromised. Read about Enigma and Ultra (the Allied cracking of Enigma) and write a paragraph explaining them. The NSA has a lot of good pamphlets on their homepage:
http://www.nsa.gov/about/cryptologic_heritage/center_crypt_history/publications/wwii.shtml.

Solution: There are many additional items on Enigma and Ultra available from the NSA's homepage.

The final three problems are challenging applications of the Pigeon-hole Principle.

#14. Consider any set of 51 distinct numbers chosen from {1, 2, …, 99, 100}. Show that, no matter what 51 numbers are chosen, two of them sum to a multiple of 50. Is this still true if we only choose 50 distinct numbers? *Hint: use the Pigeon-hole Principle.*

Solution: We split the numbers into pairs of numbers that add to 50. We have (1, 99), (2, 98), (3, 97), …, (48, 52), (49, 51) and (50,100). Notice that there are exactly 50 such pairs. We `win' if we choose two numbers from the same pair, as then we get a sum of two numbers that's a multiple of 50; we `lose' if somehow we choose 51 numbers and never get two in the same pair. This latter possibility cannot happen by the Pidgeon-hole Principle. If we choose at most one element from each pair, that only accounts for 50 numbers. We must choose one additional, which gives us the second number in a pair.

It is *not* still true if we choose only 50 numbers (for example, if we take the integers 1, 2, …, 50, then there is no sum of two distinct elements from this list that is a multiple of 50).

#15. Consider any set of 51 distinct numbers chosen from {1, 2, …, 99, 100}. Show that, no matter what 51 numbers are chosen, at least one number in this list divides another number in the list. Is this still true if we only choose 50 numbers? *Hint: use the Pigeon-hole Principle.*

Solution: Any integer x can be written as an even number times an odd number; thus $x = 2^n (2m+1)$ for some integers n and m depending on x (if m=1 then our number is a power of 2, while if n=0 then our number is odd). If two numbers have the same odd part, then whichever one has the smaller power of 2 divides the other. There are only 50 possible odd parts for numbers from 1 to 100. Thus two numbers in our set of 51 must have the same odd part (by the Pidgeon-hole Principle). Let's say our numbers are $x = 2^n (2m+1)$ and $y = 2^k (2m+1)$. If $k < n$ then y divides x, while if $k > n$ then x divides y (k and n cannot be equal as otherwise x and y would be equal).

If we only choose 50 numbers from {1, 2, …, 100} it is not necessarily true. We can't just take the 50 odd or the 50 even numbers, though (for the 50 odd numbers note 1 divides everything, while for the 50 even numbers 2 divides everything). We can, however, take {2, 3, 5, 7, 9, 11, …, 99} (i.e., replace 1 with 2 in the list of the odd numbers).

#16. Consider any list of *n* integers. Prove there is a subset of these *n* numbers that sums to a multiple of *n*. Note the numbers need not be distinct. Must this still be true if our list only has *n-1* numbers? *Hint: use the Pigeon-hole Principle.*

Solution: Label our numbers x_1, x_2, …, x_n. Look at the increasing sequence of sums x_1, $x_1 + x_2$, $x_1 + x_2 + x_3$, $x_1 + x_2 + x_3 + x_4$, and so on through $x_1 + x_2 + \ldots + x_n$. If one of these sums is a multiple of n (in other words, congruent to 0 modulo n), then we are done. Assume that none of these sums are congruent to 0 modulo n. There are thus only n-1 possibilities for what these sums are congruent to. As we have n sums, by the Pidgeon-hole Principle two of the sums are the same modulo n. Thus, if we subtract the smaller from the larger, the result is congruent to 0 modulo n. Note that when we subtract we get a sum of elements from our list. For example, if $x_1 + x_2 + x_3 + x_4$ and $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7$ are congruent to the same number modulo n, then their difference $x_5 + x_6 + x_7$ is congruent to 0 modulo n, and we have found our sum which is a multiple of n.

It is false if we only have n-1 numbers. We can see this by taking 1 a total of n-1 times.

Day 3: Motivation / RSA Statement

Activity:
We start today with what looks like a silly riddle, but is not. Don't be fooled by appearances; we'll see the solution to this riddle is a billion (or a trillion) dollar industry!

Problem: Imagine Alice and Bob are stranded on two different desert islands. Neither can travel to the other's island, but a pirate ship constantly sails between the two. Her captain isn't a bad guy and he's happy to carry any cargo between Alice and Bob. Unfortunately his sailors are easily tempted, and will steal anything from an unlocked box. Bob wants to send a ring to Alice and ask her to marry him. Bob and Alice each have their own lock and a key that opens only their lock. There's a box on the pirate ship that can be locked by either (or both) of their locks. How can Bob get the ring to Alice without the pirates taking it? Remember, so long as either Alice or Bob have their lock on the box, the pirates cannot open it and they'll transfer the item. If you think about this, you should see the connection between this problem and the previous lectures; here the ring is the information we want to send, and the pirates are the insecure channel.

This should be a group activity – give the students a chance to break into small groups and brainstorm on how to proceed.

Solution: Bob places the ring in the box and puts on his lock. When Alice receives the locked box she can't open it, but she can add her lock. She does this, and sends it back to Bob. He can't open it as it has both his and Alice's locks, but he can and does remove his lock. The pirates now bring the locked box back to Alice. She has received a locked box before, but the last time it had Bob's lock on it. This time it's her lock and she can use her key to open the box, and retrieve the ring. Seeing how clever Bob is, how can she refuse his proposal?
It's often easier to follow a solution if we draw a picture. Let's keep track of what is where and when.
[[NOTE: DRAW THESE PICTURES]]
Step 1: Bob puts the ring in the box, and just Bob's lock on the box. The box is taken to Alice's island.
Step 2: Alice puts her lock on the box. Both Alice and Bob's locks are now on, and it goes to Bob's island.
Step 3: Bob removes his lock, and the box with just Alice's lock on it goes back to Alice's island.
Step 4: Alice removes her lock and gets the ring.

It’s very easy to miss the magnitude of what was just done. Alice and Bob were able to share a secret without meeting. Furthermore, everything they did was open to all to see, but no one was able to interfere with it.

Entire books have been written on how the internet has changed our lives. One very important example of these changes is e-commerce. Millions of consumers buy products or pay bills online on a daily basis. It’s essential that the two parties to these transactions are able to communicate securely. If a customer wants to buy something, the customer needs to authorize a credit card company or bank to transfer funds. Clearly this person doesn’t want someone else to be able to pretend to be him and access his accounts, so it is necessary to verify his identity with the bank. However, it’s not convenient to go in person to verify each transaction. In fact, one of the reasons e-commerce is so desirable is consumers want to avoid the hassle of going places! The danger that *must* be guarded against is that if information is sent to the bank to verify the transaction, a third party can intercept that information and pretend to be the actual customer in the future.

Teacher's note: We’ll first describe how we can verify an identity without allowing personal information to leak out in public, then describe how RSA works. The next lesson has the students use RSA to encrypt and decrypt. We won’t justify why it works as that is well beyond what can be done in a five or six day unit without the students knowing some Abstract Algebra, and beyond the scope of most high school math curricula. Instead, we’ll concentrate on what RSA can do, and how we implement it. In doing so we’ll see a lot of important mathematical concepts, as well as being introduced to some good ways to approach problem solving.

What is needed is a procedure, similar to the one Bob and Alice used, that allows information to be passed back and forth, in public without anyone along the route being able to read the information. This procedure is called an **"algorithm,"** and this particular problem is one of the most fundamental problems in cryptography. An algorithm for solving this problem was introduced to the public in 1978 by Ron Rivest, Adi Shamir and Leonard Adleman. The algorithm is called RSA, from their first letters of their last names.

Computational Thinking Alert:
An "algorithm" in a computational sense is a step by step procedure for solving a problem using a finite number of steps.

Keep in mind, it’s not enough to say this procedure can be done; we need to be able to do it in a reasonable amount of time. For example, earlier we saw it would take one billion years for everyone on the planet to enumerate all possible permutation ciphers if they worked at the unrealistically fast rate of one cipher per second. It’s *possible* to write the list in a finite amount of time, but it isn’t practical. For the bank transaction, the information must be encrypted and decrypted, by computers, in a matter of seconds (or less).

Before the algorithm can be described, some terminology is necessary.

1. **Prime**: A positive integer $n > 1$ is prime if the only positive integer divisors of n are 1 and n. Thus 7, 11 and 47 are prime while 12 (divisible by 2, 3, 4 and 6), 55 (divisible by 5 and 11) and 91 (divisible by 7 and 23) are not.
2. **Greatest Common Divisor (gcd)**: The greatest common divisor of two positive integers x and y is the largest integer dividing both. Using function notation this can be written as *gcd(x,y)*. Thus, *gcd(12, 46) = 2*, *gcd(16, 36) = 4* and *gcd(12,35) = 1*. If the gcd of two numbers is 1, we say the two numbers are relatively prime.
3. **Relatively prime:** Two positive integers x and y are relatively prime if the only integer dividing both is 1. Equivalently, x and y are relatively prime if their greatest common divisor is 1.

Before encrypting the message, encode the message in a numerical form. There are a number of advantages to this. The primary advantage is that computers can do mathematical operations on numbers quickly and accurately, and save us the effort of encrypting and decrypting.

The following table (or something similar, such as ASCII) can be used for encoding the alphabet, numbers and some basic punctuation.

A	b	C	d	e	f	G	h	i	j
01	02	03	04	05	06	07	08	09	10
K	l	m	n	o	p	Q	r	s	t
11	12	13	14	15	16	17	18	19	20
u	v	w	x	y	z	0	1	2	3
21	22	23	24	25	26	27	28	29	30
4	5	6	7	8	9	.	,	?	"
31	32	33	34	35	36	37	38	39	40

Then write the number pairs one after another for the message. Thus *fire* encodes as 06091805, which, as a number, is 6,091,805. Do not worry about the leading zero being dropped, it does not change the value of the number and causes no loss of information.

The method below allows Bob to send a message to Alice.

Step 1: Alice chooses two large primes *p* and *q*. She calculates their product *n = pq*. The number *n* is the modulus of both the public and private keys.

Step 2: Alice chooses a number *d* relatively prime to *(p-1)(q-1)*, and then finds the unique number *e* ($1 \le e \le (p-1)(q-1)$) such that *ed = 1 mod n*. She keeps *d* (which stands for decrypt) secret but publishes *e* (for encrypt).

Step 3: Alice makes the numbers *n* and *e* available for anyone who wants to send her a secure message. She makes sure to keep *p*, *q* and *d* secret.

Step 4: Bob takes his message *M*, which is a number between *0* and *n-1*, and computes $X = M^e \bmod n$. He sends *X* to Alice. In order to make it hard to break the code, *M* has to be relatively prime to *n*. If the original message is not relatively prime, then Bob has to tweak the message a bit and try again.
Step 5: Alice receives *X* (which is $M^e \bmod n$) and computes $X^d \bmod n$. Amazingly, this turns out to be *M*, the original message that Bob encrypted. She can undo the encoding as numbers, turning the numerical message into characters which can be read.

There is a lot going on here and it shouldn't be clear at all why it works. Let's isolate the key implementation issues.

1. How do we find the primes? The statement "choose two large prime numbers *p* and *q*" may be easier said than done. And just because it can be done, doesn't mean it can be finished in a reasonable amount of time. In practice, *p* and *q* should have at least 200 digits. This means these numbers exceed 10^{200}! **Aside to the teacher: the students need to know there are infinitely many primes, or at least many primes…. All of a sudden we're seeing that results from theoretical mathematics have applications; we need to make sure we don't run out of primes. One of the most common ways to prove there are infinitely many primes goes back to Euclid's works:** http://en.wikipedia.org/wiki/Euclid%27s_theorem
2. How do we find an integer *d* such that $ed = 1 \bmod (p-1)(q-1)$?
3. What if our message *M* isn't relatively prime to *n*? How hard is it to find a close message that is relatively prime to *n*?
4. How hard is it to compute $X = M^e \bmod N$ (as well as $X^d \bmod n$)? Typically *e* and *d* will be roughly the same size as *p* and *q*. If the primes have 200 digits, so do *e* and *d*. This means raising *M* to the $10^{200\text{th}}$ power! That's an incredibly large number of operations to do. Is it really possible to do this many operations in a reasonable amount of time? Remember one of the selling points of the Caesar and Permutation ciphers is their ease to use. If we can do a googol (that's 10^{100}) operations a year (which is well beyond current capabilities), it would take a googol years to reach 10^{200} operations.

In order to implement the algorithm, $M^e \bmod n$ must be calculated in significantly less than 10^{200} operations, as this many multiplications is clearly impractical. A method called **fast exponentiation** allows these calculations to be done with incredible speed.

Teacher's note: The purpose of this activity is to show that there are methods to solve problems that can considerably shorten the amount of time taken to solve the problem. You may choose not to do the calculations in class, but should emphasize that it is not enough to have a solution to a problem, if the solution is so time consuming as to be impractical.

Activity: Break the class into small groups. Each student should randomly choose some day in the year (or birthday) and write it down. Each student is then paired with another, and is tasked with finding the *secret day of the other person in the fewest yes-earlier-later questions. This means a student can only ask questions of the form: Is your date* X*? (Here* X *is a specific date, such as June 6 or December 7.) If a student guesses the correct date, the other student must say yes. If the student guesses wrong, the other student must say if the date is earlier or later in the year.*

Solution: The simplest strategy is to go through the year. Is it January 1st? Is it January 2nd? This will work, but it will take a long time. We can improve it and figure out the month. Is it February 1st? If the answer is earlier, we know the date was in January. If it's later, the second question should ask if it is March 1st. If it is earlier, then the date was in February. In this manner we can get the month in at most 11 questions. Once we have the month, we can go day by day to get the date. This gives us the day in around 40 questions.
We can do even better. Rather than going for the month first, we should realize that months are an arbitrary human division. There are 365 (or 366) days in the year; let's assume it's not a leap year and use 365. Our first question should be: Is it July 1? Before the question there were 365 possibilities. After this question (assuming we weren't lucky enough to get the date) then we know it's either in the first half or the second half of the year. We've split 365 into two sets of about 180 days. No other choice splits things so well; by asking the middle date, we cut the uncertainty in half. Let's assume we were told earlier. The next question is whether or not April 1 is the date, as this essentially splits the first half of the year in half. We continue along these lines, always guessing the middle of the remaining time. This is known as a binary search, as each guess halves the number of possibilities. We can find the answer in at most 10 guesses; this is because $2^{10} = 1024$ (which is larger than 365).
Once students catch on to the best strategy, ask them what dates would be the best to choose if they want their partner to have to make the most possible guesses. We want to choose a date that requires the most guesses. Thus dates like July 1, April 1, February 14, January 22 are bad; good days would be January 1, December 31.

Binary operations play a key role in making RSA practical. Since $10^3 = 1000 < 1024 = 2^{10}$, we have $10^{200} < (2^{10})^{200} = 2^{2000}$. This means the binary (i.e., base 2) representation of e has about 2000 digits. Using fast exponentiation, $M^e \bmod n$ can be computed in about 2000 multiplications, which is astronomically smaller than 10^{2000} steps! This shows how RSA may be practical. Remember 10^{200} is well beyond anything that can be calculated; there aren't even 10^{100} subatomic particles in the universe. It's impossible to implement anything needing that many calculations. However, 2000 multiplications of 200 digit numbers can be accomplished without too much trouble.

The implementation issues aren't the only issues. There are also security concerns.

1. Identity Validation: Just because the message says it comes from Bob doesn't mean it's from Bob. Imagine Alice receives a message saying:

   Help! I'm in trouble and I need $5000 fast! Please wire the money to the following account. I owe you. Thanks, your friend, Bob.

What should Alice do? If you say wire the money immediately, you have a kind heart but are likely to have an empty wallet. This looks like the dreaded spam messages that populate inboxes around the world. Alice should, reasonably, want Bob to confirm his identity before opening up her wallet.

2. Security: Imagine Eve the Eavesdropper intercepts Bob's message. Is she able to decode it? If Eve can pretend to be Bob, can she use that message to perpetrate other falsehoods?

The first issue will have to wait for another module. The second issue: Eve intercepts Bob's message and wants to decode it. All she needs to do is figure out the secret *d* and she's in, as she already knows *n*. How easy is it to find *d*? Remember what's publically posted: *n* and *e*. Alice kept *p*, *q*, and *(p-1)(q-1)* secret. Eve is trying to solve the equation $de = 1 \bmod (p-1)(q-1)$, where *d* and (p-1)(q-1) are unknown to her. It turns out that if Eve can figure out *(p-1)(q-1)* then she can easily solve this equation and find *d*. The solution involves the Euclidean algorithm, which is used to efficiently find greatest common divisors. So, Eve is reduced to either determining *d* or *(p-1)(q-1)* from *e* and *n*., not knowing *p* or *q*. If Eve could factor *n* then she could find *p* and *q*, and then it's trivial to find *(p-1)(q-1)*. To date, no one has made public a good algorithm for factoring. There may be one, but if so it hasn't been discovered, or at least not publicly announced.

It's worth dwelling on this issue for a bit. The key to many of the modern encryption systems is the trapdoor principle. The idea is that there is a process which is easy if you know some extra information, but hard otherwise. For RSA, it's factoring. If *n* is the product of two primes, it's easy to factor if you know one of the divisors; the belief is that without such knowledge there's no easy way in general to find the other. This may be false. Perhaps there is a simple method that's waiting to be discovered, or perhaps some agency knows it and is keeping quiet!

For example, think back to factoring quadratics. This was briefly discussed when studying permutation ciphers. Given $x^2 + bx + c$, if we know one factor then it's easy to find the other. What if neither factor is known? When first learning about factoring quadratics, you learn to look for two factors $(x – r_1) (x – r_2)$ such that $(r_1 + r_2) = b$ and $r_1r_2 = c$. This is often done by inspiration. Given $x^2 + 9x + 14$, the roots have to add to 9 and multiply to 14. There aren't too many possibilities that multiply to 14 *if the roots are integers!* The only possibilities are *(1, 14)* and *(2, 7)*, and a little work shows the second is the solution, since $2+7 = 9$. However, given $x^2 + 1701x – 24601$, *24601* factorizes as (*1,24601*) or (*73, 337*), and neither 1+24601, nor 73 + 337 is equal to *1701*. This means that the roots are not integers. Without additional mathematics knowledge, it is quite difficult to find the roots. However, after taking algebra, it is understood that the roots of $ax^2 + bx + c = 0$ are $(-b \pm \sqrt{(b^2 – 4ac)}) / 2a$. Thus, even though at first it looked like a daunting task to find the roots to a general quadratic, it is actually quite easy. We must always be alert and on guard against this in cryptography. Unfortunately, it's very hard to prove that there isn't an easier way to do a problem that you've just happened to overlook!

Homework Problems.

#1. Let's revisit the pirate problem, with the only change being that there are now three islands with three stranded people (Alice, Bob and Cameron), each having their own lock. Instead of a ring Bob wants to send a message to Alice and Cameron without the pirates reading it. It is fine if Alice and Cameron know that the other knows the message; all that matters is that the pirates do not know the message. What is the fastest way to do this? What if we had four people (Alice, Bob, Cameron and Danny) and now Bob wants to send the secret to the other three?

Solution: One way is to first send the message to Alice (Bob puts in the message, locks it and sends to Alice, who locks it and returns to Bob, who unlocks it and sends it back to Alice, and then Alice removes her lock, taking three trips on the pirate ship), and then send the message to Cameron. Unfortunately, if Bob needs the message back from Alice, it takes awhile to send the message back to him. A better way is to have Alice then send the message to Cameron. Thus, after Alice removes her lock and reads the message, she immediately locks it again. If we had four people we could have Bob send the message to Alice, then Alice sends the message to Cameron who then sends the message to Danny.

#2. Find the greatest common divisors of the following pairs: (12, 30); (53, 91); (15, 66).

Solution: One way to solve these problems is to factor (as every integer at least 2 can be written uniquely as a product of prime powers), another is to use the Euclidean algorithm. (1) If we factor we get $12 = 2^2 * 3$and $30 = 2 * 3 * 5$; thus the greatest common divisor is $2 * 3 = 6$. We can also use the Euclidean algorithm; WolframAlpha implements this for us and it suffices to type GCD[12, 30]. (2) As 53 is a prime number and does not divide 91, the greatest common divisor is 1. Note we do not need to factor 91 (we can, it's $7 * 13$). (3) We have $15 = 3 * 5$ and $66 = 2 * 3 * 11$; thus the greatest common divisor is 3; we could also type GCD[15, 66].

#3. Let $p = 3$, $q = 5$ and $d = 7$. Find a value of e such that $ed = 1 \bmod (p-1)(q-1)$, or show that no such d exists. What if $p = 3$ and $q = 5$ but now $d = 6$? Is there an e such that $ed = 1 \bmod (p-1)(q-1)$?

Solution: (1) We want an e such that $7e = 1 \bmod 2 * 4$, or $7e = 1 \bmod 8$. Going through the possibilities (in other words, trying all numbers from 1 to 7), we see that only 7 works, which is the answer. We could also try to do this by using WolframAlpha; however, it doesn't like using `e' as a variable, so it's better to type Solve[Mod[7 x, 8] == 1, x]. (2) We now want to find e so that $6e = 1 \bmod 8$. Now there are no solutions. To see this, if there were an e such that $6e = 1 \bmod 8$ then there would have to be an integer k so that $6e = 1 + 8k$, or $6e + 8k = 1$. As the left hand side is even and the right hand side is odd, there cannot be a solution.

#4. In RSA we choose two primes p and q and keep them secret, but we make $n = pq$ public. Later we choose a pair of integers d and e such that $ed = 1 \bmod (p-1)(q-1)$. To find e and d we need to compute $(p-1)(q-1)$. Imagine we are careless and we accidentally leave the value of $(p-1)(q-1)$ in a public space, and Eve discovers it. She now knows n and $(p-1)(q-1)$. Can she easily figure out what p and q are, and if so, how? Hint: look for a solution other than factoring. For example, what if she knows the two numbers are $n = 143$ and $(p-1)(q-1) = 120$, can she find p and q without factoring?

Solution: Yes, she can find p and q. We assume we know N and (p-1)(q-1); let's write R for (p-1)(q-1). We want to find p and q. Since their product is N, we get q = N/p. Substituting gives R = (p-1)(N/p – 1) = N – N/p – p + 1. If we multiply through by p we find Rp = Np – N – p^2 + p, or p^2 + (R – N – 1)p + N = 0. This is a quadratic in p, and we know the coefficients. We can thus find p by the quadratic formula; it's

$$p = (-(R - N - 1) \pm \sqrt{}(R - N - 1)^2 - 4N) / 2.$$

By symmetry we can't tell the difference between p and q; the positive root above gives p and the negative gives q (or we could of course switch them around, or once we have p we can find q from q = N/p). Using R = 120 and N = 143, the formula above gives p = 13, and we then deduce q = 11.

#5. Imagine moving to Pluto, where it takes almost 100,000 days for Pluto to complete one orbit around the sun. For simplicity, just use 100,000 days as a Plutonian year. When playing the yes-earlier-later question game, how many questions will it take to figure out the chosen day? In general, if there are *n* days in a year, about how many questions will it take?

Solution: After the first question we split the set into two groups of 50,000 days. The next question splits into a region of 25,000, while the third then splits into 12,500 days. We continue, with question four reducing us to a window of size 6,250 days, question six gives us a set of 3,125 days, and so on. The solution is we look at the smallest power of 2 larger than our number (in this case it's 17 as 2^{17} = 131,072, while 2^{16} is 65,536) and add one. Thus the answer is 18.

More generally, if we have N days in the year we find the smallest k such that $2^k > N$ and add one. Thus k is approximately $1 + \log_2(N)$.

Exercises 6 and 7 are not necessary for the use of RSA algorithm, but highlight key mathematical ideas behind RSA. These may be used as further explorations for advanced students.

#6. There are very few theorems in mathematics that are called "fundamental." That label is not given out lightly. One such theorem is the Fundamental Theorem of Arithmetic, which states that every positive integer greater than 1 can be uniquely factorized as a product of prime powers. This means we can write a number *n* as $n = p_1^{r1}\, p_2^{r2} \cdots p_k^{rk}$. For example, $120 = 2^3 \cdot 3 \cdot 5$, and all the ways of writing 120 as a product of prime powers have exactly three factors of 2, exactly one factor of 3 and one factor of 5. Find the prime factorizations of 91, 93, 101, 117 and 1800.

Solution: We start dividing these numbers with 2 and see how many powers of 2 go into the number. We then continue with the next prime, 3, and continue incrementing our primes until the number is factored. We can also type Factor[91] into WolframAlpha. We find $91 = 7 * 13$, $93 = 3 * 31$, 101 is prime, $117 = 3^2 * 13$, and $1800 = 2^3 * 3^2 * 5^2$.

#7. (The Sieve of Eratosthenese) The following is a way to find all the primes up to a given number *n*. We often take N to be a square (say M^2), and write the first N numbers in an M by M grid. Cross out 1, as 1 is not a prime. The next number is 2. Circle 2, and then cross out all the remaining numbers that are a multiple of 2. Look for the next non-crossed number, which is 3. Circle that, and then cross out all multiples of 3. Some numbers, like 6, are already crossed out. That's okay. Again look for the first non-crossed number. As 4 is crossed out, move on to 5 which is not crossed out. Circle that, and then cross out all multiples of 5. Continue this process until all numbers are either circled or crossed out. The circled numbers are all the primes at most N. Use this process to find all the primes at most 100.

Solution: The reason this works is that every integer at least 2 is either prime or a divisible by a prime. If we reach a number and it hasn't been crossed out, it can't be composite (if it were composite, it would be divisible by something smaller, and we would have crossed it out). Thus, any number we reach that hasn't been crossed out is a prime. Another way of viewing this is that whenever we reach a new prime we immediately remove all of its multiples, so whatever is left cannot be divisible by that prime. We start by circling 2 (as it's prime) and then remove all of its multiples. Thus no number remaining can have 2 as a factor. We then reach 3; as it's uncircled it must be a prime. We circle it and then remove all of its multiples, which implies that all remaining numbers are not multiples of 2 or 3. The next number on our list is 4, which has been removed. We then continue and reach 5, which is not circled and is thus a prime. Going through this procedure yields all the primes at most 100. What is particularly nice is that we need only check multiples of the primes up to 7.

Day 4: Implementing RSA

It's now time to implement RSA to encrypt and decrypt messages. We'll first discuss finding the needed parameters (two large primes p and q and an integer e relatively prime to (p-1)(q-1)), and then show how to do the steps.

There are a number of algorithms to determine if a number is prime. Remember, x is prime if and only if all the numbers from 2, 3, …, x-1 are relatively prime to x. Equivalently x is composite if there are two integers $a, b > 1$, such that $x = ab$. Note 1 is neither prime nor composite. It is called a unit.

Activity: Have the students break into groups and come up with a way to tell if a number is prime. Emphasize that the process may be SLOW! That's okay. Showing the existence of a solution is the first step in mathematical problem solving. After existence is shown, then efficiency of the solution is considered.

Method 1: The first approach is the naïve, brute force approach. Keep trying to divide n by all the numbers less than n. If none divide n, then n is prime.

This is a nice, simple method. It's easy to use, and it does correctly determine if a number is prime. Unfortunately, it's too slow to be practical. The bigger p and q are, the harder we expect it to be to factor $n = pq$, and the longer it'll take us to check all possible factors. When p and q have 200 digits, to check all possible factors is well beyond current computational capabilities. A better approach is needed. For example, if $p < q$ then we have to check 2, 3, …, p before finding a factor. If p has 200 digits, that's around 10^{200} numbers!

Activity: Have the students meet again in groups and try to find a better approach.

Method 2: We don't need to check all of the integers up to x; it's enough to check just up to $x/2$. Let x be a composite number, so we may write $x = ab$, and, for definiteness, we can assume $a \leq b$. Since x is composite, a has to be at least 2. As $b = x/a$, we see b is at most $x/2$. Thus the largest factor of x (if it's composite) is at most $x/2$.

At first you might feel like celebrating. After all, this prunes the number of factors to check by 50%. That's pretty good! Unfortunately, it's 50% of an astronomically large number, and $5 * 10^{199}$ is still far too large to be approachable. The savings is negligible – there are still too many possible factors to check.

Fortunately, the search can be refined even further. If $a \le b$ then $a^2 \le ab = x$, or $a \le \sqrt{x}$. This means that the *smallest* factor of x is at most $\sqrt{x}$. In other words, if all the integers up to $\sqrt{x}$ are checked and none divide x, then x is prime. Remember, once one factor is found, the check is done, no more numbers need to be checked.

This translates into *real* savings. Instead of 10^{200} we have $\sqrt{10^{200}}$, or 10^{100}. While this number is still too large to be practical, it's at least significant and gives hope that further progress can be made. Note that all of these methods can be improved by a little bit. It's a bit wasteful to check all numbers up to a given size. If it is known that 2 doesn't divide x, then neither can 4. Similarly, if 3 does not divide, 9 doesn't divide. It's enough to check just primes as factors.

Further progress can be made, but the methods are beyond the scope of this unit. For those who wish to investigate further, the following methods can be researched:

- Fermat's factorization method
- Euler's factorization method
- Miller-Rabin test
- AKS primality test

You can go to WolframAlpha (http://www.wolframalpha.com/) and type in the box NextPrime[34850386039486093850285208520985O2] (or some other large number), and it will output the next prime. After you find the next prime, find the next one after that, and then the next few. It's interesting to see that, even though the number you have chosen may be enormous, the distance between primes is pretty small. On average, if x is prime then the distance to the next prime is approximately $ln(x)$ (the natural logarithm of x). If x is about 10^{200}, $ln(x)$ is only about 460.

This is a fairly easy way to find primes. Not surprisingly, a few hours in a high school classroom isn't enough to reach the state of the art methods. Hopefully, though, you can see that checking for primality isn't as bad as first feared. Further, if you numerically explore where the primes are on WolframAlpha, you'll see that they are reasonably close to each other. What this means is that if a number chosen is not prime, testing other numbers in its vicinity will surely find a prime number.

Next, given e and $(p\text{-}1)(q\text{-}1)$ we must find d with $ed = 1 \bmod (p\text{-}1)(q\text{-}1)$. This is actually a specific instance of a more general problem: solving $xy = 1 \bmod z$, given x and z. Unfortunately, this equation isn't always solvable. If x and z share a common factor, for example, $x = 21$ and $z = 35$, then for a solution to exist there must be an integer k such that $xy = 1 + kz$. Subtracting kz from each side gives $1 = xy - kz$. If however x and z share a common factor, then that number divides the right hand side (in the example, $xy - kz$ is $21y - 35k = 7(3y – 5k)$, so the right hand side is divisible by 7). However, if the right hand side is divisible, then so is the left. This is a contradiction, as the only divisor of 1 is 1! Thus, if *x* and *z* share a divisor, there are no solutions.

If there are no common factors it turns out that there is always a solution, and one way of finding that solution is to just keep trying numbers until one works. This is similar to our first naïve attempts at factorization. It'll work, but it'll take too long to be practical. Remember the numbers are on the order of 10^{200}, so this would mean checking as many as 10^{200} numbers! It's worth mentioning that if there is a solution *y* to $xy = 1 \bmod z$, then *y* cannot be unique. Any number of multiples of *z* can be added to *y* without changing the congruence. Thus, there are either no solutions, or infinitely many.

Fortunately there is a very fast way to find the answer. It's called the **Generalized Euclidean algorithm**, and it runs in logarithmic time. This means that the time it takes to run is polynomial in $\ln(10^{200})$. As mentioned earlier, while 10^{200} is enormous, its logarithm is only about 460, which is quite small and manageable.

Teacher Note: The next paragraph deals with using WolframAlpha (or some other program) to compute modular inverses. WolframAlpha accomplishes this by using the Generalized Euclidean. As these notes do not describe how the Generalized Euclidean Algorithm works, this is an excellent place to incorporate the square-root argument from the teacher's introduction. Students have used the square-root function on their calculator for years without knowing the nuts and bolts of how it works. While it is of course preferable to know why an algorithm works, for many purposes it is fine to just use it. It may be worth reminding students of this fact.

The Generalized Euclidean Algorithm is a method for computing modular inverses. Similar to Fast Exponentiation, it runs very quickly. You can use the algorithm without having to know what it does by going to *WolframAlpha.com*. To solve for *x*, choose values for *d* and *n* and type *Solve[Mod[x d, n] == 1, x]* into WolframAlpha (note that == is two equal signs next to each other). The output is all values of x such that $x * d = 1 \bmod n$. When you run this code, in addition to having the general solution listed, WolframAlpha will also provide a specific solution. For example, if you type *Solve[Mod[x 1345, 1992] == 1, x]* the specific solution is 2617. Note that WolframAlpha did **not** reduce this modulo 1992. You will have to do the reduction yourself getting the minimal solution of 625.

The final step is to choose a message. This message will take the form of a number, *M,* from 0 to *n-1* so long as *M* is relatively prime to *n*. How restrictive is this condition? Fortunately very few numbers share a factor with $n=pq$. Of the *n* numbers from 0 to n-*1*, exactly $n/p = q$ are multiples of *p*, $n/q = p$ are multiples of *q*, and $n/pq = 1$ are multiples of both (these last have been double counted). Thus, the number of bad numbers is $p + q - 1$ out of *pq*. For large values of *p* and *q*, a very small fraction of choices will fail. For instance, if $p < q$ then the number of bad numbers is less than *2q*, so the fraction of bad *M* is at most $2q/pq = 2/p$. If p is enormous (think 10^{200}), this probability is so small it's unlikely to happen. To put things in perspective, some lotteries have you choose 6 numbers out of some set; for definiteness let's say there are 36 possibilities. You win if you get the six numbers right, and order doesn't matter. The odds of getting all six are one in 1,947,792. Your probability of winning this lottery each week for 30 straight weeks is about 1 in 10^{188}. This means it is more likely you will win a lottery 30 weeks in a row than it is you will choose a message that is not coprime to *n*. That said, it's still worthwhile to be safe. We can leave the last few digits of our message *M* free so that if our initial choice fails, we can modify it slightly and the new message will be good.

Example of RSA implementation:

Bob wants to send a message to Alice. Alice has to start the process by creating her public (*e* for encryption) and private (*d* for decryption) keys and then making her public key available. The act of making a public key available is called "publishing." Once she has published her public key, then anyone can send a message to Alice using Alice's public key to encrypt the message. Anyone else who intercepts the message will not be able to decrypt it because they do not have *d*, the decryption key.

Step 1. Alice chooses two primes, $p = 43$ and $q = 59$ and calculates $n = pq = (43)(59) = 2537$. These are much smaller primes than she would take for a real world application where she cares about security, but they will be easier to use than large numbers, to show how the method works.

Step 2. Alice must choose *e* such that $1 < e < (p-1)(q-1)$ and *e* is relatively prime to *n*. If she chooses a prime number for *e*, then she only has to make sure that *e* does not evenly divide *n*, which is easier than seeing if two composite numbers are relatively prime. She chooses $e = 79$, a prime number that does not divide 2436 evenly. She now needs to find *d*, which turns out to be 2251. WolframAlpha uses the Generalized Euclidean Algorithm and gives the infinitely many solutions. By typing *Solve[Mod[79 d, 2436] == 1, d]* the output is *2436 n + 2251.* There is no reason not to take the simplest *d* and thus the solution with $n = 0$ (so $d = 2436\ (0) + 2251 =$ **2251**). Checking shows that: $ed = 79 * 2251 = 177829$, and 177829 = 1 modulo 2436.

Step 3. Alice publishes the numbers $n = 2537$ and $e = 79$.

Step 4. Bob first must encode the message he wished to send. The message is "hi." From the encoding table Bob finds:
h = 08
i = 09

His message encodes to:
809
He checks to be certain that his message, 809 is relatively prime to 2537. Typing *gcd[809, 2537]* into Wolfram Alpha returns 1, showing the two numbers are relatively prime. Bob is ready to calculate the message to send. He next computes $X = M^e \bmod n$. This is 809^{79} mod 2537, which looks quite intimidating for two reasons. One is that if Bob were to multiply 809^{79} by brute force, the answer is approximately 10^{260}! For real world applications, it's even worse as typically *e* would have 200 digits. In

addition to being a large number, he needs to do far too many multiplications to be practical. There isn't enough computer power on the planet to do 10^{200} operations. He needs a better method than brute force multiplication. Fortunately, Bob knows **fast exponentiation**, which helps make RSA practical.

Teacher Note: The text below describes how to quickly compute high powers of a number. The main takeaway for students should not be the nuts and bolts of the method, but rather that a method exists to get the answer in significantly less time than the naïve multiplication they know. It's okay if students are unfamiliar with binary expansions. If time permits this is a possible topic, or something for advanced students to explore, but it is not essential to the unit.

The easiest way to explain fast exponentiation is through the needed computation, $M^e \mod n$. The basic idea is to write the exponent e in binary (base 2), and use that expansion to multiply efficiently. This requires the binary expansion of 79 to encrypt, and 2251 to decrypt. The easiest way to determine the base 2 value of a decimal number is to let WolframAlpha calculate it. Typing "79 decimal to binary" (without the quotes) returns the value 1001111_2 for e. Similarly typing "2251 decimal to binary" (again, without the quotes) returns the value 100011001011_2 for d.

To calculate the value of 809^{79} modulus 2537 in the naive method would take 78 (79-1) multiplications. Even reducing by the modulus after each multiplication, this is going to be very involved computationally. It would be better to find a more efficient method of exponentiation.

We can take advantage of the fact that $809^{79} = 809^{64}\cdot 809^{8}\cdot 809^{4}\cdot 809^{2}\cdot 809^{1}$. The exponents correspond to the powers of two in the binary form of 79. That is, $79 = \mathbf{1} * 2^7 + \mathbf{0} * 2^6 + \mathbf{0} * 2^5 + \mathbf{1} * 2^4 + \mathbf{1} * 2^3 + \mathbf{1} * 2^2 + \mathbf{1} * 2^1$. (Notice that the bold numbers in the previous expansion correspond to the binary expansion of 79.)

$$\begin{aligned}
809^{79} &= 809^{64}\cdot 809^{8}\cdot 809^{4}\cdot 809^{2}\cdot 809^{1} \\
&= (809^{32})^2\cdot(809^4)^2\cdot(809^2)^2\cdot 809^2\cdot 809 \\
&= ((809^{32}\cdot 809^4\cdot 809^2)\cdot 809)^2\cdot 809 \\
&= ((809^{16\cdot 2}\cdot 809^{2\cdot 2}\cdot 809^2)\cdot 809)^2\cdot 809 \\
&= ((809^{16}\cdot 809^2\cdot 809)^2\cdot 809)^2\cdot 809 \\
&= ((809^{8\cdot 2}\cdot 809^2\cdot 809)^2\cdot 809)^2\cdot 809 \\
&= (((809^{8}\cdot 809)^2\cdot 809)^2\cdot 809)^2\cdot 809 \\
&= (((809^{4\cdot 2}\cdot 809)^2\cdot 809)^2\cdot 809)^2\cdot 809 \\
&= (((809^{4})^2\cdot 809)^2\cdot 809)^2\cdot 809)^2\cdot 809 \\
&= ((((809^{2\cdot 2})^2\cdot 809)^2\cdot 809)^2\cdot 809)^2\cdot 809 \\
&= ((((809^{2})^2)^2\cdot 809)^2\cdot 809)^2\cdot 809)^2\cdot 809.
\end{aligned}$$

This expansion leaves us having to do 10 multiplications, rather than 78. Even at that, it is not a calculation for the faint of heart. It would be best to use a spreadsheet, a programmable calculator or WolframAlpha to do the work for us.

Bob types "809^79 mod 2537" (without the quotes) into Wolfram Alpha and it returns the result 1741. This is *X*, which Bob sends to Alice.

Step 5. Alice receives the message 1741 from Bob. She has to decrypt it first by calculating $M = X^d \bmod n$. Alice is also going to need fast exponentiation (or a program that uses it) to do the calculation. She types "1741^2251 mod 2537" (without the quotes) into WolframAlpha and gets the value 809. She pads this with a leading 0 (since all of the characters encode to two digits, our message has to be an even number of digits long) getting 0809. She can then decode the numbers to characters and recover the original message "hi."

Homework problem: what would it mean if Bob's message was NOT relatively prime to *n*? We saw it's extremely unlikely that his *M* has a common factor with *n*, but what if it does? What are the consequences? Answer: Bob can now pretend to be Alice! The only non-trivial factors of $n = pq$ are *p* and *q*. The Euclidean algorithm allows Bob to find his common factor with Alice's *n*; it's either *p* or *q*. Once he knows one he can find the other, and thus he knows $(p-1)(q-1)$. Knowing that and *e* he can now find *d*. In other words, he can pretend to be Alice and read ALL her mail!

Homework Problems.
#1. Using any of the methods discussed in class, determine which of the following numbers are prime: 91, 101, 117, 199, 1235, 1237.

Solution: We can attack these by brute force, trying all factors up to the square-root of the number. We could also do the Sieve of Eratosthenese, and determine 91, 101, 117 and 199 without too much difficulty. Finally, we can type PrimeQ[number] into WolframAlpha and it will tell us if the number is prime or not. We find 91 is composite (it is $7 * 13$), 101 is prime, 117 is composite (it's divisible by 3; we don't need its full factorization, so we can stop here), 199 is prime, 1235 is composite (it's a multiple of 5), and after some work we see 1237 is prime (its square-root is a little more than 35, so we need only check whether or not it is divisible by primes up to and including 31).

#2. Read about Fermat's or Euler's factorization method, and use one of them to determine if 117 and 199 are prime.

Solution: See http://en.wikipedia.org/wiki/Fermat%27s_factorization_method and http://en.wikipedia.org/wiki/Euler%27s_factorization_method. We want to find a and b so that $a^2 – N = b^2$; if such a pair exists then $N = a^2 – b^2$ and we get the factorization $N = (a-b)(a+b)$. If we take $N = 117$ then its square-root is about 10.8, so the smallest a we may take is 11. If we try $a = 11$ we get $11^2 – 117 =$

121 – 117 = 4, which is a square. Thus a = 11 and b = 2, for the factorization 117 = (11-2)(11+2) = 9 * 13. Note that this is not a complete factorization of 117, but it suffices to see that it is composite.

If we now take N = 199, then its square-root is a little over 14, so the smallest possible a is 15. If we try Fermat's method, we find there are no a and b that work. Note that a and b are not allowed to be arbitrarily large. If such a pair exists then N = (a-b) (a+b), and thus clearly a + b must be at most N. Thus we need only check a up to N (with a little more work we see we don't need to check so high).

#3. Find all the primes from 1500 to 2000, and make a histogram plot of the distance between adjacent primes. For example, the primes from 10 to 20 are 11, 13, 17 and 19, and the distances are 2, 4 and 2 again.

Solution: The primes are

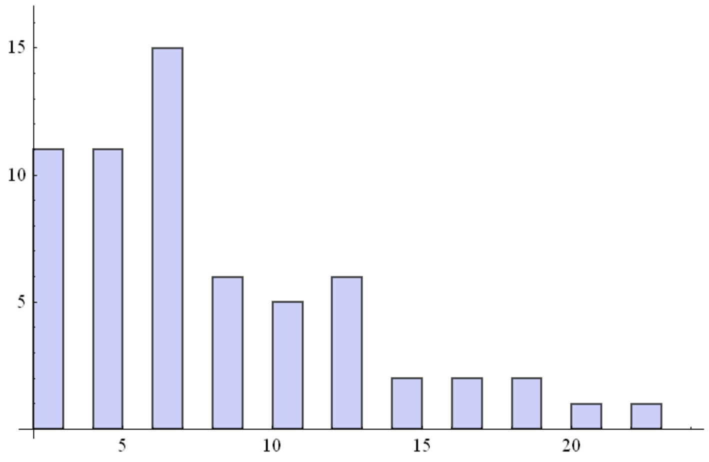
{1511, 1523, 1531, 1543, 1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619, 1621, 1627, 1637, 1657, 1663, 1667, 1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787, 1789, 1801, 1811, 1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997, 1999},

and the spacings between the primes are

{12, 8, 12, 6, 4, 6, 8, 4, 8, 4, 14, 4, 6, 2, 4, 6, 2, 6, 10, 20, 6, 4, 2, 24, 4, 2, 10, 12, 2, 10, 8, 6, 6, 6, 18, 6, 4, 2, 12, 10, 12, 8, 16, 14, 6, 4, 2, 4, 2, 10, 12, 6, 6, 18, 2, 16, 2, 22, 6, 8, 6, 4, 2}.

Sorting the spacing list gives

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 8, 8, 8, 8, 8, 8, 10, 10, 10, 10, 10, 12, 12, 12, 12, 12, 12, 14, 14, 16, 16, 18, 18, 20, 22, 24}.



Plot of the spacings between adjacent primes from 1500 to 2000.

#4. Find the binary expansions of 3, 33, 333 and 3333. Are you surprised that the binary expansions have more digits than the decimal expansions? Why or why not?

Solution: 3 = 2 + 1, which is 11 in base 2. Similarly 33 = 32 + 1; as 32 is 2^5, we get 100001 base 2. We continue by subtracting the highest power of 2 possible. We can also use WolframAlpha by typing BaseForm[number, 2] to get the binary expansion. We get 333 is 101001101 base 2, and 3333 is 110100000101 base 2. It's not surprising that we have more digits base 2, as 2^d grows far more slowly than 10^d.

#5. Use fast exponentiation to find 11^{15} modulo 31.

Solution: In base 2, 15 is 8 + 4 + 2 + 1, so we need all powers. We find $11^2 = 11 * 11 = 28$ modulo 31, $11^4 = 11^2 * 11^2 = 28 * 28 = 9$ modulo 31, and $11^8 = 11^4 * 11^4 = 9 * 9 = 19$ modulo 31. Thus $11^{15} = 11^8 * 11^4 * 11^2 * 11 = 19 * 9 * 28 * 11 = 30$ modulo 31.

#6. (Advanced) We use the binary expansion of the exponent for fast exponentiation. Why do we use binary expansions and not decimal expansions?

Solution: Binary expansions have many particularly nice features. We either have a factor or we don't as the digits in a binary expansion are only 0s and 1s.

Day 5: **Information Versus Disinformation**

Teacher Note: One option is to assign these pages to students to read at home, and use as a springboard for a conversation about information and disinformation. What are some other examples? What are issues with implementation?

To this point, our concentration has been on how to encrypt. The other side of this coin is what should be encrypted. The phrase "Information Age" is used often to describe modern times and we are told that information is the new currency. What exactly does this mean and why is it relevant? First, it's important to realize that there are two parts to information. It's not enough for something to be known; it needs to be quickly accessible. A hundred years ago movies were flip frames (see http://en.wikipedia.org/wiki/American_Mutoscope_and_Biograph_Company and http://en.wikipedia.org/wiki/Animation); no sound, and each frame was drawn by hand. Things then advanced to moving pictures (http://en.wikipedia.org/wiki/Moving_picture), but of course just in black and white, with the dialogue confined to cards every few moments. Things improved in the 1920s when sound was added, but you still had to go to a movie theatre. Then television (http://en.wikipedia.org/wiki/Television) breaks into the scene, and allowed people to watch programs in their home. For many years there weren't that many choices, but soon cable channels appeared. And now you can watch streaming video on your phone or iPad, and if the download takes more than a few seconds, you're frustrated (and justifiably so!).

These issues have been with us for millennia, but in a post-9/11/2001 world, they're a bit more on our minds. The following scenario is quite likely to happen in the near future: An aide runs into the Oval Office, interrupting a meeting the president is having with key congressional leaders. He gives a nod and the president ends the meeting. The aide turns to him with the latest news. There has been a huge spike in chatter from suspected Al-Qaeda cells. Homeland security fears a go signal has been sent, and an attack is imminent. The problem is where and when?

The U.S.A. can't afford to spend billions of dollars to thwart attacks that costs millions, or even thousands, of dollars to plan and execute. The country's leaders need to know how to deploy its forces efficiently. Consider how many cities, airports, bridges and harbors there are in the U.S. It just isn't practical to have each of these on constant alert. Unfortunately, the terrorists communicate in codes, trying to keep secret their choice of target and date. Often national security agencies have copies of their transmissions, but have to figure out how to read them. Additionally, deciphering a message isn't enough. The knowledge must be acted upon. The big question is, are the message contents true, or was this a ruse to mislead?

The following example is a beautiful combination of the two points we've just mentioned; it deals with the need for speed, and the issues that arise when messages are encoded. Moreover, as we'll see shortly, it also includes a third part of information theory, one that is easily overlooked but is extremely important nevertheless.

In the spring of 1942 America has been at war with Japan for a couple of months, and it has not been a good few months for the United States. The war began on December 7, 1941, when the Japanese launched a surprise and successful attack on Pearl Harbor, shocking America. They destroyed much of the fleet moored there, but missed the aircraft carriers which were all at sea. In the following months, Japanese forces continued to advance in the Pacific, capturing (among others) the Philippines, Malaya, Singapore and Indonesia. While the war was not going well for America, many Japanese were justly afraid of their long term prospects. As America shifted to a war footing, its mighty industrial power would churn out huge numbers of ships and tanks and planes. If the war were to be protracted, the Japanese would lose. Their goal was to go for a quick, decisive victory to knock America out of the war before these superior forces could enter the battle.

The Japanese planned to attack Midway in early June 1942. The island is a little over 1000 miles west of Hawaii. The Japanese commander, Yamamoto, hoped to deceive the Americans as to his true intentions, and engage and destroy the U.S. carriers on his terms. His hope was a conclusive Japanese victory which could lead to a demoralized America losing its will to fight, or at the very least greatly enhance Japan's strategic position, by giving them a base within striking distance of the U.S. west coast and removing the American carrier threat.

Unknown to the Japanese, however, U.S. code-breakers had deciphered much of code being used to plan the battle and convey the orders. The Americans knew a major attack was being planned. Unfortunately, the target was listed as "AF" in the messages and the U.S. code-breakers did not know what "AF" meant. Imagine you're Admiral Nimitz of the U.S. Navy. You have to decide how to deploy your forces. Where do you send them? Are the Japanese going for the West Coast cities? Are they preparing for an attack on Alaska or Hawaii or Australia? How was Nimitz to decide where to deploy his carriers? "AF" was thought to be Midway. This was a logical target, and the Japanese had used the "A" designation for islands near Hawaii. It was a big gamble for him to deploy his forces based on Midway as the target because the Japanese had so many options. Nimitz needed to know where the Japanese fleet was going. As promised, this problem involves the two key ideas we've discussed so far. Speed is clearly important, and the messages are encoded. But there is also a third component to this story, which is a key part in the battle for information, namely *dis*information. The U.S. found a solution to determining the identity of "AF." Midway was contacted through secure means, and told to report in an uncoded message that the water distillation plant on the island was damaged and that fresh water was needed immediately. Soon afterward, Americans intercepted encrypted Japanese messages. When deciphered, some of these mentioned that "AF" was low on water! The mystery of "AF" was solved, and Nimitz knew where to deploy the carriers. (http://en.wikipedia.org/wiki/Battle_of_Midway, http://www.nsa.gov/about/cryptologic_heritage/center_crypt_history/publications/battle_midway.shtml).

Disinformation isn't limited to the military. It occurs in many forms in numerous fields. Consider, for example, the problem of choosing players for a baseball team. There are several sources for players. The most obvious are the current members of the team, who are often under contract to play for the team again next year. Another source are the minor league teams affiliated with the major league ballclub, where young prospects are nurtured and trained to play in the pros. Another important source are the free agents. These are often star or superstar players, usually from another team, who's contract has expired and are available for the right price. Frequently these players are from weak teams with limited budgets and almost non-existent playoff hopes, and the players are typically striving to get the largest possible contract (though sometimes they're willing to take less money in order to be on a team that has a chance of winning it all).

Each team must decide how much a given player is worth. Do they want to make the player an offer? If so, how much? A team faces two dangers: they could overpay for a player, or they could lose out to a team that offers more. While a complete analysis of this problem involves game theory, there is a disinformation component.

What can happen is that a team may pretend to be interested in a player they don't want in order to make him more expensive to another team. This can have two effects if done well. The other team ends up overpaying for a player, which then results in the team having less money available for other signings. A recent example of this is Carl Crawford. He was a free agent after the 2010 season, posting good numbers with Tampa Bay (he batted .307, with an on-base percentage of .356 and a slugging percentage of .495, while at the same time successfully stealing 47 times out of 57 attempts). The Boston Red Sox were very interested in him, and so too were the New York Yankees. In order to get Crawford, the Red Sox made him a large offer: $142 million dollars over 7 years! The Yankees made motions indicating that they were also interested in Crawford, and drove up the price. It later came out that the Yankees were ***not*** interested in Crawford, and this was a (successful) attempt to get the Red Sox to overpay for him, and thus have less ability to pursue other players.

This strategy worked out beautifully for the Yankees. Crawford posted poor numbers in Boston (his batting average, on-base percentage and slugging lines were .255, .289 and .405 in 2011 and .282, .306 and .479 in 2012). In the end, the Red Sox wanted to unload him and free up the money owed him to be used for other players. They were able to do this in 2012 by trading him (and his expensive salary) to the Los Angeles Dodgers, but at a high cost: to get the Dodgers to take him, the Red Sox had to give up one of their performing stars, Adrian Gonzalez (whose line in Boston was .338, .410 and .548 in 2011 and .300, .343 and .469 in 2012).