Ciphers Where Alice and Bob Need to Meet
Exposition by William Gasarch

We will use three characters: Alice and Bob who want to communicate secretly, and Eve who wants to see what they are talking about. Alice and Bob do not want Eve to be able to decode their messages.

1. The *Plaintext* is the message you want to send. For example *Discrete Math Plus Plus is the nickname for this CMSC 389.*
2. The *Ciphertext* is the message after it is encoded. For example *EJTDS FUFNB UIQM VTQM VTJTU IFOJD LOBNF GPSDN TD490.* (See note on this below.)
3. If Alice and Bob want to exhange messages then they need to both know SOMETHING ahead of time. What they know is called a *key.* This will be clearer with examples.

Note 0.1 In the above example of coding *Discrete Math Plus Plus is the nickname for this CMSC 389* I used several conventions:

1. I wrote the message in all capitals. This is a standard convention— the plaintext is in normal font, the ciphertext is all capitals.
2. I wrote the message in blocks of five. This is also a standard convention. If I wrote it like *EJTDSFUF NBUI QMVT QMVT JT UIF OJDLOBNF GPS DNTD 490.* then it would be MUCH easier for Eve to decode. Using the blocks-of-five does NOT make it much harder for Alice and Bob.
3. In the above I shifted the letters by 1. This is a standard cipher discussed below.

Does Eve know what type of cipher Alice and Bob are using? We will assume yes.

Definition 0.2 *Kerckhoff's principle* states that in cryptography, Alice and Bob should assume that Eve knows the type of cipher but not the key.

Why does it make sense to assume this?

1. While very few people know the key, may will know the type of cipher being used. "Loose Lips Sink Ships".
2. If your security is based on Eve not knowing the cipher, then as soon as you find out the cipher has been hacked, you have to change the entire cipher. If your security is just based on the key, you just need to change the key.
3. If you prove that your system is secure even given that Eve knows what type of cipher you are using, then that's a very strong statement.

1 Shift Cipher

Alice and Bob have wanted to exchange secret messages for the last 4000 years. One of the earliest techniques for this, called the *Caesar Cipher*, operates as follows:

First imagine all letters as numbers. A is 0, B is 1, C is 2, etc, Z is 25. Map every letter to the letter that is three higher (modulo 26). So, the the last three letters shift to the first three. Then

A goes to D
B goes to E
⋮
V goes to Y
W goes to Z
X goes to A
Y goes to B
Z goes to C.

More generally, a *shift cipher* is a code where every letter shifts a constant amount.

Lets say that Alice shifts by $s \in \{0, 1, 2, \ldots, 25\}$. We can write this as

$$f(x) = x + s \bmod 26.$$

If Alice uses f what does Bob use to decode? He will use

$$g(x) = x - s \bmod 26.$$

Note that $f(g(x)) = x$. Also note that for ANY choice of s there is a $-s$. We do not actually use $-s$, we use $26 - s$ which accomplishes the same thing.

Are shift ciphers good?

PROS

1. The scheme is easy to describe, easy to code, and easy to decode. So Alice and Bob can operate very fast.
2. Alice and Bob only have to agree on the shift. Since the shift is in $\{1, \ldots, 25\}$, they can easily communicate to each other which shift to use.

CONS

1. The scheme is easy so Eve may spot the pattern.
2. If Eve knows that it is a shift cipher then she can just try all 25 possible shifts. (See later for a fuller explanation.)
3. Alice and Bob do have to meet privately once to agree on the shift. (Is this avoidable?)

Historical Note: This is a very old cipher. Its called *the Ceaser Cipher* and was used by the ancient Greeks. Note that they WOULD NOT have phrased it as Map x to $x + s \pmod{26}$. They did not have the notation and they might have had the notion of mod. They would instead say to do the following (I use the English alphabet- they would have used Greek lettes.)

Take the sequence

$$a \ b \ c \ d \ e \ f \ g \ h \ i \ j \ k \ l \ m \ n \ o \ p \ q \ r \ s \ t \ u \ v \ x \ y \ z$$

If you want to shift by three then place the shifted-by-3 sequence right below it to obtain the following:

$$\begin{array}{cccccccccccccccccccccccc} a & b & c & d & e & f & g & h & i & j & k & l & m & n & o & p & q & r & s & t & u & v & x & y & z \\ d & e & f & g & h & i & j & k & l & m & n & o & p & q & r & s & t & u & v & x & y & z & a & b & c \end{array}$$

That gives you how to encode a message. We leave it to the reader to obtain the table to decode.

The point is that when studying mathematics of an older era its important to keep in mind that they had different ideas then we do now. In these notes we will express things in the modern way, but just be aware that they did not.

Breaking the Cipher:

As noted above Eve could just "try" all 26 possible shifts. But what does that mean? She could, for each shift, decode and see which text "makes sense". But this would be time consuming by hand. Worse- it may be hard to automate. What does it mean to a computer to "make sense?" Is there a better way?

YES! The frequencies of each letter in English is known. (E.g., e is the most common letter). Let p_i be the expected relative frequency of the ith letter in a text. These values are known. Hence $\alpha = \sum_{i=1}^{26} p_i^2$ is known and is 0.065.

Let T be a text. Assume that it was coded with a shift of s. Let q_i be the relative freq of the ith letter in T. Then we expect q_{i+s} to be roughly p_i. Hence $I_s = \sum_{i=1}^{26} p_i q_{i+s}$ will be around 0.065. Also, if s' is NOT the shift then $I_{s'} = \sum_{i=1}^{26} p_i q_{i+s'}$ will be around 0.038.

SO, rather than try to see what shift "looks right" or "makes sense" just compute I_s for $s = 1, 2, \ldots, 25$ and whichever one yields something close to 0.065 is the shift. It is likely there will only be one such.

2 Affine Cipher

We can use a more complicated function. For example

$$f(x) = (3x + 4) \bmod 26.$$

If Alice uses f to code, what does Bob use to decode? He needs to use $g(x) = Ax + B \bmod 26$. We need to find A, B that work We need

$$f(g(x)) = x$$

$$f(Ax + B) = x$$

$$3(Ax + B) + 4 = x$$

$$3Ax + 3B + 4 = x$$

We'll set $3B + 4 = 0$ and $3A = 1$. AH- can we solve those equations? In both cases we need a number whose multiplicative inverse is 3 mod 26. For now we'll try all possibilities (there are faster ways).

$3 \times 1 = 3$
$3 \times 2 = 6$
$3 \times 3 = 9$
$3 \times 4 = 12$
$3 \times 5 = 15$
$3 \times 6 = 18$
$3 \times 7 = 21$
$3 \times 8 = 24$
$3 \times 9 = 27 \equiv 1$

AH- so 9 is the number we seek. We set $A = 9$.

$$3B + 4 = 0$$

Mult both sides by 9 and reduce mod 27.

$$B + 4 \times 9 = 0$$

$$B + 36 = 0$$

$$B + 10 = 0$$

$$B = -10 = 16$$

So Bob uses $g(x) = 9x + 16$.

Does EVERY affine cipher have an inverse? Lets try $f(x) = 2x$. We need an inverse of 2 mod 26. There isn't one— $2x$ is always even mod 26.

It turns out that the coeff of x is relatively prime to 26 iff an inverse exists. So the coeff can be any of $\{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$. The constant term can be anything.

Are these codes good?

PROS

1. The scheme is easy to describe, easy to code, and easy to decode (once you know the trick). So Alice and Bob can operate very fast, though not as fast as with the shift cipher.

2. Alice and Bob only have to agree on the multiplier and the shift. This amounts to knowing one number from $\{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$ and another from $\{0, 1, \ldots, 25\}$. We represent the numbers in base 2. Each number is 5 bits long, so two numbers take 10 bits. This is small.

CONS

1. The scheme is easy so Eve may spot the pattern, though it's not as easy as the Shift Cipher.

2. If Eve knows that it is a affine cipher then she can just try all $12 \times 26 = 312$ possible affine ciphers. Notice that this is harder than for a shift cipher.

3. Alice and Bob do have to meet privately to agree on the parameters. (Is this avoidable?)

3 Quadratic Cipher

One can look at quadratic ciphers, for example:

$$f(x) = (2x^2 + 5x + 9) \bmod 26.$$

These are called quadratic ciphers. They have similar PROS and CONS to affine ciphers. However there is one more serious CON: given a quadratic polynomial it is hard to determine if it has an inverse on $\{0, \ldots, 25\}$. Note that there are more quadratics than affine function so a PRO is that its harder to crack than a affine cipher.

4 Polynomial Cipher

One can look at any polynomial, for example:

$$f(x) = (2x^8 + 5x^2 + 9) \bmod 26.$$

(Frankly I do not know if this would work.)

The higher the degree the harder for Alice and Bob to tell if it has an inverse AND the harder for Eve to try to decode it.

5 General Substituition Cipher

Alice and Bob pick a *random* permutation of $\{A, \ldots, Z\}$. So Alice and Bob meet in a dark alley and generate a random re-ordering of $\{A, \ldots, Z\}$. For example, they may end up with this:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	x	y	z
q	w	e	r	t	y	u	i	o	p	a	s	d	f	g	h	j	k	z	x	c	v	b	n	m

That gives you how to encode a message. We leave it to the reader to obtain the table to decode. (NOTE- I didn't really generate that encoding at random. I leave it to the reader to determine how I did it.)

Is this a good code?

PRO:

Eve has to go through ALL 26! possibilities to crack this code! Hence this code is UNBREAKABLE!!!!!! (Note- this is NOT TRUE as we will see soon. This is typical of the entire history of crypto which can be summarized as:

CODE MAKER: I have an unbreakable code!

CODE BREAKER: I just broke it.

CODE MAKER: Whoops.

Throughout history codes thought unbreakable were devised to thwart any attack that the inventor could think of, but then broken in a way that inventor had not thought of.

CONS: The key Alice and Bob use is a list of the letters of the alphabet in some order. In base 2 this is $26 \times 5 = 130$ bits (though it can be done in somewhat less).

6 Frequency Analysis

Given a text that came from a random permuation how can Eve crack it? NOT by going through all 26! possiblities. That would take too long. But note that e is the most common letter in the English Language. th is the most common two-letter pair. People have compiled tables of such information and these can be used to crack a coded text if its long enough.

Also, if Eve knows you are cracking (say) military codes she may use that to her advantage— slightly different patterns may hold.

In old times (say 2000 years ago) it was still fine to use a random cipher since the computation power to do a Freq. Analysis wasn't there yet. But now it is. Hence in the real world today nobody uses any of the ciphers mentioned above.

All of the ciphers discussed so far are mono-sub ciphers, meaning that they map the alphabet letter by letter. Any such cipher can be broken by a Freq. Analysis.

Its easy to just say *use Freq. Analysis* and this is fine for (say) cryptograms in newspapers. But how would you really code this up?

Writing a program that, given a text, determines how many times each letter appears, is easy.

Given a potential decoding, checking if its correct can be done similar to the methods shown above. You would not want to check all 26! of them, so use Freq. Analysis to cut down on the number of options.

You can also use the most common 2-letter pairs and 3-letter triples as well.

7 Matrix Codes

Here is one way to defeat the Freq. Analysis.

Let A be the following matrix.

$$\mathbf{A} = \begin{pmatrix} 8 & 9 \\ 11 & 7 \end{pmatrix}$$

We can map *pairs of numbers* with this matrix as follows. The pair (x, y) will map to the pair you get by applying the matrix and reducing modulo 26, which is

$$((8x + 9y) \bmod 26, (11x + 7y) \bmod 26).$$

The matrix must have determinant that is rel prime to 26 to guarantee that it has an inverse.

From start to finish: take a text, convert the letters to numbers, (assume it has an even number of letters), break the sequence of numbers into blocks of 2 numbers each, and apply the matrix to each pair to get an encoded pair.

Notice that this can be extended to 3×3 matrices or more generally $k \times k$ matrices.

If a 2 by 2 matrix is used then a Freq. Analysis based on pairs may still be possible. Same for 3 by 3. What about for 10 by 10 code?

PRO: It would seem that Eve has a hard time using Freq Anal if the matrix is large enough. Brute force also looks hard (see later in this section when we discuss cracking the code.

CON: Alice and Bob still have to meet to establish the key. Is that avoidable?

Definition 7.1 A sequence of k letters is called a *k-gram*. Freq. anal. usually uses 1-grams and 2-grams. For a $k \times k$ matrix we would need to use k-grams.

Ways to Crack a $k \times k$ Matrix Code

1. Try to use Freq. Analysis with k-grams.
2. There are roughly 26^{k^2} possible matrices. You can try them all (and use the summation technique above to check).

If $k = 20$ then can these attacks work? No. But can other attacks work? There are two answers to this question:

1. We are assuming that Eve has just the ciphertext and also knows that its a matrix code with a 20×20 matrix. In reality Eve will also likely know some prior messages and what they decoded to. She can use this and affine algebra to get get the matrix, or at least narrow down options for the matrix.

2. Even if Eve just has the ciphertex and the dimension of the matrix then there are still ways to do crack the code that take less than 26^{k^2} steps. See the notes on this on the course website. This is actually research in progress!

8 Vigenere Cipher

Here is another way to defeat Freq. Analysis is the *Vigenere Cipher*. Here is an example:

- Every letter that is in a place $\equiv 0 \bmod 5$ is coded by a shift-4.
- Every letter that is in a place $\equiv 1 \bmod 5$ is coded by a shift-15.
- Every letter that is in a place $\equiv 2 \bmod 5$ is coded by a shift-7.
- Every letter that is in a place $\equiv 3 \bmod 5$ is coded by a shift-13.
- Every letter that is in a place $\equiv 3 \bmod 5$ is coded by a shift-1.

Alice could communicate the key to Bob as the sequence (3, 14, 6, 12, 0). Or she could just say DOGMA since D is the third letter of the alphabet Recall that A is 0) O is the 14th, etc. We call DOGMA the key, and 5 the keylength. They could be any word (or sequence of numbers in $\{0, \ldots, 25\}$) and any length.

How to crack it? We give two methods. Both require long texts. Both involve finding the key length and then doing a Freq. Analysis on the appropriate subtexts (e.g., in the above example you would do a Freq. Analysis on every 5th letter).
Kasiski Examination: We first find the key length. Imagine that in the text you see ABDUQ several times. It is likely that the DISTANCE between them is the key length of a multiple of it. So find the differences between the repeated text and then find a common factor for all of them. This gives you a SMALL set of key lengths to look at.

Assume you have the key length (or a small set of candiates for it). IMPORTANT POINT: The freqs of letters in English are about the same if you look at (say) every 5th letter. We now use this:

1. Given T the text and m the key length.

2. For $0 \leq i \leq m-1$

   (a) Look at the letters in the positions $\equiv i \pmod{m}$.

   (b) Treat these letters as if they came from *the same* shift cipher.

   (c) Use the techniques discussed earlier to find the shift s_i. (If they do not work then m is the wrong key length and try another candidate for key length.)

   (d) Shift all of the letters posistions $\equiv i \pmod{m}$ by $26 - s_i$.

We defer PROS and CONS to the next section.

9 Vigenere Plus Cipher

In the Vig cipher we used a SHIFT on every Lth letter. We could instead use a Affine, Quadratic, Polynomial, Matrix on every Lth letter.

To my knowledge no such code was ever used. This is just an accident of history. Let Y be a year (I do not know what year it is). Before year Y, using a Vig Plus Cipher would have been too cumbersome for Alice and Bob. After year Y, there were better techniques.

There is another difficulty in using (say) affine. For Shift we had a very short key: just a word and then use the letters positions. Any word works. For affine we would need two words. But the first one can only use letters whose corresponding numbers are rel prime to 26.

PROS: Very Short Key give you very large complexity. Even though it's crackable, we need to make Eve really work at it. Some texts say that it was uncracked for 300 years, though Jon Katz doubts this. And note that if it was cracked then the crackers wouldn't want to brag about it.

CONS: Alice and Bob have to meet. Is this avoidable?

10 An Uncrackable Code: the One-Time Pad

Definition 10.1 If a and b are bits (0 or 1) then $\oplus$ (also written XOR and called "exclusive or") is defined as follows:

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

The following facts are easy to verify.

Fact 10.2 *Let a, b, c be bits.*

1. $(a \oplus b) \oplus c = a \oplus (b \oplus c)$.
2. *For all bits a,* $a \oplus a = 0$.
3. $a \oplus b \oplus b = a \oplus (b \oplus b) = a \oplus 0 = a$.

We now describe the one-time pad.

1. Alice and Bob have to meet. and agree on a randomly generated sequence of bits – a VERY long sequence. Say it's

$$r_1 r_2 \cdots r_N.$$

2. If (later) Alice wants to send

$$a_1 a_2 a_3 \cdots a_m$$

she sends

$$(r_1 \oplus a_1)(r_2 \oplus a_2) \cdots (r_m \oplus a_m).$$

When Bob gets this string, which he sees as

$$s_1 \cdots s_m$$

he can decode it by taking

$$\begin{aligned}(r_1 \oplus s_1)(r_2 \oplus s_2) \cdots (r_m \oplus s_m) &= (r_1 \oplus (r_1 \oplus a_1))(r_2 \oplus (r_2 \oplus a_2)) \cdots (r_m \oplus (r_m \oplus a_m)) \\ &= ((r_1 \oplus r_1) \oplus a_1)((r_2 \oplus r_2) \oplus a_2)) \cdots ((r_m \oplus r_m) \oplus a_m)) \\ &= a_1 a_2 \cdots a_m\end{aligned}$$

3. If either Alice or Bob wants to send another message they will start with r_{m+1}.

PROS: This is impossible to crack! Since the original key was random, if Eve sees the message

$$s_1 s_2 \cdots s_m$$

it will look random to her.

CONS: N is LARGE! They have to meet and exchange A LOT of information. In fact, if they plan to later communicate N bits they need to have a key of length N.

PROBLEM: Can Alice and Bob use a shorter key?

ANSWER: They can if they use a recurrence to generate the sequence, like

$$r_n = r_{n-1} + r_{n-2} \pmod 2$$

but then the sequence is no longer random. People have looked at getting random-looking sequences.

PROBLEM: Can Alice and Bob agree on a secret key (e.g., $r_1 r_2 \cdots r_N$) without having to meet?

11 Alice and Bob Do Not Have To Meet

The following problem plagues all of the systems we have considered: Alice and Bob must meet in secret to establish a key.

Is there a way around this? Is there a way for Alice and Bob to NEVER meet, and yet establish a secret key? That is, can they, by talking *in public* establish a shared secret key?

The answer will be yes, assuming that whoever is listening in has some limits on what they can compute.

11.1 Needed Math

We'll use multiplication modulo p in the set $Z_p = \{1, 2, \ldots, p-1\}$, where p is a prime number. It will be useful to find an element $g \in Z_p$, called a "generator", for which the sequence $g^0, g^1, g^2, \ldots, g^{p-2}$, taken modulo p, contains all of the elements of Z_p.

Let's look at $p = 11$. Notice that

$$\begin{aligned}
2^0 &\equiv 1 \bmod 11 \\
2^1 &\equiv 2 \bmod 11 \\
2^2 &\equiv 4 \bmod 11 \\
2^3 &\equiv 8 \bmod 11 \\
2^4 &\equiv 5 \bmod 11 \\
2^5 &\equiv 10 \bmod 11 \\
2^6 &\equiv 9 \bmod 11 \\
2^7 &\equiv 7 \bmod 11 \\
2^8 &\equiv 3 \bmod 11 \\
2^9 &\equiv 6 \bmod 11
\end{aligned}$$

These calculations are not hard if you use that $2^n \equiv 2 \times 2^{n-1} \bmod 11$. Notice that $\{2^0 \bmod 11, 2^1 \bmod 11, \ldots, 2^9 \bmod 11\} = \{1, 2, \ldots, 10\}$.

Do all elements of Z_{11} generate the entire set? No:

$$\begin{aligned}
5^0 &\equiv 1 \bmod 11 \\
5^1 &\equiv 5 \bmod 11 \\
5^2 &\equiv 3 \bmod 11 \\
5^3 &\equiv 4 \bmod 11 \\
5^4 &\equiv 9 \bmod 11 \\
5^5 &\equiv 1 \bmod 11 \\
5^6 &\equiv 5 \bmod 11 \\
5^7 &\equiv 3 \bmod 11 \\
5^8 &\equiv 4 \bmod 11 \\
5^9 &\equiv 9 \bmod 11
\end{aligned}$$

Notice that $\{5^0 \bmod 11, 5^1 \bmod 11, \ldots, 5^9 \bmod 11\} = \{1, 3, 4, 5, 9\}$. This is NOT all of Z_{11}.

Convention 11.1 We will be using a prime p. We will assume that p is LARGE but that $\log p$ is not too large. Hence if Eve needs a computation of p steps to crack a code we will consider it a good code. Even if Eve needs a computation of $\sqrt{p}$ steps (or p^ϵ steps where $\epsilon > 0$) this is a long time and we will consider it a good code. Also, if Alice and Bob have to do operations that take $\log p$ steps, that's okay, they can do that. Even if they have to take $(\log p)^2$ (or some larger polynomial in $\log p$) thats okay, they can do that.

Convention 11.2 For the rest of this document when we say "roughly p" we will mean p^ϵ for some ϵ, $\epsilon > 0$. When we say "roughly $\log p$" we will mean $(\log p)^a$ for some $a \in N$.

Theorem 11.3 *For every prime p there is a g such that $\{g^0 \bmod p, g^1 \bmod p, \ldots, g^{p-2} \bmod p\} = Z_p = \{1, \ldots, p-1\}$. There is an algorithm which will, given p, find such a generator g in roughly $\log p$ steps.*

We have already seen that $+, -, \times$, and (if p is prime) division can be done modulo p. We now have a way to do LOGARITHMS modulo p.

Definition 11.4 Let p be a prime and g be a generator of Z_p. Let $x \in Z_p$. The *Discrete Logarithm of x with base g* is the $y \in \{0, \ldots, p-2\}$ such that $g^y \equiv x \bmod p$. We denote this $DL_g(x)$.

Example 11.5 We rewrite the table above for $p = 11$ and add to it. The Discrete Logarithm lines follow from the prior line. We assume $g = 2$ and denote DL_2 by just

DL.

$$\begin{aligned} 2^0 &\equiv 1 \bmod 11 \\ DL(1) &= 0 \end{aligned}$$

$$\begin{aligned} 2^1 &\equiv 2 \bmod 11 \\ DL(2) &= 1 \end{aligned}$$

$$\begin{aligned} 2^2 &\equiv 4 \bmod 11 \\ DL(4) &= 2 \end{aligned}$$

$$\begin{aligned} 2^3 &\equiv 8 \bmod 11 \\ DL(8) &= 3 \end{aligned}$$

$$\begin{aligned} 2^4 &\equiv 5 \bmod 11 \\ DL(5) &= 4 \end{aligned}$$

$$\begin{aligned} 2^5 &\equiv 10 \bmod 11 \\ DL(10) &= 5 \end{aligned}$$

$$\begin{aligned} 2^6 &\equiv 9 \bmod 11 \\ DL(9) &= 6 \end{aligned}$$

$$\begin{aligned} 2^7 &\equiv 7 \bmod 11 \\ DL(7) &= 7 \end{aligned}$$

$$\begin{aligned} 2^8 &\equiv 3 \bmod 11 \\ DL(3) &= 8 \end{aligned}$$

$$\begin{aligned} 2^9 &\equiv 6 \bmod 11 \\ DL(6) &= 9 \end{aligned}$$

COMMON BELIEF: It is believed that the problem of computing the discrete logarithm *requires* roughly p steps. This is a long time, so we assume Eve cannot do this.

THIS IS THE REPEATED SQUARING METHOD:

Lemma 11.6 *Given p, $a \in \{0, 1, \ldots, p-1\}$, and m, determining $a^m \bmod p$ takes roughly $\log m$ steps. (This is by repeated squaring.)*

Proof: We do an example which should show the general idea.

We want to compute $3^{278} \pmod{17}$.

First write 278 in binary. 100010110

So we really want $3^{2^8}3^{2^4}3^{2^2}3^{2^1} \pmod{17}$

(All computation in this example is now mod 17).

We will compute more than we need: We will compute 3^{2^i} for $0 \leq i \leq 8$.

$3^{2^0} \equiv 3^1 \equiv 3$

$3^{2^1} \equiv (3^{2^0})^2 \equiv 3^2 \equiv 9$ (NOTE- we knew 3^{2^0} from the prior line)

$3^{2^2} \equiv (3^{2^1})^2 \equiv 9^2 \equiv 81 \equiv 13$ (NOTE- we knew 3^{2^1} from the prior line)

$3^{2^3} \equiv (3^{2^2})^2 \equiv 13^2 \equiv (-4)^2 \equiv 16$ (NOTE- we knew 3^{2^2} from the prior line)

(NOTE- This is not really part of the proof, but it made life slighly easier to write 13 as -4. I may do this again without commenting on it.)

$3^{2^4} \equiv (3^{2^3})^2 \equiv 16^2 \equiv (-1)^2 \equiv 1$ (NOTE- we knew 3^{2^3} from the prior line)

$3^{2^5} \equiv (3^{2^4})^2 \equiv 1^2 \equiv 1$ (NOTE- we knew 3^{2^4} from the prior line)

$3^{2^6} \equiv (3^{2^5})^2 \equiv 1^2 \equiv 1$ (NOTE- we knew 3^{2^5} from the prior line)

$3^{2^7} \equiv (3^{2^6})^2 \equiv 1^2 \equiv 1$ (NOTE- we knew 3^{2^6} from the prior line)

$3^{2^8} \equiv (3^{2^7})^2 \equiv 1^2 \equiv 1$ (NOTE- we knew 3^{2^7} from the prior line)

Hence

$$3^{2^8}3^{2^4}3^{2^2}3^{2^1} \equiv 1 \times 1 \times 13 \times 9 \equiv -4 \times 9 \equiv -36 \equiv 34 - 36 \equiv -2 \equiv 15$$

More generally: to compute $a^b \pmod{c}$ you compute a^{2^i} as above, using a^{2^i} to computer $a^{2^{i+1}}$. Once you have all of the powers of 2 you can then write b in binary and use the powers of two that you need. ∎

Lemma 11.7

1. *Given p, testing if p is prime can be done in roughly $\log p$ steps.*
2. *Given n, finding a prime p such that $n \leq p \leq 2n$. can be done in roughly $\log^2 n$ steps.*
3. *Given n, finding a prime p such that $n \leq p \leq 2n$ and a generator g for Z_p can be done in roughly $\log^3 n$ steps.*

Proof:

1) This is known and we skip it.

2) It is known (the prime number theorem) that between n and $2n$ there are roughly $\frac{n}{\log n}$ primes. Hence if we pick numbers between n and $2n$ at random we will almost surely encounter a prime within the first $O(\log n)$ picks. This leads to the following algorithm:

1. Input n
2. Repeat until you find a prime:
   (a) Pick a number $p \in [n, 2n]$ at random.

(b) Test if it is a prime using part 1. If so then output p and STOP.

Each iteration takes roughly $\log n$ steps, and the number of iterations will be at most roughly $\log n$. Hence the algorithm takes $\log^2 n$ steps.

We give some notes on how to speed this up, though we ignore plus or minus 1's.
NOTE: We can speed this up by only guessing odd numbers. Do that by picking $q \in [n/2, n]$ at random and guessing $p = 2q + 1$. Instead of having to guess from n numbers, we only have to guess from $n - n/2 = 0.5n$ numbers!
NOTE: We can speed this up by even more by only guessing numbers that are not divisible by 2 or 3, so numbers that are $\equiv 1, 5 \pmod 6$. Do that by picking $q \in [n/6, /n/3]$ at random, picking $i \in \{1, 5\}$ at random, and guessing $6q + i$. Instead of having to guess from n numbers, we only have to guess from $n - n/2 - n/3 + n/6 = 0.33n$ numbers!
NOTE: We can speed this up by even more by only guessing numbers that are not divisible by 2 or 3 or 5, so numbers that are $\equiv 1, 7, 11, 13, 17, 19, 23, 29 \pmod{30}$. Do that by picking $q \in [n/30, /n/60]$ at random, picking $i \in \{1, 7, 11, 13, 17, 19, 23, 29\}$ at random, and guessing $30q + i$. Since there are 8 i's to pick the number of numbers we are looking at is $8n/30 = 0.26n$

We could go on. If n was really big and we were the NSA we would go on. However, past some point the gains are not worth the effort. We stop here.

These improvements help the constant but not the main term of $\log n$.

3) We first give an algorithm that is too slow.

1. Input(n)
2. Find a prime $p \in [n, 2n]$ by using the algorithm from part 2.
3. For $g = 2, 3, 4, \ldots$ until you get a generator do the following:
   - If $g^2, g^3, \ldots, g^{p-1} = \{1, 2, \ldots, p-1\}$ (in a diff order) then output (p, g) and STOP

The problem with this is that looking at $g^2, g^3, \ldots, g^{p-1}$ is p calculations which is too many.

BUT notice the following: it is known that if g is not a generator then there exists a number $i < p - 1$ such that i divides $p - 1$ and $g^i = 1$. This inspires this next algorithm that does not work fast enough.

1. Input(n)
2. Find a prime $p \in [n, 2n]$ by using the algorithm from part 2.
3. Factor $p - 1$. Let F be the set of factors (not including $p - 1$). F will be small- roughly $\log n$.

4. For $g = 2, 3, 4, \ldots$ until you get a generator do the following:
   - Find $A = \{g^a : a \in F\}$.
   - If $1 \notin A$ then output (p, g) and STOP.

GOOD NEWS: the calculation of the set A is fast.
BAD NEWS: We need to factor $p-1$. That could be hard! In fact, many protocols in crypto are based on factoring being hard.
IDEA: Choose your prime p so that $p-1$ is easy to factor. We will choose p to be a *safe prime* which means that $p-1 = 2q$ where q is a prime. It is know that the number of safe primes in $[n, 2n]$ is roughly $\frac{n}{\log^2 n}$.

1. Input(n)
2. Repeat until you find a prime:
   (a) Pick a number $p \in [n, 2n]$ at random.
   (b) Test both p and $q = \frac{p-1}{2}$ for primality part 1. If both are primes then get out of this loop and goto the next step. NOTE: We know that $p - 1 = 2q$.
3. Let $F = \{2, q\}$. Note that this is the set of ALL factors of $p-1$.
4. For $g = 2, 3, 4, \ldots$ until you get a generator do the following:
   - Find $A = \{g^a : a \in F\}$.
   - If $1 \notin A$ then output (p, g) and STOP.

Since there are roughly $\frac{n}{\log^2 n}$ safe primes there will be roughly $\log^2 n$ iterations of the first loop. Each iteration takes roughl $\log n$ steps. Hence the first phase where you find p takes $\log^3 n$ steps.

we hope there are a lot of generators so that the second loop ends quickly. One can show the the number of generators is also the number of numbers $\leq p-1$ that are rel prime to $p-1$. We are in luck! Since $p - 1 = 2q$ and q is prime the numbers relatively prime to $p-1$ are

$$\{1, 3, 5, 7, 9, \ldots, p-2\} - \{q\}$$

Which is roughly half of the numbers. So the second loop goes a constant number of iterations. Each iteration takes 2 calculations which take $\log n$ each. So the second loop takes roughly $\log n$ steps.

Hence the entire process takes $O(\log^3)$ steps.

∎

11.2 Diffie Helman Key Exchange

We can USE this mathematics to have Alice and Bob exchange information in public and in the end they have a shared secret key.

1. Alice generates a large prime p and a generator g (this takes roughly $\log p$ steps) and sends it to Bob over an open channel. So now Alice and Bob know p, g but so does Eve.

2. Alice generates a random $a \in \{0, \ldots, p-2\}$. Bob generates a random $b \in \{0, \ldots, p-2\}$. They keep these numbers private. Note that even Alice does not know b, and even Bob does not know a.

3. Alice computes $g^a \bmod p$. Bob computes $g^b \bmod p$. Both use repeated squaring so it takes roughly $\log p$.

4. Alice sends Bob $g^a \bmod p$ over an open channel. Notice that Eve will NOT be able to compute a if computing DL_g is hard (which is the common belief). Even Bob won't know what a is.

5. Bob sends Alice $g^b \bmod p$. Notice that Eve will NOT be able to compute b if computing DL_g is hard. Even Alice won't know what b is.

6. RECAP: Alice now has a and g^b. SHE DOES NOT HAVE b. Bob has b and g^a. HE DOES NOT HAVE a. Eve has g^a and b^b. SHE DOES NOT HAVE a OR b.

7. Alice computes $(g^b)^a \bmod p = g^{ab} \bmod p$. Bob computes $(g^a)^b \bmod p = g^{ab} \bmod p$. They both use repeated squaring so this is fast.

8. SO at the end of the protocol they BOTH know $g^{ab} \bmod p$. This is their shared secret key. Eve likely does NOT know g^{ab} since she only gets to see g^a and g^b.

This scheme LOOKS good but we must be very careful about what is known about it.

1. Alice and Bob can execute the scheme quickly.

2. If Eve can compute DL_g quickly then she can crack the code.

3. There MIGHT BE other ways for Eve to crack the code. That is, being able to compute DL_g quickly is sufficient to crack this scheme, but might not be neccesary.

4. This scheme can be used for Alice and Bob to establish a secret key without meeting. This can then be used in other schemes such as the one-time pad.

5. Reality: This scheme is used in the real world for secret key exchange. The RSA algorithm and the ElGamal methods are also used for Public Key Cryptography (which is similar).

6. Reality: Quantum Key distribution is also a way for Alice and Bob to not have to meet.