Cryptography

KNU Math 406

Classnotes

Mark Siggers

v. May 24, 2018

These notes are for the first few classes of a course taught from Hoffstein Pipher and Silverman's 'An Introduction to Mathematical Cryptography'. which is herein refereed to as 'the text'. They draw largely from the text but reorder it a bit: bringing Sections 1.7 and 2.5 forward. I number the sections to try to reflect the this.

This is an upper level undergraduate first class in Cryptography.

1 An Introduction to Cryptography

This class is about *Cryptography*. *Coding Theory* is the study of *codes*. The main types of codes are:

i). *Secret codes* - used to transfer information so that only the intended recipient can read it.

ii). *Compression codes* - used to transfer information efficiently.

iii). *Error correcting codes* - used to correctly transfer information that might get corrupted in transfer.

'Cryptography' is the study of secret codes, and this is what we will study. Though 'Coding theory' can refer to everything, it more commonly refers to the study of the last two types of codes.

1.1 Intro to the Intro: Simple Substitution Cipher

Example 1.1. Say Bob wanted to send a message to Alice, but didn't want Eve to be able to read it.

TOMMORROW AT LUNCH

He could replace every letter with the letter to the left of it on the keyboard, and instead send

YPZPTTPE SY AIMVJ

Eve can't read his message, but Alice knows she has to replace every letter with the letter to the left of it on the keyboard, and can convert it back to the original message.

What Alice and Bob have used is called a *simple substitution cipher*. It is an example of a *cryptosystem*. The elements of a cryptosystem are, informally:

- The message 'TOMMORROW AT LUNCH' is the *plaintext*.
- The message 'YPZPTTPE SY AIMVJ' is the *ciphertext*.
- The algorithm for converting between ciphertext and the plaintext is called the *cipher*, it depends on a *key*.
- The process of converting the plaintext to the ciphertext is called *encryption*.
- The process of converting the ciphertext to the plaintext using the key is called *decryption*.
- The process of converting the ciphertext to the plaintext without the key is called *cryptanalysis*, or *cracking* the cipher.

Simple substitution ciphers are an obvious cryptosystem and have historically been very useful. But they can be cracked quite easily. Especially for long messages.

1.1.1 Cracking a Simple Substitution Cipher

The most obvious approach is to try every possible substition of the letters and see which gives some proper English words. But 26! is a pretty big number and this would be tricky even with computers. This is the approach, but one has to be a bit more clever. Just a bit though, this approach is still quite obvious. It appears in Poe's 'The Gold Bug'.

The letter 'e' occurs most frequently in the English language, accounting for about .13 of all letters. Then comes 't' at about .10 and then 'a' at about .08. Given a long enough ciphertext, the basic idea is to count the occurences of each symbol, replace the most frequent with an 'e', the next most frequent with a 't', etc.

Sure this isn't going to work out exactly all the time. Indeed it usually won't. But the most common letters are 'e, t, a, o, n'. Try putting these randomly into the 8 or so most frequent ciphertext symbols, you have a much more managable number of combinations. You can check these for agreement with words in the english language: there should be no two letter words 'ta' or 'ea'. Much better.

One can also use the frequency of two letter combinations. The most common two letter combinations, or digraphs, in English are 'th' 'he' and 'in'. Choosing 't' and 'e' in the first step, if there is some ciphertext letter X such that tX and Xe are frequent, you can be pretty sure that this is an h.

And if you have a three letter work 'PYQ' occuring several times you can be pretty sure this 'the' or 'and'.

With a bit of math, but more contextual reasoning, it is not hard to solve a simple substition cipher. But it is hard enough to be fun. That is why there

are Word Jumbles. http://www.wordjumble.com/ Give them to your Dad and tell him they fight off Alzheimier's.

Problem 1.1. Decrypt the following message:

> GAYU V XVUE WHKYQX VU CVWYK SX CTSNRQY WSCAYT WMTH ZSWYK CS WY KLYMIVUF GSTEK SX GVKESW QYC VC RY

Read Section 1.1.1 of the text if necessary for hints on how to do it.

1.1.2 Private vs. Public key Cryptosystems

In the simple substitution cipher, it is required that Alice and Bob both know the key. Such cyptosystems are called *private key* cryptosystems, or *symmetric* cryptosystems.

Contrastingly, in a *public key* cryptosystem, or *asymmetric* cryptosystem, there are two keys, one for encryption, and one for decryption. In such a cryptosystem, Alice keeps the decryption key secret, but makes the encryption key public. Under this scheme, anyone, including Bob, can send Alice a message, which only Alice can read.

There are advantage to both private key and public key systems.

As we saw, private key cryptosystems can be cracked, but this is not really one of their shortcomings. There are many different private key cryptosystems, and it is not hard to come up with one that is impossible to crack without the key.

Problem 1.2. Come up with a private key cryptosystem that is impossible to crypoanalyse, and explain why it is impossible.

But this suggests one of the shortcomings. What if Bob loses the key? Wiki up the story of the Enigma code. Those Germans were using a private key cryptosystem called the Enigma - a fancy sort of shift cipher with a progressive shift. They were sneaking around in their submarines wreaking German havoc. Nobody could crack their code. Not even the guy from Rubicon[1]. But then the Americans or the British, or probably the Canadians, caught one of their submarines and got the key. End of story, Germans.

Actually, it turns out it was the Polish that got the Enigma Machine from the Germans, and that that wasn't the end of the story. The Enigma Machine wasn't the key to the German code, it gave the British the cipher. The key changed everyday, and even with the cipher, the English had to use a contextual attack to crack it. This took too long to be useful, until our hero, Alan Turing

[1] What year were the first version of these notes written?

built the first computer, specifically to crack the cipher. Everyone should watch the beautiful movie, 'The Imitation Game'.

If you want a code that many people can use, and any one person loses the private key, the code is compromised.

In a public key cryptosystem, anyone can have the encryption key. Alice doesn't care if one of the Bobs gives it[2] up to Eve, she even makes it public, Eve still can't decode other peoples ciphertexts.

Another shortcoming of the private key cryptosystem is that Alice and Bob must first meet to share the key, (well, not actually, as we will see) and this is not always practical.

While there are uncrackable private key cryptosystems, public key cryptosystems, by necessity, can be cracked. What makes one good, is that it is difficult to crack. In this course, except in the first few classes, we will look exclusively at public key cryptosystems, and the mathematics behind them.

1.7 A more rigourous introduction

This is mostly Section 1.7 of the text. We make our definitions more rigourous, and lay out the rules of the game- those assumptions that we will be making to allow us to get to the math of the subject.

1.7.1 Encryption Schemes or Cryptosystems

An *encryption scheme* or *cryptosystem* is a tuple $(\mathcal{K}, \mathcal{M}, \mathcal{C}, e, d)$ where $\mathcal{M}$ is a space of *plaintexts* or messages, $\mathcal{C}$ is a space of *ciphertexts*, $\mathcal{K}$ is a space of *keys*, e is function

$$e : \mathcal{K} \times \mathcal{M} \to \mathcal{C} : (k, m) \mapsto e_k(m),$$

called *encryption*, and d is a function

$$d : \mathcal{K} \times \mathcal{C} \to \mathcal{M} :\mapsto d_k(c)$$

called *decryption*, such that $d_k(e_k(m)) = m$ for any key $k \in \mathcal{K}$ and message $m \in \mathcal{M}$.

Once k is chosen, we often refer to the map $e_k : m \mapsto e_k(m)$ as the encryption key, and $d_k : c \mapsto d_k(c)$ as the decryption key. In private key cryptosystems, it is usually easy to determine d_k from e_k. The important feature of public key cryptosystems is that it is hard.

Example 1.2. (Modular shift cryptosystem) A more mathematical way to implement the substitution cipher we saw earlier is to use a a modular shift.

[2]The encryption key!

Encoding the i^{th} letter of the alphabet into the number $i-1$, (so $a=0$ and $b=1$ and, ..., $z=25$), encrypting by adding $k=9$ mod 26, and then decoding back to letters, we get the following substitution cipher:

TOMORROW AT LUNCH
↓ (encoding)
19 14 12 14 17 17 14 22 00 19 11 20 13 02 07
↓ (encryption)
03 23 21 23 01 01 23 06 09 03 20 04 22 11 16
↓ (decoding)
DXVXBBXG JD UEWLQ

Its the bit in the middle, the encryption, where the math is involved, the encoding and decoding is just overhead, and is not part of our course. We think of $\mathcal{M}$ and $\mathcal{C}$ in this case as $\mathbb{Z}_{26}$, the integers modulo 26. Because of this, the only real keys k are also integers mod 26. So $\mathcal{K}=\mathbb{Z}_{26}$. Our encryption key is $e_k(m)=m+k$ modulo 26 and our decryption key is $d_k(c)=c-k$ modulo 26.

Properties that we want of a cryptosystem are

P1) $e_k(m)$ is easy to compute for $k\in\mathcal{K}$ and $m\in\mathcal{M}$, and $d_k(c)$ is easy to compute given k and c.

P2) Given many $c_1,\ldots,c_n\in\mathcal{C}$, it is difficult to compute any of the $d_k(c_i)$ without having k.

P3) Given several pairs $(c_1,d_k(c_1)),\ldots,(c_n,d_k(c_n))$, it is difficult to compute $d_k(c)$ for some $c\notin\{c_1,\ldots,c_n\}$.

Property P1 is so that the cryptosystem is usable, properties P2 and P3 essentially say that cryptoanalysis is hard.

Property P2 and blocksize

In the modular shift cryptosystem even the weaker property P2 fails. We saw this earlier when we solved our Word Jumble. The problem here is that $\mathcal{M}$ is too small. Each letter was encrypted individually, so it is not too hard to match the letters up with what they encrypt to. Once this is done, we can read every ciphertext. To overcome this we string several letters together to make an element of $\mathcal{M}$. Using 30 for spaces and to pad the strings of to the right length, our 'TOMORROW AT LUNCH' example encodes to a couple of strings of length 20, say:

19 14 12 14 17 17 14 22 00 19 11 20 13 02 07

↓

19141214171714223000 19 30112013020730303030

Actually, it is computers doing it, so we usually encode it in binary strings:

00110111001100011101000101000101000101010

10001011101011011111010001000100101011 1

Then we pick some big string k and encrypt with it. The length B of the strings in $\mathcal{M}$ is called the *blocksize* of the cipher.

Usually $\mathcal{M}, \mathcal{C}$, and $\mathcal{K}$ are all regular length binary strings. We use B_m, B_c and B_k to refer to their respective blocklengths. It is generally accepted that with present computing power, a cipher is safe against an exhaustive search attack if the blocklength is at least 80.

On Property P3

With big block size, we usually have property P2, but the simple shift cryptosystem still fails property P3. If Eve knows we are using a shift cipher (and this is an assumption we usually make) then as soon as she has one pair (m, c) where $c = e_k(m) = m + k$ then she can compute $c - m = k$ and she has Alice's key k. So we need a better cryptosystem.

Example 1.3. (Modular Multiplication Cryptosystem) Let $n \approx 2^{160}$ be a large integer of 160 binary digits, and let let $\mathcal{M} = \mathcal{C} = \mathcal{K}$ be the set of integers modulo n. We will study these a bit later, but for now, if you are unfamiliar with them, just pretend that you are not.

Let k be an invertible element mod n and let encryption be

$$e_k(m) = k \cdot m.$$

Then

$$d_k(m) = k^{-1} \cdot m$$

is decryption.

Is this any better than the modular shift cryptosystem? If Eve has m and $c = km$ then she can divide c/m and get k, so for any ciphertext c' she can then divide c'/k to get $d_k(c')$. Right?

Well. We aren't playing with the reals. We are playing with integers modulo n. So dividing isn't as easy as all that. But you are right, it is not so hard. We will see that it takes only about $3 \log_2(n) = 480$ 'calculations' to find k^{-1} when we are playing modulo some n with about 160 binary digits. A computer can do this in seconds, so this is easy.

So a Modular Multiplication Cryptosystem is not such a good public cryptosystem either. What is? Well, what's harder than dividing? Yep. Taking roots.

Example 1.4. (RSA Cryptosystem) Again let n be an integer of about 160 binary digits, and let $\mathcal{M} = \mathcal{C} = \mathcal{K}$ be the set of integers modulo n.

We encrypt a message m by $e_k(m) = m^k$ modulo n. Decryption is then taking a k^{th} root. It turns out that this is pretty hard modulo a properly chosen n. Unless that is, you know the secret key.

We will describe RSA properly latter, and several other cryptosystems based on hard problems like taking modular roots or logarithms. Before we do so we take a brief detour into Algoritmic Complexity.

2.5 Algorithm Complexity

This sections draws some from Section 2.5 of the text, but adds other things.

The most important thing about an asymmetric encryption scheme is that it is easy to encrypt, easy to decrypt with the decryption key, but hard to decrypt without the key. We have to make the words 'easy' and 'hard' more clear. Our goal is to understand statements such as 'Multiplication is easy, but taking roots is hard.'

For this, we talk about the running time of algorithms and computational complexity.

2.5.1 Order Notation

Where f and g are positive functions of x, we say that *'f is of order at most g'*, or *'f is big oh of g'* and write $f = O(g)$ if there exist constants c and N such that

$$(x > N) \Rightarrow (f(x) < c \cdot g(x)). \tag{1}$$

It is easy to show that if

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = c,$$

then (1) holds for some N.

So, for example, $x^2 = O(x^2)$ and $x^2 = O(x^3)$, but $x^3 \neq O(x^2)$. Those are sillyish examples. One of the point of writing something like this is to simplify a function, to get rid of unimportant terms. The typical example is one such as

$$1/2x^2 - 5x + 7 = O(x^2).$$

Another use of the notation is to compare the eventual growth of functions. Exponentials grow faster than polynomials, so

$$f(x) = 34x^{100} + 5x^2 + 100000 = O(e^x) \text{ and } e^x \neq O(f(x)).$$

We say that exponentions have greater order than polynomials. There is a heirarchy of orders, those orders that we are most interested in are

$$\text{logorithms} < \text{roots} < \text{polynomial} < \text{exponentials}$$

Problem 2.3. Show that $\sqrt{x} = O(x)$, and that $x^2 = O(2^x)$. (Hint: l'Hopital might be useful).

Problem 2.4. Show that if $f(x) = O(x^2)$ and $g(x) = O(\log(x))$, then $f(x) + g(x) = O(x^2)$, and $f(x) \cdot g(x) \neq O(x^2)$.

2.5.2 Algorithms and Running Time

All of our encryption schemes, if done properly, will require computers to encrypt and decrypt. (We of course will do examples with absurdly small encryption and decryption spaces, but we will always understand that our words are really hundreds of digits long.)

We define an *algorithm* a little informally. It performs a task by taking an *input*, performing a finite series of steps, and returns an *output*.

Such a task is addition, or ADD. It has an input of two integers, A and B and must output $A + B$.

There are many algorithms for ADD. The Lucy algorithm is as follows, in Sage code.

```
def ADD(A,B):
  while B > 0:
    A = A + 1
    B = B - 1
  return(A)
```

We would like to count how long it takes to compute ADD(A, B). This depends on the size of A and B, which we measure in their number k of binary digits, or *bits*. We measure the length of the algorithm in binary operations, or *bops*.

Note

I'm note sure if 'bops' is standard terminology.

Adding 1 to a number A of k binary digits takes one bop if the last digit is 0, but more otherwise. In the worst case it takes $O(k)$ bops (it seems like k bops, but carrying makes it more like $2k$). Doing this B times, takes $O(k \cdot B) = O(k \cdot 2^k)$ bops, so in the worst case the algorithm takes $O(k \cdot 2^k)$ bops.

The *running time* of an algorithm A is a function $f_A : \mathbb{N} \to \mathbb{N}$, such that for $k \in \mathbb{N}$, $f_A(k)$ is the maximum number of bops that the algorithm A requires to return the anwser for an input I of size k. For ADD it is standard to take k as $\max(k_A, k_B)$ where these are the number of bits in A and B respectivley. The running time for Lucy's algorithm is $O(k \cdot 2^k)$.

There are other algorithms for ADD. The computer sees the sum

$$4 + 23$$

in binary representation

$$00100 + 10111$$

and does elementary school addition.

This has running time $f(k) = 2k$: for each of the five digits, it adds two, or three if there is some carrying, binary digits. This running time of $O(k)$ is better than the running time of $O(k \cdot 2^k)$ for Lucy's algorithm.

Problem 2.5. Write a sage/pseudo-code function to do the elementary school algorithm for addition.

The *computational complexity* $\mathrm{Comp}(A)$ of a problem A is the running time of the best possible algorithm for A.

Sometimes it is hard to know exactly what the 'best possible' algorithm is. But using big-O notation allows us to gloss over this. We do not know that Comp(ADD) is $2k$, but we do know that it is $O(k)$.

MULT is the task of multiplying two numbers together. Lucy's algorithm for $\mathrm{MULT}(A, B)$ is as follows, in Sage code.

```
def SLOWMULT(A,B):
  T = 0
  for i in range(B):
    T = T + A
  return(T)
```

Note

Technically, the 'for i in range' also calls another ADD per loop, but such things are swallowed up in the big-O notation.

This calls ADD at most 2^k times with input of at most $2k$ bits, so taking time $O(2k) = O(k)$ each. Thus $\mathrm{Comp(MULT)} = O(2^k \cdot \mathrm{Comp(ADD)}) = O(2^k \cdot k)$.

Problem 2.6. In computing the running time for the the elementary addition algorithm, it doesn't hurt to assume A and B both have the same number of bits: the *worst case* running time is the same whether we pad the smaller number with 0s or not. In the SLOWMULT algorithm, this is not true. Find the running time for $\mathrm{SLOWMULT}(A, B)$ in k_A and k_B where these are the number of bits of A and B respectively.

Problem 2.7. Give an algorithm showing that MULT is linear in ADD (that is, uses ADD only $O(k)$ times), and so Comp(MULT) = $O(k^2)$.

We have said that Comp(ADD) = $O(k)$ and that Comp(MULT) = $O(k^2)$. It is not hard to see that subtraction and division (returning quotient and remainder) have complexities Comp(SUB) = $O(k)$ and Comp(DIV) = $O(k^2)$ respectively. We say that the running times are *linear* and *quadratic* respectively.

Problem 2.8. If Comp(A) = $O(k^2)$ and Comp(B) = $O(k^2)$ for problems A and B, we might be tempted to say Comp(A) = Comp(B). Why is this not okay?

Note

Actually, if you are clever about it, you can do MULT in running time $O(k^{1.45})$. For people doing heavy computation, this can be an important improvement over $O(k^2)$. For us it is not so important.

Problem 2.9. More common than bops are flops– floating point operations – which are the addition or multiplication of two floating point numbers of (say) 8 bits. Show that an algorithm has a running time of $O(k)$ flops if and only if it has a running time of $O(k)$ bops.

2.5.3 How about base 10?

Computers use base 2, and we think of our algorithms in these terms, but we will find it useful to talk in base 10. We may compute the running time of an algorithm as a function of the input integer n, instead of the number k of binary (base 2) digits.

If the input is n, then it has $k = \log_2 n$ binary digits. When we talk of the running time of the algorithm, an n will always stand for the input integer, and k will always stand for the number of binary digits in the input integer.

As Comp(ADD) = $O(k)$ we will also say Comp(ADD) = $O(\log_2 n)$.

Problem 2.10. What is the complexity of MULT in terms of input size n?

2.5.4 Polynomial Reductions

Problems with complexity $O(k)$ or $O(k^2)$ are usually considered to be 'easy'. What 'hard' is depends on who you talk to. For some mathematicians a problem is easy as long as its complexity is polynomial, and is hard if its complexity is exponential. (What about in between?) For some mathematician a problem of complexity above k^6 is hard. For all of them showing that a problem is hard is a hard problem.

Note

There are many problems, those as NP-complete (NPC) problems, for which the best known algorithm is exponential. We cannot prove that these problems will not be solved in polynomial time, but most mathematicians believe this. NPC problems are usually considered hard.

So what we do, to show that a problem is 'hard', is to show that it is as hard, (or not much easier) then another problem that experience has shown us is hard. To show it is easy, we can show it is not much harder than another easy problem.

We say that $\text{Comp}(B)$ is *polynomial in* $\text{Comp}(A)$ if $\text{Comp}(B) = f(k, \text{Comp}(B)$ for some polynomial function f. We do this via polynomial reductions. We give an example of a polynomial reduction in a graph theory. It is easy to explain, and we don't want to spoil of the upcoming good bits in our course.

Example 2.5. The to k-colour a graph, you must assign one of k colours to each of the vertices so that adjacent vertices get different colours. Not all graphs can be 3-coloured, but for those that can, the task of 3-colouring is known to be hard.

Could the problem of 4-colouring a 4-colourable graph be easy?

No! If it were then we could use this, as follows, to easily colour a 3-colourable graph G:

i). Add a vertex v_0 to G that is adjacent to all other vertices. (The new graph G' is 4-colourable.

ii). Use the good algorithm for 4-colouring to colour G'. We may assume that v_0 gets colour 4.

iii). Remove v_0. We have a 3-colouring of G.

The running time of this algorithm is not much more than the running time of the algorithm for 4-colouring. The instance is maybe slightly bigger, having one more vertex and n more edges, but this is at worst linear in the size of G which we measure by its number of vertices.

In the example above we showed that we can solve $\text{Comp}(3-Col)$ in polynomial time, using an *oracle* for $\text{Comp}(4-Col)$. This is a *polynomial reduction* of A to B. If one exists, we say that B is *polynomially reducible* to A and write $B \leq_{\text{poly}} A$. A and B are *polynomially equivalent*, written $A =_{poly} B$ if $A \leq_{poly} B$ and $B \leq_{poly} A$.

Problem 2.11. Show that if $\text{Comp}(A) \leq_{poly} \text{Comp}(B)$ and $\text{Comp}(B)$ is polynomial then $\text{Comp}(A)$ is polynomial. Show that if $\text{Comp}(A)$ is exponential, then $\text{Comp}(B)$ is exponenial.

1.7.5 Back to encryption schemes

In our public key encryption schemes, we have an encryption algorithm e and a decryption algorithm d. As a rule of thumb, the scheme is good if e and d are polynomial, but cryptoanalysis is exponential.

If encryption has complexity $f(k) = 10k^3 + 1000k$ while cryptoanalysis has complexity $g(k) = .2 \cdot 2^k$, then this isn't so good for blocksize $k = 10$, but choosing k properly, say $k = 60$ we get that $g(60) = 2 \times 10^{17}$ is about 10^{10} times as big as $f(60) \approx 2 \times 10^7$. Today's computers can do $O(10^7)$ bops in a second, it would take 100 years to do 10^{17}. So for the right blocksize, this is a good code.

Similarily, there can be good codes, according to our definition, that are good, but the blocksize necessary to ensure that cryptoanalysis is impossible also makes encryption impossible for today's computers. We still call them good though, because when computers get fast enough, they will become useful.

Problem 1.12. Assume for some encryption scheme that encryption and decryption have complexity $f(k) = 10k^2$ and cryptoanalysis has complexity $g(k) = 200k^6$. Is this scheme useful for any blocksize k?

1.2 Divisibility and GCDs

Recall that for integers a and b, we we say that a *divides* b, and write $a \mid b$, if there is another integer q, called the *quotient*, such that $aq = b$. If a divides b, then a is a *divisor* or *factor* of b, and b is a *multiple* of a.

Where all letters involved are integers it should be simple from this definition to prove such properties as:

i). $a|0$

ii). $0|a \Rightarrow a = 0$

iii). $1|a$

iv). $a|1 \Rightarrow a = \pm 1$

v). $a|b_1$ and $a|b_2 \Rightarrow a|c_1b_1 + c_2b_2$

vi). If $b + c = d$, and a divides two of b, c, d then it divides all of them.

With a bit more work you should be able to prove the Division Algorithm: that for two integers a and b there is a unique pair of integers q and r with $0 \leq r < a$ such that

$$a = b \cdot q + r.$$

In this case r is called the *remainder* upon division of a by b.

We mentioned before, there is an algorithm to do this in $O(k^2)$ time.

The greatest common divisor $\gcd(a, b)$ of two integers a and b is the greatest integer that divides them both. From the last of the six properties given above, and the Division Algorithm, you should be able to find the gcd of two integers using the *Euclidean Algorithm (EUC).*

```
def EUC(a,b):
  SET r_{-1} = a, r_0 = b and i = 1
  WHILE r_{i-1} > 0 do:
    LET r_i be the remainder of DIV(r_{i-2},r_{i-1})
  RETURN r_{i-1}
```

Example 1.6. To find $\gcd(1253, 234)$ we use the Euclidean Algorithm:

Step	r_{i-2}	$=$	r_{i-1}	$\cdot$	q_i	$+$	r_i
$i=1$	1253	$=$	234	$\cdot$	5	$+$	83
2	234	$=$	83	$\cdot$	2	$+$	68
3	83	$=$	68	$\cdot$	1	$+$	15
4	68	$=$	15	$\cdot$	4	$+$	8
5	15	$=$	8	$\cdot$	1	$+$	7
6	8	$=$	7	$\cdot$	1	$+$	1
7	7	$=$	1	$\cdot$	7	$+$	0

So $\gcd(1253, 234) = 1$.

Problem 1.13. Use the euclidean algorithm to compute $\gcd(291, 252)$ and $\gcd(5132664589, 128437704317)$.

Lets determine how long this algorithm takes for integers a and b each having k bits. There are two things that we consider. One is how many steps the algorithm requires, and the other is how long each step is.

Observe first that for each i we have

$$r_{i-2} = r_{i-1} \cdot q_i + r_i$$

where $q_i \geq 1$ and $r_{i-1} > r_i$. So $r_{i-2} > 2r_i$. Where k_i is the number of bits of r_i, we there for have that $k_i \leq k_{i-2} - 1$: every two steps we go down by at least one bit.

We stop when we get to 1 bit, so we have $O(k)$ steps. In each step we are doing DIV of numbers of at most k bits. This takes time $O(k^2)$. So overall the algorithm takes time $O(k^3)$.

Problem 1.14. Using the fact that the division in the $2i^{th}$ step is of numbers having about $k - i$ bits, and assuming that this takes time exactly $(k - i)^2$ show that the algorithm has running time at most $f(k) = k^3/3$?

Let a have k_a bits and b have k_b bits. What is the running time of the algorithm in k_a and k_b?

Note

This is a pretty fast algorithm GCD will be one of our easy problems.

1.2.1 The Extended Euclidean Algorithm

The Euclidean algorithm allows us to write $\gcd(a, b)$ as a linear combination of a and b. Recall we had

Step	a	$=$	b	$\cdot$	q	$+$	r
1	1253	$=$	234	$\cdot$	5	$+$	83
2	234	$=$	83	$\cdot$	2	$+$	68
3	83	$=$	68	$\cdot$	1	$+$	15
4	68	$=$	15	$\cdot$	4	$+$	8
5	15	$=$	8	$\cdot$	1	$+$	7
6	8	$=$	7	$\cdot$	1	$+$	1
7	7	$=$	1	$\cdot$	7	$+$	0

From the first line we get that $83 = 1253 - 5(234)$. Continuing line by line, we get

83								$=$		(1253)	-5	(234)
68	$= 234 - 2(83)$	$=$	$(0-2)$	(1253)	$+$	$(1-2(-5))$	(234)	$=$	-2	(1253)	$+11$	(234)
15	$= 83 - 68$	$=$	$(1+2)$	(1253)	$+$	$(-5-11)$	(234)	$=$	3	(1253)	-16	(234)
8	$= 68 - 4(15)$	$=$	$(-2-4(3))$	(1253)	$+$	$(11-4(-16))$	(234)	$=$	-14	(1253)	$+75$	(234)
7	$= 15 - 8$	$=$	$(3-(-14))$	(1253)	$+$	$(-16-75)$	(234)	$=$	17	(1253)	-91	(234)
1	$= 8 - 7$	$=$	$(-14-17)$	(1253)	$+$	$(75-(-91))$	(234)	$=$	-31	(1253)	$+166$	(234)

Problem 1.15. Use the Extended Euclidean Algorithm to express $\gcd(a, b)$ as a linear combination of a and b for the examples in Problem 1.13

Problem 1.16. Write an Extended Euclidean Algorithm function in Sage. Read Exercise 1.12 for a more efficient algorithm for the Euclidean Algorithm.

Problem 1.17. Show that the Extended Euclidean Algorithm has running time $O(k^3)$, (where we assume a and b both have about k binary digits).

Problems from the Text

Page 49: (1.9),(1.10), 1.11

1.3 Modular Arithmetic

Note

The book doesn't use the notation $a \equiv_m b$.

You've probably seen Modular Arithmetic before, so we skip many details. The text fills in the details if you are unfamiliar.

Fixing an integer m, we say integers a and b are *congruent mod* m, and write $a \equiv b \mod m$, or more commonly $a \equiv_m b$, if $m \mid (a - b)$. Equivalently, a and b are congruent mod m if and only if there exists some integer c such that $a = cm + b$.

Example 1.7.

$$-2 \equiv_7 5 \equiv_7 12 \equiv_7 712$$

Some basic properties of Modular Arithmetic are the following.

If $a_1 \equiv_m b_1$ and $a_2 \equiv_m b_2$ then

i). $a_1 a_2 \equiv_m b_1 b_2$,

ii). $a_1 + a_2 \equiv_m b_1 + b_2$,

iii). $a_1 - a_2 \equiv_m b_1 - b_2$.

Problem 1.18. Prove these properties.

One sees that $a \equiv_m r$ where r is the remainder upon division of a by m:

$$a = q \cdot m + r$$

So every integer is congruent modulo m to a unique integer in $\{0, 1, \ldots, m-1\}$.

We say that b is a *multiplicative inverse* of a modulo m, if $ab \equiv 1 \mod m$. If a has a multiplicative inverse, it is *invertible modulo m.*

For example, modulo 10, 7 has an multplicative inverse 13,

$$7 \cdot 13 = 91 \equiv 1 \mod 10,$$

but 5 does not, as all multiples of 5, $0, 5, 10, 15, 20, 25$ are congruent to 0 or 5 modulo 10. Observe that 3 is also a multiplicative inverse of 7; but $3 \equiv 13 \mod 10$.

Proposition 1.8. *If a is invertible modulo m, its inverse is unique modulo m, that is, if b and b' are inverses of a, then $b \equiv_m b'$.*

Proof. Assume that b and b' are inverses of a. Then

$$a(b - b') = ab - ab' \equiv_m a - a = 0$$

and multiplying both sides of this by b we get

$$(b - b') = 1(b - b') \equiv_m ba(b - b') \equiv_m b0 = 0.$$

But $(b - b') \equiv_m 0$ means that $b \equiv_m b'$, as needed. □

In light of this, if a is invertible modulo m, we let a^{-1} or $1/a$ refer to the unique inverse in the range $\{1, \ldots, m-1\}$, and call this **the inverse modulo m.**

Example:

$$4/7 = 4 \cdot 7^{-1} \equiv_{10} 4 \cdot 3 = 12 \equiv_{10} 2.$$

Two integers a and b are *relatively prime* if $\gcd(a, b) = 1$.

Proposition 1.9. *Let $m \geq 1$ be an integer. An integer a is invertible modulo m if and only if it is* relatively prime to m.

Proof. See Prop 1.13 of the text. □

Problem 1.19. Is 2536 invertible modulo 353? If so, what is its inverse?

Note

As the extended euclidean algorithm has running time $O(k^3)$ for inputs both having k bits, the problem INV(m) of finding the inverse of an instance a modulo m has complexity $O(k^3)$ where k is the number of bits of m.

This will be another 'easy' problem for us.

1.3.1 The modular ring $\mathbb{Z}_m$

The map taking and integer a to its remainder r upto division by m is the canonical quotient map that takes the ring $\mathbb{Z}$ to the quotient ring $\mathbb{Z}_m := \mathbb{Z}/m\mathbb{Z}$ *modulo* the ideal $m\mathbb{Z}$. In fact, the properties you proved in Problem 1.18 simply show that the 'remainder' map is a ring homomorphism of $\mathbb{Z}$ to $\mathbb{Z}_m$. That $a \equiv_m b$ means that the images a and b in $\mathbb{Z}_m$ are equal in $\mathbb{Z}_m$. Without getting too rigourous, working in $\mathbb{Z}_m$ simply means we can to modular arithmetic without saying 'mod' all the time: In $\mathbb{Z}_7$, $2 + 6 = 1$.

Note

The book doesn't use the notation $\mathbb{Z}_m$, and often this notation means something else.

Proposition 1.9 tells us that an element a of $\mathbb{Z}_m$ has a multiplicative inverse, or is a *unit*, if and only if is is relatively prime to m. The set $(\mathbb{Z}_m)^*$ of units of $\mathbb{Z}_m$ form a group. The *order* $|(\mathbb{Z}_m)^*|$ of this group of units will be important. It occurs enough that there is a function to describe it– *Euler's phi function:*

$$\phi(m) = |(\mathbb{Z}_m)^*|.$$

Problem 1.20. What is $\phi(32)$? What is $\phi(6)$? What is $\phi(p \cdot q)$ where $p \neq q$ are prime?

Now. We are going to be using $\phi(N)$ a lot. Lets settle this once and for all.

Theorem 1.10 (Euler's Formula)**.** *Where $\phi(N) = |\mathbb{Z}_N^*|$ is the number of elements in $\mathbb{Z}_N$ that are relatively prime to N, and $N = p_1^{e_1} \cdots p_d^{e_d}$ is the prime factorisation of N,*

$$\phi(N) = \prod_{i=1}^{d} (p_i - 1) p_i^{e_i - 1} = N \prod_{i=1}^{d} (1 - 1/p_i)$$

Proof. We have already seen that $\phi(p) = p - 1$, for prime p. When $N = p^e$ for some exponent $e \geq 2$, the elements in $\{1, \ldots, p^e\}$ that are not prime to p are exactly those with p as a factor. There are p^{e-1} of these, so $\phi(p^e) = p^e - p^{e-1} = (p-1)p^{e-1}$.

Problem 1.21. Complete the proof by showing that $\phi(ab) = \phi(a)\phi(b)$ for mutually prime integers a and b.

□

In several applications, we will be computing large powers of an integer a in $\mathbb{Z}_m$. To compute powers in $\mathbb{Z}$ is exponential in a and b: it must be, even writing out a^b is exponential. Modulo m however, the size of a^b is $O(m)$ for all a and b. To compute 7^{415} and then reduce it modulo 3317, we have to do several multiplications (perhaps 414) it will be best to reduce it modulo after each multiplication, so that we are not multiplying increasingly large numbers.

Problem 1.22. Compute 7^5 modulo 11.

But we still have to do b multiplications of two integers of size m. So the obvious algorithm has $O(b)$ mutliplications of integers of size m. This is still $O(2^{k_b} k_m^2)$, which is exponential in the number k_b of binary digits of b. Usually when we are working in $\mathbb{Z}_m$ we simply assume that all input numbers are about the size of m, so have $k = k_m$ bits. So this algorithm in $\mathbb{Z}_m$ has running time $O(2^K \cdot k^2)$.

We can do better. Observe that $415 = 2^8 + 2^7 + 2^6 + 2^2 + 2 + 1$. So

$$7^{415} = 7^{2^8+2^7+2^6+2^2+2+1} = 7^{2^8} \cdot 7^{2^7} \cdot 7^{2^6} \cdot 7^{2^2} \cdot 7^2 \cdot 7.$$

It takes us $7 \approx \log_2(415)$ multiplications to evaluate all the factors 7^i modulo 3317, and about the same to multiply them together. This is 7 verses the 415 operations required to so it stupidly. This is called the fast powering algorithm.

Problem 1.23. What is the running time of the fast powering algorithm in $\mathbb{Z}_m$? This will be the complexity of the problem POW($\mathbb{Z}_m$) of powering modulo m.

Problem 1.24. Write a program in Sage to do fast powering. (Sage, of course, uses fast powering, but write it out yourself. The worksheet on the website does a slow powering algorithm.)

Problems from the Text

Page 51: 1.16, 1.18, 1.20

1.4 Prime numbers, Unique Factorisation, and finite fields

Recall that an integer $p \geq 2$ is *prime* if and only if for positive integers d

$$d \mid p \Rightarrow d = 1 \text{ or } p.$$

I expect that you've seen and could prove the following results:

- If a prime p divides ab for integers a and b, then it divides a or b.
- There are infinitely many primes.

The first of these is infact only true of 1 and primes, and their negatives. Further, you should have seen the following. You certainly believe it.

Theorem 1.11 (Fundamental Theorem of Arithmetic)**.** *Let $a \geq 2$ be an integer. Then a can be written as a product of primes:*

$$a = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots p_d^{e_d},$$

where each p_i is a distinct prime, and each e_i is a positive integer. Furthermore, except for order, this representation is unique.

This representation is called the *prime factorisation* of the integer a. For a particular prime p_i, the exponent e_i is called the *order of p_i in a*, and is denoted $e_i = \text{ord}_{p_i}(a)$. If a prime p does not occur in in the prime factorisation of a, then $\text{ord}_p(a) = 0$.

Example 1.12. The prime factorisation of 204 is

$$2^2 \cdot 3 \cdot 17,$$

so $\text{ord}_2(204) = 2$, $\text{ord}_{17}(204) = 1$, and $\text{ord}_7(204) = 0$.

For prime p, the numbers $1, \ldots p-1$ are all relatively prime to p, and so we have the following useful fact.

Proposition 1.13. *If p is a prime then the group $(\mathbb{Z}/p\mathbb{Z})^*$ of units of $\mathbb{Z}/p\mathbb{Z}$ consists of the set*

$$\{1, 2, \ldots, p-1\}$$

of non-zero elements.

By Proposition 1.13, we have that for prime p, $\mathbb{Z}/p\mathbb{Z}$ is not just a ring, but a field. From field theory we know that the is a unique field of order p. This field is often denoted $\mathbb{F}_p$.

Problems from the Text

Page 52: 1.26, 1.29

1.5 Powers and Primitive Roots in Finite Fields

We will use properties of $\mathbb{Z}_p \cong \mathbb{F}_p$ that we know from field theory. Several of these properties, we can prove without much theory though, so we do that.

Theorem 1.14 (Fermat's Little Theorem)**.** *Let p be prime, and a be an integer in $\mathbb{Z}_p$. Then $a^{p-1} = 0$ if $p \mid a$ and $a^{p-1} = 1$ otherwise.*

Proof. If $p|a$ then $a^{p-1} = 0^{p-1} = 0$, so we may assume that $p \nmid a$, and so a is in $\mathbb{Z}_p^*$. The elements

$$a, 2a, \ldots (p-1)a$$

are distinct elements, for if $ia = ja$ then $p \mid j - i$ which implies $i = j$. So we have

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

Thus $a^{p-1} = 1$. □

Immediate corollaries of this are the fact that for a with $\gcd(a, p) = 1$, we have

- $a^p \equiv_p a$
- $a^{-1} \equiv_p a^{p-2}$

This second seems nicer than the Extended Euclidean Algorithm for computing inverses modulo p. Conceptually it is easier, but computationally it is fast powering, so also $O(k^3)$.

An important problem for us will be the problem PRIME(n) of determining if an integer n is prime. It is a somewhat difficult problem, which we discuss more in Chapter 3.

Theorem 1.14 gives us a useful check in this problem.

Given an integer n,

i). Choose some $a \in \mathbb{Z}_n$.

ii). Compute a^{n-1}.

iii). If a^{n-1} is anything other than 1, then n is not prime, done. Otherwise, go back to i).

Now if n is not prime, this algorithm usually tells us this in a couple of loops. If it is prime, the algorithm doesn't stop, so it isn't good as an algorithm, but we can stop it after 10 or 100 or k loops, and be pretty sure that n is prime.

But the test is not perfect: there are known composite numbers n for which any a that is relatively prime to n yields $a^{n-1} \equiv_n 1$. Such numbers are called Carmichael numbers. We will look at better prime verification algorithms later.

The *order* $\mathrm{ord}(a)$ of an element a of $\mathbb{Z}_p$ is the minimum n such that $a^n \equiv_p 1$. The following is a special case of Legrange's Theorem. You can prove it using Fermat's Little Theorem (or some simple group theory: the cosets in $\mathbb{Z}_p^*$ of the subgroup generated by a partition $\mathbb{Z}_p^*$ and all have the same size).

Proposition 1.15. *Let p be a prime, and $a \in \mathbb{Z}_p^*$. . If $a^n \equiv_p 1$ then $\mathrm{ord}(a) \mid n$. In particular, the $\mathrm{ord}(a) | p-1$.*

Proof. See text. This is Prop 1.30.

□

Now we know from field theory that any finite field has a cyclic multiplicative group. This is too much to prove here, even for finite fields $\mathbb{F}_p$ of prime order, (there are also finite fields of prime-power order), but we will use it.

Theorem 1.16 (The Primitive Root Theorem)**.** *Let p be a prime number. Then there exists an element $g \in \mathbb{F}_p^*$ such that*

$$\mathbb{F}_p^* = \{1, g, g^2, \ldots, g^{p-2}, \}.$$

Note

This is called the primitive *root* theorem because it is saying that the polynomial $x^n = 1$ has a primitive root g.

The elements g of the theorem are *generators* or *primitive elements* of $\mathbb{F}_p^*$, the text also calls them *primitive roots*.

Problem 1.25. Find a primitive element of $\mathbb{F}_7^*$. How many are there?

You might do this by taking an element a to every power in $1, 2, \ldots, 6$, but you don't really have to do this. You only have to check certain powers. Which ones?

Problem 1.26. Show that 10 is a primitive element of $\mathbb{F}_{47}$.

Problem 1.27. How many primitive elements are there of $\mathbb{F}_7^*$?

Notice that all the results of this section can easily be proved if we assume the Primitive Root Theorem.

Though $\mathbb{Z}_p$ and $\mathbb{F}_p$ are isomorphic and the multiplicative structure is clear when representing the field as $\mathbb{F}_p = \{0, 1, g, g^2, \ldots, g^{p-2}\}$ for some primitive

root g, what we will often use is the fact that the multiplicative structure in terms of $\mathbb{Z}_p$, is not clear.

That is, if we expand the powers

$$g, g^2, \ldots, g^{p-2}$$

for fixed generator g then we get what seems like a random permutation of

$$2, 3, 4, \ldots, p-1.$$

Because of this we will use both $\mathbb{Z}_p$, with elements $\{0, \ldots, p-1\}$ and $\mathbb{F}_p$, with elements $\{g^0, g^1, \ldots g^{p-1}\}$ for some generator g. We will treat them notationally differently, but will exploit the fact that the are isomorphic.

Problems from the Text

Pge 53: 1.30, 1.31, 1.32, 1.34, 1.36

2 Discrete Logarithms and Diffe Hellman

Section 2.1 is history. Read it if you are interested.

2.2 The Discrete Logarithm Problem

Definition 2.1. Let g be a primitive root of $\mathbb{F}_p$. For any element $h \in \mathbb{F}_p^*$, the *discrete log of h (in the base g)* is the solution $\log_g(h)$ modulo p to the equation

$$g^x = h.$$

Problem 2.1. Why is there necessarily a solution, and why is it unique modulo $p-1$?

The discrete log function $\log_g : \mathbb{F}_p^* \to (\mathbb{Z}/p\mathbb{Z})^*$ is a one-to-one function. As we mentioned before, viewed as as function $\log_g : (\mathbb{Z}/p\mathbb{Z})^* \to (\mathbb{Z}/p\mathbb{Z})^*$, $\log_g$ mixes things up pretty good.

Indeed, take a look at the Chapter 2 Sage worksheet for an example.

Now how do we evaluate the discrete log function.

Problem 2.2. Find $\log_g(g^3)$ in $\mathbb{F}_{13}^*$

Ah! Its 3, maybe that was too easy.

Example 2.2. 2 is a primitive root of $\mathbb{F}_{13}^*$. Lets find $\log_2 9$.

The brute force way to do this is start computing $2, 2^2, 2^3 \ldots$:

$$2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7.$$

We see that 9 is the 8^{th} power, so $\log_2(9) = 8$.

We can do this in $\mathbb{F}_{13}^*$, but it's a bad algorithm. We could have stopped after writing down 9, but computing $\log_g(a)$ in $\mathbb{F}^*(p)$ like this takes $O(p) = O(2^{k_p})$ multiplications (each $O(k_p^2)$). This is much longer than the $O(k_p)$ multiplications of fast powering.

We can do better than $O(2^k)$, but are not able to do anywhere near $O(k)$.

Problem 2.3. Recall that 10 is a generator of $\mathbb{F}_{47}^*$. Compute $\log_{10}(22)$.

Okay, you did that. Did you write out all 46 powers of 10 in $\mathbb{F}_{47}$? You might have done better to start by factoring $46 = 2 \cdot 23$. If 10 is not primitive, it has order dividing 46. So we only have to show that $10^2 (= 100 = 6)$ and $10^{23} = (10^{16} * 10^4 * 10^2 * 10)$ are not 1.

This seems a better way. But it is not so good either: factoring is hard! So finding a primitive root seems tricky right now. It is something we will want to do, but we don't need it to define the discrete log.

Definition 2.3 (The Discrete Log Problem (DLP(p)))**.** Let p be a prime. DLP(p) is the following problem.

> **Input:** A generator g of $\mathbb{F}_p^*$ and an element h.
> **Output:** The integer $\log_g(h)$.

One can define the discrete log problem more generally for any group G and element g of G. The problem would then also involve deciding if a discrete log $\log_g(h)$ exists for given h. We won't do this.

Note

The DLP is our first 'hard' problem. We will improve the complexity $O(2^k)$ we have for it, but it will remain hard.

Problems from the Text

Page 105: 2.3, 2.4, 2.5

2.3 The Diffe-Helleman Key Exchange

The Diffe-Helleman Key Exchange is a system whereby Alice and Bob can share a secret key. It is not a cipher, as we have defined it, as Alice and Bob cannot choose the message they share. But, in public, they share a secret word which only they will know. This can be used, for example, to create a secret key which they will then use as the key in some symmetric cipher.

Here is how it works.

i). Alice and Bob agree on a public prime p and a generator g of $\mathbb{F}_p^*$.

ii). Alice chooses a secret key a and Bob chooses a secret key b.

iii). Alice sends Bob $A = g^a$ and Bob sends Alice $B = g^b$ (publicly).

iv). Alice computes B^a and Bob computes A^b.

As $B^a = g^{ab} = g^{ba} = A^b$, Alice and Bob have the same key $k = g^{ab}$. If Eve was evesdropping on their communications, she knows $p, g, A = g^a$ and $B = g^b$, but she doesn't know a or b.

The security of the key exchange depends on the fact that the following problem is hard.

Definition 2.4 (The Diffe-Hellman Problem (DHP))**.** Let p be a prime and g be an integer. DHP(p, g) is the following problem.

Input: Elements $A = g^a$ and $B = g^b$ of $\mathbb{F}_p^*$
Output: The element g^{ab} of $\mathbb{F}_p^*$.

Problem 2.4. Assuming that DHP(p, g) is hard, show that DLP$(\mathbb{F}_p^*, g)$ is hard by giving a polynomial reduction of DHP(p, g) to DLP$(\mathbb{F}_p^*, g)$

Note

We do not know if DHP is polynomially reducible to DLP. It is another problem that we will just believe is hard.

Problem 2.5. You and Bob have shared a secret key k in $\mathbb{F}_{101}^*$. How would you use this to share the message 'TOMMORROW AT LUNCH'?

Problem 2.6 (Sage)**.** You are evesdropping on Alice and Bob. They publicly share that they will do the Diffe Hellman Key exchange with prime $p = 23473$ and generator $g = 5$. They exchange the numbers $A = 18388$ and $B = 5786$. What is their shared secret key k?

Problems from the Text

P. 106: 2.6

2.4 The El-Gamal Public Key Cryptosystem

The El-Gamal was not historically the first public key cryptosystem (PKC), but we look at it first because it depends on the DLP.

Recall that in a PKC, anyone can encrypt a message to send to Alice, but only Alice can decrypt a message.

El-Gamal PKC for prime p and generator g of $\mathbb{F}_p^*$:

i). Alice chooses private key a, computes $A = g^a$, and publishes the public encryption key A.

ii). Bob chooses private key k, and for message m computes $c_1 = g^k$ and $c_2 = mA^k$. He sends (c_1, c_2) to Alice.

iii). Alice decodes (c_1, c_2) to m by computing

Note

The values p and g are public.

All numbers and calculations are in $\mathbb{F}_p^*$.

$$\begin{aligned} c_2/c_1^a &= mA^k/g^{ka} \\ &= mA^k/A^k \\ &= m \end{aligned}$$

Problem 2.7. Choose a private key a for El-Gamal PKC with $p = 13$ and $g = 2$. Give a friend the public key A and have them encode a message m. Decode it.

Definition 2.5 (The El-Gamal Problem EGP)**.** Let p be a prime and g be a generator of $\mathbb{F}_p^*$. EGP(p, g) is the following problem.

> **Input:** Elements $A = g^a$ and $c_1 = g^k$ and $c_2 = mA^k$ in $\mathbb{F}_p^*$.
> **Output:** m.

Theorem 2.6. EGP *and* DHP *are polynomially equivalent problems.*

Proof. Consider the following reduction of EGP to DHP.

Given $A = g^a$ and $c_1 = g^k$ and $c_2 = mA^k$, we use DHP on A and c_1 to compute $g^{ak} = A^k$. Inverting A^k and computing $c_2/A^k = mA^k/A^k = m$ gives m.

Besides a single application of DHP(p, g), this uses inversion ($O(k^3)$) and multiplication $O(k^2)$. So it has running time $O(k^3) + \text{Comp}(\text{DHP}(p, g))$. This is a polynomial reduction.

On the other hand, reduce DHP to EGP as follows.

Given g^a and g^b, solve EGP with $c_1 = g^b$ amd $c_2 = 1$. This gives $m = c_2/c_1^a = 1/g^{ab} = g^{-ab}$. Inverting this, we have g^{ab}.

This has running time $\text{Comp}(\text{EGP}(p, g)) + O(k^3)$ so is a polynomial reduction. □

Problem 2.8. Give a polynomial reduction of EGP to DLP. Are the two polynomially equivalent?

Problems from the Text

P. 106: 2.8, 2.9, 2.10

2.7 A Collision Algorithm for DLP

Recall the Lucy algorithm for solving the DLP(p, g) instance $g^x \equiv_p h$ is would be computing

$$g_1 = g \text{ then } g_2 = g_1 * g \text{ then } g_3 = g_2 * g \text{ then } \ldots.$$

For g having about $k = \log_2(p)$ bits this takes uptp 2^k steps of $O(k^2)$ each. So the algoritm has running time $O(2^k \cdot k^2)$. The following algorithm for the descrete log does better.

Shank's Babystep Giantstep (or Collision) Algorithm

i). Let $n = 1 + \lfloor \sqrt{p-1} \rfloor$.

ii). Create lists: $L_1 : 1, g, g^2, \ldots, g^n$
$L_2 : h, hg^{-n}, hg^{-2n}, \ldots, hg^{-n^2}$.

iii). Check L_1 and L_2 for a common element c.

iv). Where $g^i = c = hg^{-jn}$ return $x = jn + i$.

To see that this works, observe that if $h = g^x$, then by the division algorithm x can be written as $x = jn + i$ where $i < n$, so there is such a c. On the other hand, if $g^i = hg^{-jn}$ then

$$h = g^i g^{jn} = g^{i+jn} = g^x$$

and so $x = \log_g(h)$.

Problem 2.9. Show that the running time of the Collision algorithm is $O(2^{k/2}k^2) = O(\sqrt{2}^k k^2)$ where $k = \log_2(p)$.

Problem 2.10. The text says that one can find a common integer in two ordered lists of n integers, in time $\log(n)$. Can this possibly be true?

Problems from the Text

P. 108: 2.17

2.8 The Chinese Remainder Theorem

Definition 2.7 (The Chinese Remainer Problem (CRP))**.**

Input: Identities of the form $x \equiv_{m_i} a_i$ for integers m_i and a_i.
Output: An integer x satisfying all identities.

Problem 2.11. Solve the CRP for input $x \equiv_{11} 3$ and $x \equiv_{13} 5$.

Problem 2.12. Where x is your solution for the previous problem, show that that $x + z \cdot 11 \cdot 13$ also satisfies the identities for any integer z. Show that there is only one solution modulo $11 \cdot 13$.

This works in general.

Theorem 2.8 (The Chinese Remainder Theorem)**.** *Let $m_1, \ldots, m_d$ be pairwise relatively prime integers, let $a_i \in \mathbb{Z}_{m_i}$ for $i \in [d]$, and let $M = \prod m_i$. There is a unique x in $\mathbb{Z}_M$ satisfying $x \equiv_{m_i} a_i$ for each $i \in [d]$.*

Proof. We prove existence of a solution x. It is clear that we can find a solution $x_1 \in \mathbb{Z}_{m_1}$ to the first identity $x \equiv_{m_1} a_1$; assume that we have a solution $x_i \in \mathbb{Z}_{M_i}$, where $M_i = \sum_{j=1}^{i} m_j$, to the first i identities. So

$$x_i + c_i M_i$$

also satisfies the identities for any integer c_i. We find x_{i+1} of this form that also satisfies

$$x_{i+1} \equiv_{m_{i+1}} a_{i+1}.$$

Indeed,

$$x_i + c_i M_i \equiv_{m_{i+1}} a_{a+i} \iff c_i M_i \equiv_{m_{i+1}} a_{i+1} - x_i$$

so inverting M_i modulo m_{i+1} we get

$$c_i = (a_{i+1} - x_i)/M_i$$

modulo m_{i+1} and let

$$x_{i+1} = x_i + c_i M_i.$$

As x_{i+1} is of the form $x_i + c_i M_i$ it satisfies the first i identities. And modulo m_{i+1} we have

$$x_{i+1} = x_i + c_i M_i \equiv x_i + ((a_{i+1} - x_i)/M_i)M_i = x_i + (a_{i+1} - x_i) = a_{i+1}$$

as needed.

□

Problem 2.13. Prove the 'uniqueness' in Theorem 2.8 with a counting argument.

It follows that the problem can be rephrased.

Definition 2.9 (The Chinese Remainer Problem (CRP))**.**

> **Input:** Identities of the form $x \equiv_{m_i} a_i$ for integers m_i and integers $a_i \in \mathbb{Z}_{m_i}$.
> **Output:** An integer $x \in \mathbb{Z}_M$, where $M = \prod m_i$, satisfying all identities.

Problem 2.14. Write an algorithm for the CRP out in Sage code or pseudocode, and compute its running time (in d and the number k of bits in $\max\{m_i\}$.

The CRP algorithm is useful in solving equations. Observe that a solution, say to

$$x^2 \equiv_{143} 27$$

would also be a solution to

$$x^2 \equiv_{11} 5 \text{ and } x^2 \equiv_{13} 1.$$

Problem 2.15. Solve the equation $x^2 \equiv_{143} 27$ as follows. Find a solution a_{11} to $x^2 \equiv_{11} 5$ and a solution a_{13} to $x^2 \equiv_{13} 1$. Use the Chinese Remainder Theorem to find a such satisfying $a \equiv_{11} a_{11}$ and $a \equiv_{13} a_{13}$. Show that a is a solution to the original equation. This is called *lifting* the solutions a_{11} and a_{13} to the solution a modulo 143.

Problems from the Text

P. 108: 2.18, 2.20, 2.21, 2.23, 2.25

2.9 The Pohlig-Hellman Algorithm

This algorithm uses the chinese remainder theorem to get a faster algorithm for the DLP problem.

It doesn't look at first that the CRP would be useful in solving

$$g^x \equiv_p h$$

as p is prime. But, recall that $g^{p-1} \equiv_p 1$, so the value of x that we are looking for in the DLP is in $\mathbb{Z}_{p-1}$.

Let $N = p_1^{e_1} p_2^{e_2} \dots p_d^{e_d}$ be the prime factorisation of $N = p - 1$. If $a = \log_g h$ is the solution to $h \equiv_p g^x$, then where $a_i = a \mod p_i^{e_i}$, we have

$$h^{N/p_i^{e_i}} \equiv_p (g^a)^{N/p_i^{e_i}} = (g^{N/p_i^{e_i}})^a = (g^{N/p_i^{e_i}})^{a_i + c p_i^{e_1}} \equiv_p (g^{N/p_i^{e_i}})^{a_i}.$$

So a_i is the solution to

$$(g^{N/p_i^{e_i}})^x \equiv_p h^{N/p_i^{e_i}}.$$

Though this is an equivalence modulo p, the solution is unique in $\mathbb{Z}_{p_i^{e_i}}$, rather than $\mathbb{Z}_p$. So it is relatively easy to find.

To solve the discrete log problem $g^x \equiv_p h$ then, we can find a solution a_i to

$$(g^{N/p_i^{e_i}})^x \equiv_p h^{N/p_i^{e_i}}$$

for each i, and then use the CRP to lift it to a solution a.

Pohlig Hellman Algorithm for DLP

i). Factor $N = p - 1 = p_1^{e_1} p_2^{e_2} \dots p_d^{e_d}$.

ii). For each i, let $g_i = g^{N/p_i^{e_i}}$ and $h_i = h^{N/p_i^{e_i}}$ and solve $g_i^{a_i} \equiv_p h_i$, using say Shank's Collision algorithm.

iii). Lift the images $a_i \in \mathbb{Z}_{p_i^{e_i}}$ up to a solution $x \in \mathbb{Z}_N$ using CRP.

Problem 2.16. Use the Pohlig Hellman Algorithm to solve $7^x \equiv_{433} 166$.

Now lets analyse the algorithm. Assume for the time being that N has already been factored for us. For each i we can solve the equation

$$g_i^{x_i} \equiv_p h_i$$

using say, Shanks Collision Algorithm, in time $O(k_i^2\sqrt{2}^{k_i})$ where $k_i = e_i \log_2 p_i$. This gives us the image $x_i \in \mathbb{Z}_{p_i^{e_i}}$ of a solution x to our main problem. Then we can lift these images to x using CRP in time $O(d^3k^2 + k^3)$ where $k = \max k_i$. Overall this has running time $O(d \cdot k^2\sqrt{2}^k)$. This is not so much longer than running Shanks for only the largest $p_i^{e_i}$ in the factorization of $p-1$ rather than for p itself! Not bad, compared to the $O((\sum k_i)^2\sqrt{2}^{\sum k})$ of using Shanks directly.

2.9.1 A step further

One can actually do better. With similar ideas, you can show that a solution to

$$g^x \equiv_{p^e} h$$

can be lifted from solutions to

$$(g^x)^{p^i} \equiv_{p^e} h^{p^i},$$

for $i = e-1, e-2, \ldots, 1$. At each step one must essentially solve DLP modulo p, so this takes $O(ek^2\sqrt{2}^k)$ where $k = \log_2 p$. Using this in the Pohlig-Hellman algorithm, our complexity becomes $O(dek^2\sqrt{2}^k)$ as compared to the $O(d(ek)^2\sqrt{2}^{ek})$ we had there.

This should be a warning: when we are using a cryptosystem that can be solved by DLP, as all those we have seen this chapter can be, we should choose a prime p such that $p-1$ has a large factor.

Another thing we can do is make sure that $p-1$ is hard to factor: this also comes down to having at least two large factors, as we will see in the next chapter.

Problems from the Text

P. 110: 2.26, 2.28a

3 Integer Factorisation and RSA

The security of our previous cyptosystems were based on the fact that DLP and related problems are hard. For the next cryptosystem, it will be based on the fact that taking e^{th} roots modulo N is hard. This, in turn, is based on the fact that factoring N, and so determining $|\mathbb{F}_N^*|$, is hard.

The e^{th} root $c^{1/e}$ of a number c in $\mathbb{Z}_N$ is a value x such that $x^e \equiv_N c$.

We will encode a message m as

$$c = m^e \mod N,$$

so to recover m we have to find the e^{th} root $c^{1/e}$ of c modulo N.

This seems a wonderful idea, but the e^{th} root of a number c might not exist. Indeed, it might not even exist when N is prime.

Problem 3.1. Show that if e is not relatively prime to $p-1$ then the e^{th} root $c^{1/e}$ modulo p is not well defined. That is, show that there may be zero solutions x to $x^e \equiv_p c$ or there may be more than one.

Replacing N with a prime p, we can show by Fermat's theorem that the e^{th} roots exist and are unique for all $c \not\equiv_p 0$, as long as $\gcd(e, p-1) = 1$. Indeed, we know that $|\mathbb{F}_p^*| = p-1$, and so $c^{p-1} \equiv_p 1$. Finding d such that $de \equiv_{p-1} 1$ we have that

$$(c^d)^e = c^{de} \equiv_p c^1 = c,$$

so $c^{1/e} = c^d$. That is, we just have to invert e modulo $p-1$. We can do this, as long as e is relatively prime to $p-1$, using the euclidean algorithm.

For properly chosen e and N, we can generalise this to show that e^{th} roots exists modulo N, and that $c^{1/e}$ is c^d for d the multiplicative inverse of e modulo $\phi(N)$. This is what we are going to want to do in our next cryptosystem. We want to choose N so that $\phi(N)$ is hard to compute. We are going to take $N = pq$ for big primes p and q.

3.1 Roots modulo $N = pq$

Generalising the use of Fermats little theorem above, we know that if $\gcd(e, \phi(N)) = 1$ then e is invertible modulo $\phi(N)$, and so

$$c^{\phi(N)} \equiv_N 1,$$

for any c that is relatively prime to N. Thus for e with $\gcd(e, \phi(N)) = 1$, the e^{th} root function $c \mapsto c^{1/e}$ is well defined on the set of c such that $\gcd(c, N) = 1$.

Note that if $c^{\phi(N)} \equiv_N 1$ then $c^{\phi(N)+1} \equiv_N c$, and this is actually all that we need. So when $N = pq$ for primes pq we can actually say more.

Proposition 3.1. *Let p and q be distinct primes and $N = pq$. For all $c \in \mathbb{F}_N$,*

$$c^{\phi(N)+1} \equiv_N c.$$

Proof. There are four cases to consider depending on $\gcd(c, N)$, as this can be $1, p, q$, or N. For the case that $\gcd(c, N) = 1$, we gave the proof above using Fermat's Little Theorem, and for the case $\gcd(c, N) = N$ the result is trivial as $c \equiv_N 0$. We assume that $\gcd(c, N) = p$, the case $\gcd(c, N) = q$ is the same. We show that $c^{\phi(N)+1} \equiv_p c$ and $c^{\phi(N)+1} \equiv_q c$. From this is follows that p and q divide $c^{\phi(N)+1} - c$ and so N does, givng the result.

That $c^{\phi(N)+1} \equiv_p c$ is trivial as $c \equiv_p 0$. On the other hand, using Fermats little theorem, we have

$$c^{\phi(N)+1} \equiv_q c^{\phi(p)\phi(q)+1} \equiv_q 1^{\phi(p)} \cdot c = c,$$

as needed. □

Thus following is now immediate.

Corollary 3.2. *Let p and q be distinct primes and e be prime to $(p-1)(q-1)$. The equation*

$$x^e \equiv_{pq} c$$

has the unique solution $x = c^d$ in $\mathbb{Z}_{pq}$, where d is the multiplicative inverse of e modulo $(p-1)(q-1)$.

This shows that the following problem is well defined.

Definition 3.3 (The root problem (ROOT))**.** For $N = pq$ and ROOT(N) is the problem:

> **Input:** Exponent $e \in \mathbb{Z}_N$ prime to $(p-1)(q-1)$ and $c \in \mathbb{Z}_n$.
> **Output:** Integer $r \in \mathbb{Z}_N$ such that $r^e \equiv_N c$.

The following is a nice observation that can save some computing time.

Problem 3.2. Show that where $g = \gcd(p-1, q-1)$, and c is relatively prime to N, then

$$c^{\phi(N)/g+1} \equiv_N c.$$

Problem 3.3. Solve $x^5 \equiv_{35} 4$, $x^5 \equiv_{35} 7$, and $x^5 \equiv_{35} 26$. Solve $x^5 \equiv_{37} 4$

Problem 3.4. Solve $x^4 \equiv_{35} 4$

Problems from the Text

P. 176: 3.1(b), 3.3, 3.4

3.2 The RSA Public Key Cryptosystem

The RSA Public Key Cryptosystem is as follows.

RSA PKC:

i). Alice chooses private primes p and q, computes $N = pq$, chooses exponent $e \in \mathbb{Z}_N$ prime to $n = (p-1)(q-1)$, then computes $d = e^{-1}$ modulo $n/\gcd(p-1, q-1)$.

ii). Alice publishes N and e.

iii). Bob chooses private message m and sends Alice $c = m^e$ modulo N.

iv). Alice decodes c to m by computing c^d modulo N.

Definition 3.4 (The RSA Problem)**.** Let p and q be primes. RSA(p, q) is the following problem.

> **Input:** $N = pq$, $e \in \mathbb{F}_N$ prime to $(p-1)(q-1)$, and $c = m^e \mod N$.
> **Output:** m.

Clearly RSA is solvable if we have Alices information, so is polynomially reducible to the problem of factoring the product of two primes.

Definition 3.5 (The pq-Factorisation Problem pq-FACT)**.** Let p and q be primes. pq-FACT(p, q) is the following problem.

> **Input:** $N = pq$
> **Output:** p.

We do not know if pq-FACT is polynomially equivalent to RSA. But it is polynomially equivalent to the pq-EULER problem of computing $\phi(N)$ from $N = pq$.

Problem 3.5. Show that pq-FACT is polynomially equivalent to pq-EULER.

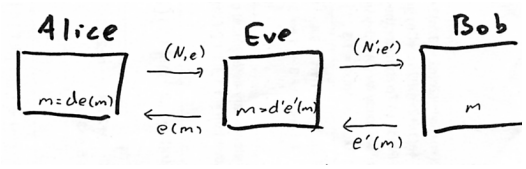
Problems from the Text

P. 177: 3.6, 3.7, 3.8, 3.9, 3.10

3.3 Implementation and Security Issues

There are often ways to attack a cryptosystem without solving the hard problem involved.

Any PKC can be beaten with a *man-in-the-middle* attack, which Eve would do by hijacking the lines of communication between Alice and Bob (maybe Eve is the government, or your ISP).



Bob initiate contact with Alice but Eve intercepts it and pretends she is Alice. She also initiates contact with Alice, pretending to be Bob, and Alice responds. Eve passes her response to Bob, only replacing Alice's public key with one of her own. She can then decrypt Bob's message. She re-encrypts it with Alice's key before sending it to her, so that Eve's involvement is not detected.

There is no real math involved in this, but it cautions us that in a PKC it is good to make sure we can verify the public key. We do this either with digital signatures, which we will see later, or maybe by simply making the public key very public, so that it can be verified from a independent source.

Let's look now at a more mathematical attack for RSA in particular.

For this one, Eve must trick Alice into decoding a 'random' message r. She might do this by telling Alice she needs her to decode r to verify that Alice is Alice.

- Eve overhears Bob's encoded message $c = m^e \mod N$ to Alice.
- She disguises it as $c' = k^e c = (km)^e$, and sends it to Alice claiming that it is her encoded random message r.
- Alice decodes c' to $r = km$, it seems random, so she is not suspicious, and sends it back to Eve.
- Eve computes r/k to get m.

Problems from the Text

P. 178: 3.12

3.4 Primality Testing

The security of RSA depends on the fact that it should take Eve a year to factor pq for primes p and q, so they should be big. But how do we find two such primes? It is not so good for Alice if it takes her a year to find them. Alice needs to find large primes relatively quickly.

To find a large prime, we basically guess a number and check if it is prime, so finding a prime comes down to the problem of checking if a number is prime.

How many guesses we must make depends on the density of the primes. We will look at that later. First we look at how we check if a number is prime. Both factoring n and checking if it is prime can be done in time $O(2^{k/2}k^2)$ by dividing n by every number upto $\sqrt{n}$. We need a better way.

Definition 3.6 (The primality problem PRIME)**.**
Input: Integer n
Output: 'Yes' if n is prime, 'No' otherwise.

Our algorithms for the primality problem will not solve it perfectly. If they answer 'No' then n is definitily not prime, but if they answer 'Yes' we only know that n is prime **with high probability**.

3.4.1 Check Small Integers

A number is N-prime if it has no factors less than or equal to N. Half of integers are 2-prime, $1/2 \cdot 2/3 = 1/3$ of integers are 3-prime,

$$\frac{1}{2} \cdot \frac{2}{3} \cdot \frac{4}{5} = \frac{4}{15}$$

of integers are 5-prime, etc.

Problem 3.6. Show that $\prod \frac{p-1}{p}$ of all integers are N-prime where the product is over all prime p less that or equal to N.

The returns are diminishing, but about .88 of all integers are 100-prime. So if we check n for factors upto 100 and it has none, then, properly formulated, 'it is about 6 times more likely of being prime than if we had not checked'.

This test is clearly not enough on its own. But it is useful in any algorithm for PRIME to first check for small factors.

3.4.2 The Euler Formula Test (or the Fermat Test)

Recalling that by Fermat's Little Theorem, if p is prime then

$$c^{p-1} \equiv_p 1$$

for any c with $c \not\equiv_P p$, we can calculate $c^{n-1} \mod n$ for several random c and be pretty sure whether or not n is prime. Here is some Sage code

```
def euler_is_prime(n):
   flag = 1; i = 1; M = Integers(n);
```

```
while (flag=1)*(i<10):
   i = i+1;
   if M(M.random_element()^(n-1)) <> M(1):
     flag=0
return(flag)
```

This is much better than just checking small primes. But as mentioned before, there are 'false primes' by this test. If c is a value such that $c^{n-1} \not\equiv_n 1$, then c is a *euler-witness* to the fact that n is composite. A *Carmichael* number is a composite integer n for which the only euler-witnesses are c that share a factor with n.

Problem 3.7. Use Sage to verify that 561 is a Carmichael number, and to find the next one.

Problem 3.8. Observe that $a^{561} \equiv_{561} a$ for **all** $a \in \mathbb{Z}_{561}$. Does a similar property hold for all Carmichael numbers? Which is better to use for a primality test: checking if $a^{N-1} \equiv_N 1$ or checking if $a^N \equiv_N = a$?

Carmichael numbers are rare, there are only 7 below 10000, and all of these have $3, 5$ or 7 as a factor. So a combination of the small-factors test and the Euler test is quite good. But still we want to do better.

3.4.3 The Miller Rabin Test

Problem 3.9. Show that if p is prime, then the only square roots of 1 in $\mathbb{Z}_p$ are 1 and -1. (Hint: you basically showed this in one of your exercise from Section 1.4 of the text.)

The Miller Rabin Test depends on the fact that for composite numbers N, there tend to be more solutions to $x^2 \equiv_N 1$.

Proposition 3.7. *Let p be a prime with $p - 1 = 2^d q$ for odd q, and let $a \in \mathbb{Z}_p^*$. Then either*

i). $a^q = 1$, *or*

ii). $a^{q2^i} = -1$ *for some* $i \in \{0, 1, \ldots, d-1\}$.

Proof. Indeed, if p is prime then $a^{q2^d} = a^{p-1} \equiv_p 1$. Taking square roots in Z_p, we compute

$$a^{2^d q}, a^{2^{d-1} q}, \ldots a^{2q}, a^q.$$

As p is prime, the first number that isn't a 1 must be a -1. □

This gives another primality check algorithm for n. Where $n-1 = 2^d q$ for odd q, a number $c \in \mathbb{Z}_n$ is a *Miller Rabin witness* if there is some $i \in \{1, \ldots, d-1\}$ such that $c^{2^i q} \equiv_n 1$ and $c^{2^{i-1} q} \not\equiv_n \pm 1$. The Miller-Rabin test picks some number C different random numbers $c \in \mathbb{Z}_n$ and checks if they are Miller Rabin witnesses. If any one of them is, it stops and says 'No': n is composite. If none of them are, it says 'Yes': n is prime. Depending on the C we use, we are more or less confident in a 'Yes' result.

The Miller-Rabin Test to check if n is prime

i). Find d and q such that q is odd and $n-1 = 2^d q$.

ii). Set $j = 1$. Until $j = C+1$ do:

(a) Choose random $c_j \in \mathbb{Z}_n$, and set $i = 0$.

(b) Compute $a_0 = c_j^q \mod n$. If $a_0 = \pm 1$, then let $j = j+1$ and exit inner loop. Otherwise continue.

(c) Until $a_i = \pm 1$, let $i = i+1$ and compute $a_i = (a_{i-1})^2 = c^{2^i q} \mod n$.

(d) If $a_i = 1$ then n is composite, exit algorithm and return 'NO', otherwise $a_i = -1$; let $j = j+1$ and repeat loop.

iii). As $j = C+1$, exit and return 'Yes'.

Problem 3.10. Use this algorithm with $C = 3$ to decide 81 and 89 are prime.

Now the questions arises: 'What should C be? How many a should we try before we conclude that n is prime?' and 'What are the chances of falsely concluding that n is prime?'

For $n = 561$, all but 8 of the numbers in $\mathbb{Z}_{561}$ are Miller-Rabin witnesses. The following can be shown, with a bit of work, after showing that the set of Miller-Rabin non-witnesses are a subgroup of $\mathbb{Z}_n^*$.

Theorem 3.8. *If n is an odd composite number, then at least $3/4$ of the integers in $\mathbb{Z}_n$ are Miller-Rabin witnesses.*

Thus the probability that a randomly chosen number is a MR-witness is at least .75. If we choose 3 random numbers the chance we get a witness is $1 - .25^3$ is over 98%. In practice, one tries about 50 different values of a. The chance of falsely concluding that n is prime is less than 10^{-30}. If we test a different n every second, we get a false prime about once every 10^{23} years.

Problem 3.11. Use Proposition 3.7 to show that 561 is composite.

Note

There are now tests that solve PRIME deterministically in time $O(k^6)$. But these are hard. Apparently, Miller Rabin can be made to run deterministically in time $O(k^4)$, but the prove of this requires the truth of the generalized Riemann hypothesis.

3.4.4 The Distribution of Primes

For any $x \geq 0$ let

$$\pi(x) = \#\text{ of primes between 1 and } x.$$

The following is quite to difficult.

Theorem 3.9 (The Prime Number Theorem)**.**

$$\lim_{x \to \infty} \frac{\pi(x)/x}{1/\ln(x)} = 1$$

This tells us that as x gets big, about $1/\ln(x)$ of the integers upto x are prime.

Problem 3.12. Show that about .1% of 100 digit numbers are prime.

With high probablitiy, the following algorithm gets us a d digit prime in linearly (in d) many primality checks.

i). Choose a random d digit number.

ii). Check if it is prime, if it is, stop, if not repeat.

Indeed, we have to choose about $\log(10^d) = d\log(10) \approx 2.3d$ random d-digit numbers on average, before we find one that is prime. If we choose $10d$, we have with high probablity, chosen one that is prime. (A more rigourous statement takes more definitions from probablility theory.)

Problem 3.13. Assuming that we must choose $10d$ different d-digit numbers to find one that is prime, and that we use Miller-Rabin with $C = 100$ to test if it is prime, what is the running time, in k, to find a k-bit prime number?

Still, one might want to reduce the constants in this running time calculation. One way to do this is to pick a random N-prime number for some large N.

Problem 3.14. Show that where $N = 2 \cdot 3 \cdot 5$ and n is 5-prime, the number $rN + n$ is 5-prime for every random r, and that every 5-prime number can be written uniquely in this form, with $0 < n < N$.

Problem 3.15. We saw that 4/15 of all integers are 5-prime. Use the Prime Number Theorem to show that about $\frac{15/4}{\ln x}$ of the 5-prime integers between 1 and x are prime.

Problems from the Text

P. 179: 3.13 (a,b,c), 3.14, 3.17

3.5 Pollards $p-1$ factorisation

Problems from the Text

P. 181: 3.21. 3.22 (a-e)

Pollard's algorithm quickly factors $N = pq$ in the case that $p-1$ or $q-1$ factors into small primes.

If L is an integer such that

$$(p-1)|L \text{ and } (q-1) \not| L$$

then by Fermat's little theorem, we have that

$$a^L \equiv_p 1 \text{ but (usually) } a^L \not\equiv_q 1$$

and so

$$p \mid a^L - 1 \text{ but (usually) } q \not| a^L - 1$$

giving that $\gcd(a^L - 1, N) = p$.

So how do we find this magic L such that $(p-1)|L$ and $(q-1) \not| L$. Assuming, wlog, that the largest factor n of $p-1$ is less than that of $q-1$, then $L = n!$ will do (or maybe $(ne)!$ where $\text{ord}_n(p-1) = e$).

Pollard's Algorithm to factor $N = pq$ is as follows.

i). Let $n = 2$ (or start a bit higher) and $A = 2^{n!}$.

ii). While n is not too big do:

(a) Let $n := n + 1$.

(b) Let $A := A^n = 2^{n!}$

(c) If $\gcd(A - 1, N)$ is anything but 1 or N, then this is a factor, stop, if not, repeat.

Problem 3.16. Factor $N = 10403 (= 101 \cdot 103)$ using Pollard's Algorithm.

By Stirlings approximation, $n! \sim (n/e)^n$ so by fast powering computing $2^{n!} - 1$ takes about $O(2^k \cdot k)$ multiplications. We will only be able to do this if p has small factors.

3.6 Factorisation Via Difference of Squares

The main approaches to factorisation today are apparently variations on the following idea.

If $N = pq$ is a product of large primes $p > q$, then as p and q are both odd, $p - q = 2b$. Letting $a = p - b = q + b$ we get that

$$a^2 - b^2 = (a + b)(a - b) = pq = N.$$

So factoring N comes down to finding a non-trivial $(a, b \neq 0)$ such that $a^2 - b^2 = N$, as then $p = a + b$ and $q = a - b$. Such a and b of course satisfy

$$a^2 \equiv_N b^2;$$

so we find a and b satisfying this. If a and b satisfy $a^2 \equiv_N b^2$, we do not necessarily have $a^2 - b^2 = N$, so we do not necessarily have $N = (a - b)(a + b)$, but we do have $a^2 - b^2 = kN$ for some constant k. This will be almost as good. Before we look at solving $a^2 \equiv_N b^2$, lets see how to use this idea for more general factorisations.

For an integer N, let a and b be such that $a^2 \equiv_N b^2$. So for some integer k we have

$$kN = a^2 - b^2 = (a - b)(a + b).$$

The factors of N tend to be split among $(a - b)$ and $(a + b)$, so computing

$$F = \gcd(N, a - b)$$

gives a factor F of N. Dividing out of N we get $N = F \cdot N_1$. We can then repeat with F and N_1.

Example 3.10. When $N = 914387$ we find that $164255^2 \equiv_N 9835^2$. Computing

$$\gcd(N, 164255 - 9835) = 1103$$

gives a factor 1103 of N. Dividing, we get $N = 1103 \cdot 829$. Checking that these factors are prime, we stop.

Formally, to find a non-trivial factor of N, that is, one that is neither 1 or N, we do the following Non-Trivial Factor algorithm.

i). Find several solutions (a_i, b_i) to $a^2 \equiv_N b^2$

ii). Compute $g_i = \gcd(N, a_i - b_i)$ until we find a g_i different from 1 and N.

iii). Factor N into g_i and N/g_i.

About the word 'several' in (i). We call a solution (a_i, b_i) such that $g_i = \gcd(N, a_i - b_i)$ is not 1 or N a 'good' solution. It can be shown that about half of the solutions of $a^2 \equiv_N b^2$ are good, so we usually only need a handful of random solutions to get a good one: we can take several as '10'.

Problem 3.17. Using the small factor test, we can assume that N has no factors of b bits or smaller. Doing this, show that to factor N one only has to use the use the Non-Trivial Factor algorithm $O(k/b)$ times.

Clearly if $N = pq$ for primes p and q, then the Non-Trivial Factor algorithm factors N, after only one iteration.

In the Non-Trivial Factor algorithm, everything can be done in reasonable time, except for finding a solution to $a^2 \equiv_N b^2$. This is a bit hard, but it is easier than solving $pq = N$. The main reason that it is easier is that a and b tend to have smaller factors than a well chosen N does.

Finding solutions to $a^2 \equiv_N b^2$

So how do we do this. We start with a somewhat vague example. We will factor $N = 914387$. There are three steps.

Determining some constants

For this N we compute $L = L(N) = e^{\sqrt{2 \ln N \cdot \ln \ln N}} \approx 404$ and $B = B(N) = L(N)^{1/\sqrt{2}} \approx 70$. These parameters will be optimal, but difficult to continue our example with, so instead we use smaller $B = 11$ and, to compensate, larger $L = 20000$. Finally, c is a constant such that the algorithm has a $1/2^c$ chance of failing. We take it to be $c = 10$, to be reasonably sure that we will succeed.

Building Relations

A number is *B-smooth* if its prime factors are at most B. We will use a sieve (explained later) of the integers from $\sqrt{N} + 1$ to $\sqrt{N} + L$ to find at least $\pi(B) + c$ numbers in $\mathbb{F}_N$ whose square in $\mathbb{F}_N$ is B-smooth.

For $N = 914387$ we find all numbers between 1000 and 21000 whose squares are 11-smooth. These are called relations.

a	factorisation of $a^2 \mod N$
1869	$2^4 \cdot 3 \cdot 5^6$
1909	$2^{14} \cdot 5 \cdot 11$
3387	$3 \cdot 5^3 \cdot 11^3$
4586	$3^2 \cdot 5 \cdot 11$
5023	$2^7 \cdot 5 \cdot 7 \cdot 11^2$
$\vdots$	

It is useful that the exponents in the factorisation are not all even, this is why we start looking at $a > \sqrt{N} \approx 1000$. We stopped after finding 5 relations, because this is enough to continue our example, but really we would have gone all the way up to about 20000 so that we could be sure of getting the $\pi(B)+10 = 15$ relations that we wanted.

Elimination

We now look among the relations we built in the previous step to find solutions to $a^2 \equiv_N b^2$. To do so we find combinations of the relations such that when multiplied, the powers of each prime in their decompositions combine to be even. When there are only 5 relations, this is easy to do by inspection:

$$\begin{aligned} 9835^2 &= \\ 1869^2 \cdot 1909^2 \cdot 3387^2 &\equiv_N 2^{18} \cdot 3^2 \cdot 5^{10} \cdot 11^4 \\ &= (2^9 \cdot 3 \cdot 5^5 \cdot 11^2)^2 \equiv_N 164255^2 \end{aligned}$$

Problem 3.18. Find two more solutions to $a^2 \equiv_N b^2$.

The determination of the constants is a bit beyond our scope. They are chosen to minimise the number $L(N)$ that we have to check to get around $\pi(B)$ B-smooth numbers. We look a bit more closely at the other two steps.

3.6.1 Building Relations

Recall the Prime sieve of Erathosthenes, to find all primes upto n:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\boxed{2}$	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~
$\boxed{2}$	$\boxed{3}$	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~

To sieve upto 16, we only have to sieve out factors of upto $\sqrt{16} = 4$. This is, of course, a slow way to get a prime number: it takes $O(n^{3/2})$ time to sieve upto n. (This is exponential in the bits k of n, recall).

The sieve can be altered to get B-smooth numbers. When we sieve a number such as 2, we divide it out of its multiples instead of crossing them out.

n	2	3	2^2	2^3	3^2	$5-$smooth ?
2	$\boxed{1}$					✓
3	3	$\boxed{1}$				✓
4	$\underline{2}$	2	$\boxed{1}$			✓
5	5	5	5	5	5	
6	$\underline{3}$	$\boxed{1}$				✓
7	7	7	7	7	7	
8	$\underline{4}$	4	$\underline{2}$	$\boxed{1}$		✓
9	9	$\underline{3}$	3	3	$\boxed{1}$	✓
10	$\underline{5}$	5	5	5	5	
11	11	11	11	11	11	
12	$\underline{6}$	$\underline{2}$	$\boxed{1}$			✓
13	13	13	13	13	13	
14	$\underline{7}$	7	7	7	7	
15	15	$\underline{5}$	5	5	5	
16	$\underline{8}$	8	$\underline{4}$	$\underline{2}$	2	...✓

It is not enough to simply sieve upto B, we must sieve powers of B upto n. So to find B-smooth numbers upto n we sieve $O(\pi(B) \cdot \log_2 n)$ prime powers, taking time $O(n \cdot \pi(B) \cdot \log_2 n)$ divisions. But this is assuming that when we process say 2^3 we have to process all n numbers.

We simply do this to the squares of the numbers we are interested in.

a	a^2 mod 3877	2	3	2^2	5
63	92	46		23	
64	219		73		
65	348	174	58	28	
66	479				
67	612	306	102	51	
68	747		249		
69	884	442		221	
70	1023		341		
71	1164	582	194	97	
72	1307				
73	1452	726	242	121	
74	1599		533		
75	1748	874		437	

As we continue, we notice that 5 didn't sieve anything, so neither will 5^2, so we are finished with powers of 5. Further, as we sieve the squares of numbers from $\sqrt{N}$ to $\sqrt{(N)} + L < \sqrt{2N}$, we can predict exactly which numbers will

have to be factored by a given prime, and so have to divide only those numbers. With some work, one can show that there are only $O(L)$ divisions so overall time $O(2^{c\sqrt{k}} \cdot k^2) = O(2^{c\sqrt{k}})$ for some constant c. This is subexponential, but superpolynomial.

Problem 3.19. When sieving modulo 232153 the squares of $n = 490$ to 600, determine for exactly which n we divide out by 2, in the 2^3 column.

Problem 3.20. For which numbers a between 200 and 400 are $a^2 \mod 3171$ 11-smooth?

3.6.2 Elimination

We had

a	factorisation of $a^2 \mod N$
1869	$2^4 \cdot 3 \cdot 5^6$
1909	$2^{14} \cdot 5 \cdot 11$
3387	$3 \cdot 5^3 \cdot 11^3$
4586	$3^2 \cdot 5 \cdot 11$
5023	$2^7 \cdot 5 \cdot 7 \cdot 11^2$
$\vdots$	

for $N = 914387$ and had to find that

$$1869^2 \cdot 1909^2 \cdot 3387^2 \equiv_N 2^{18} \cdot 3^2 \cdot 5^{10} \cdot 11^4.$$

We do this by Gaussian elimination. Recall that $m = \pi(B)$ is the number of primes upto and including B. For each number a such that

$$a^2 = p_1^{e_1} \cdot p_2^{e_2} \cdots p_m^{e_m}$$

is B-smooth, we make a vector

$$v_a = (e_1 \mod 2, e_2 \mod 2, \ldots, e_m \mod 2)$$

over $\mathbb{F}_2$.

So we have

a	factorisation of $a^2 \mod N$	vector v_a of exponents $\mod 2$
1869	$2^4 \cdot 3^1 \cdot 5^6 \cdot 7^0 \cdot 11^0$	$(0,1,0,0,0)$
1909	$2^{14} \cdot 5 \cdot 11$	$(0,0,1,0,1)$
3387	$3 \cdot 5^3 \cdot 11^3$	$(0,1,1,0,1)$
4586	$3^2 \cdot 5 \cdot 11$	$(0,0,1,0,1)$
5023	$2^7 \cdot 5 \cdot 7 \cdot 11^2$	$(1,0,1,1,0)$
$\vdots$		

So what we are looking for is a soluton to the matrix equation

$$[v_{a_1} \mid v_{a_2} \mid \cdots \mid v_{a_n}]\, x = \vec{0} \mod 2.$$

That is, we want a solution to

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

We made sure that $n > m$ and so a solution exists. We find it using Gaussian elimination: using row switches, and adding rows to other rows, we push all of the 1's to the upper-right triangle of the matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \to \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \to \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Recall that the above matrix equation has exactly the same solutions as

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Recall that the first 1 occuring in a row of this last matrix is called a *pivot*. The corresponding variables, x_1, x_2 and x_5 in this case, are *dependent* variables. The others, x_3 and x_4, are *free* variables. We can choose any value $a_3 \in \{0, 1\}$ for x_3 and any value $a_4 = \{0, 1\}$ for x_4, and the values of the dependent variables are then determined.

So the solutions are

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = a_4 \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} + a_3 \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}.$$

Problem 3.21. Translate these into solutions to the equation $A^2 \equiv_N B^2$.

Problem 3.22. Use Gaussian Elimination over $\mathbb{F}_2$ solve

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Problem 3.23. Use the numbers you found in Problem 3.20 to find a solution to $a^2 \equiv_{3171} b^2$. Does this factor 3171?

3.6.3 All together

So to factor $N = pq$ we

i). Compute $L(N) = e^{\sqrt{\ln N \cdot \ln \ln N}}$ and $B(N) = L(N)$.

ii). Use the quadradic sieve of the squares of number a from $\lceil \sqrt{N} \rceil$ to about $\lceil \sqrt{N} \rceil + L(N)$ to 'build relations'.

iii). Use Gaussian elimation over $\mathbb{F}_2$ to find solutions to $a^2 \equiv_N b^2$

iv). Compute $g = \gcd(N, a-b)$ for these solutions until we get g different from N or $a - b$.

Problems from the Text

P. 182: 3.23(a), 3.24(a), 3.27, 3.33

3.9 Quadratic Residues and Reciprocity

We have seen that $x^2 \equiv_p 1$ for odd prime p has 2 solutions. More generally, where g is a generator of $\mathbb{Z}_p$ we have that

$$x^2 \equiv_p g^d$$

has 2 solutions if d is even and none if d is odd. In this section we look at determining if

$$x^2 \equiv_b a$$

has a solution for any non-prime b.

If $x^2 \equiv_b a$ has a solution, then a is a *quadratic residue (QR)* module a; if it has no solution, then a is a *quadratic non-residue (NR).*

As g^d is a QR modulo an odd prime p if and only if d is even, we have the following.

Proposition 3.11. *Let p be an odd prime.*

i). The product of two QR, or of two NR, modulo p, is a QR modulo p.

ii). The product of a QR and a NR is a NR mod p.

The *Legendre symbol* $\left(\frac{a}{b}\right)$ is defined

$$\left(\frac{a}{b}\right) = \begin{cases} 1 & \text{if a is a QR mod } p \\ -1 & \text{if a is a NR} \\ 0 & \text{if } p|a. \end{cases}$$

With this, Proposition 3.11 can be stated:

$$\left(\frac{a}{p}\right)\left(\frac{b}{p}\right) = \left(\frac{ab}{p}\right).$$

Clearly we also have that $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$ if $a \equiv_p b$. We won't prove the following surprising theorem, but it's true.

Theorem 3.12. *[Quadratic Reciprocity] Let p and q be odd primes.*

i). $\left(\frac{-1}{p}\right) = \begin{cases} 1 & \text{if } p \equiv_4 1 \\ -1 & \text{if } p \equiv_4 3 \end{cases}$

ii). $\left(\frac{2}{p}\right) = \begin{cases} 1 & \text{if } p \equiv_8 1, 7 \\ -1 & \text{if } p \equiv_8 3, 5 \end{cases}$

iii). $\left(\frac{p}{q}\right) = \begin{cases} \left(\frac{q}{p}\right) & \text{if } p \equiv_4 1 \text{ or } q \equiv_4 1 \\ -\left(\frac{q}{p}\right) & \text{if } p \equiv_4 q \equiv_4 3 \end{cases}$

We can use this to determine if a is a QR mod p.

Example 3.13. As

$$\begin{aligned} \left(\frac{28}{31}\right) &= \left(\frac{2}{31}\right)^2 \left(\frac{7}{31}\right) = \left(\frac{7}{31}\right) \\ &= \left(\frac{31}{7}\right) = \left(\frac{3}{7}\right) = \left(\frac{7}{3}\right) = \left(\frac{1}{3}\right) \\ &= 1 \end{aligned}$$

we have that 28 is a QR modulo 31.

Problem 3.24. Is 40 a QR modulo 71? Is 77 a QR modulo 123?

Nice! But we had to factor the numerator to do this, and factoring is hard, so not nice! The Jacobi symbol allows us to avoid most factoring.

For any integer a and any positive odd integer b, define the *Jacobi symbol* $\left(\frac{a}{b}\right)$, from Legendre symbols by letting

$$\left(\frac{a}{b}\right) = \prod \left(\frac{a}{p_i}\right)^{e_i},$$

where $b = \prod p_i^{e_i}$ is the prime factorisation of b.

One can show that reciprocity still holds.

Theorem 3.14 (Quadratic Reciprocity)**.** *Let a and b be positive odd integers.*

i). $\left(\frac{-1}{b}\right) = \begin{cases} 1 & \text{if } b \equiv_4 1 \\ -1 & \text{if } b \equiv_4 3 \end{cases}$

ii). $\left(\frac{2}{b}\right) = \begin{cases} 1 & \text{if } b \equiv_8 1, 7 \\ -1 & \text{if } b \equiv_8 3, 5 \end{cases}$

iii). $\left(\frac{a}{b}\right) = \begin{cases} \left(\frac{a}{b}\right) & \text{if } b \equiv_4 1 \text{ or } a \equiv_4 1 \\ -\left(\frac{q}{p}\right) & \text{if } a \equiv_4 b \equiv_4 3 \end{cases}$

Problem 3.25. Show this, using Theorem 3.12.

Further, the multiplicative properties of the symbol still hold.

Proposition 3.15. *Let a and a' be integers and b and b' be positive odd integers.*

i). $\left(\frac{aa'}{b}\right) = \left(\frac{a}{b}\right)\left(\frac{a'}{b}\right)$ *and* $\left(\frac{a}{bb'}\right) = \left(\frac{a}{b}\right)\left(\frac{a}{b'}\right)$.

ii). If $a \equiv_b a'$ *then* $\left(\frac{a}{b}\right) = \left(\frac{a'}{b}\right)$.

You should be able to do the following without any factoring except removing factors of 2 from a number.

Problem 3.26. Show that $\left(\frac{40152}{50333}\right) = 1$

Does this tell us that 40152 is a QR mod 50333? Nope! 50333 isn't prime. If it were then this would be true. But 50333 isn't prime so the fact that $\left(\frac{40152}{50333}\right) = 1$ doesn't tell us anything about 40152.

Observe that if $b = pq$ for odd primes p, q

$$\left(\frac{a}{b}\right) = \left(\frac{a}{p}\right) \cdot \left(\frac{a}{q}\right)$$

is equal to 1 if either

i). $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = 1$ or

ii). $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = -1$.

In the first case, the solutions to $x^2 \equiv a$ modulo p and q lift to a solution modulo pq, so a is a QR mod b. In the second case though neither of these images have

a solution, so neither does $x^2 \equiv_b a$. We can compute $\left(\frac{a}{b}\right)$. If it is -1 it tells us it is that a is a NR modulo b but if it is 1, it tells us nothing.

In fact, it is hard to decide if a is a QR modulo b. This leads us to a cryptosystem.

Problem 3.27. What is the complexity of computing the Jacobi symbol?

Problems from the Text

P. 185: 3.36, 3.38, 3.39

3.10 Probabilistic Encryption and the Goldwasser-Micali cryptosystem

This cryptosystem is another that does not fit our Chapter 1 definition of a cryptosystem: it has only two codewords! 0 and 1. For a cryptosystem, this cannot be secure, because Eve can soon see what 0 and 1 encrypt to. But heres the trick: they (essentially) never encrypt to the same thing twice.

There are still a huge set of ciphertexts, partitioned into sets C_0 and C_1 of the same size, and a codeword $i \in \{0, 1\}$ is encrypted to a random element of C_i.

Goldwasser-Micali PKC:

i). Alice chooses private primes p and q, and common non-residue a with $\left(\frac{a}{p}\right) = -1 = \left(\frac{a}{q}\right)$. She computes $N = pq$.

ii). Alice publishes the public encryption key (N, a).

iii). Bob chooses private message $m \in \{0, 1\}$ by choosing a random r in $\mathbb{Z}_N$ and computes

$$\begin{cases} r^2 \mod N & \text{if } m = 0 \\ ar^2 \mod N & \text{if } m = 1, \end{cases}$$

which he sends to Alice.

iv). Alice computes $\left(\frac{c}{p}\right)$ and decodes c to

$$\begin{cases} 0 & \text{if } \left(\frac{c}{p}\right) = 1 \\ 1 & \text{if } \left(\frac{c}{p}\right) = -1. \end{cases}$$

When $m = 0$, $c = r^2 \mod N$ has image $(r \mod p)^2$ modulo p so is a q.r. and so $\left(\frac{c}{p}\right) = 1$, as needed for Alice to decrypt c back to 0. When $m = 1$, we

have that $c = ar^2 \mod N$ has image $(a \mod p)(r \mod p)^2$ modulo p, and a was chosen to be a NR modulo p, so $\left(\frac{c}{p}\right) = -1$, as needed.

Now Eve can compute $\left(\frac{c}{N}\right)$, but since $\left(\frac{c}{p}\right) = \left(\frac{c}{q}\right) = 1$ when $m = 0$ and $\left(\frac{c}{p}\right) = \left(\frac{c}{q}\right) = -1$ when $m = 1$ she always gets that $\left(\frac{c}{N}\right) = \left(\frac{c}{p}\right)\left(\frac{c}{q}\right) = 1$. So this gives her no information.

Note

This cryptosystem, as shown, is impractical for long messages. For each bit, we have to send a ciphertext of size N, which must be big so that N cannot be factored.

That is, this system has big *expansion*. There are other probabilistic schemes with smaller expansion.

For short messages it is good though. To send a 2-bit message with RSA, we still encode it into a size N codeword.

Problem 3.28. For primes $p = 7$, $q = 11$, find a common NR a.

Problem 3.29. Using your common NR a from the previous question in the public key $(77, a)$ for Goldwasser-Micali, encrypt the plaintext bits $m_1 = 0$, $m_2 = 1$ and $m_3 = 0$ into codewords c_1, c_2 and c_3 as Bob, and then decrypt them as Alice.

Problems from the Text

P. 186: 3.41

5 Elliptic Curves

5.1 Elliptic Curves

An elliptic curve is a curve of the form

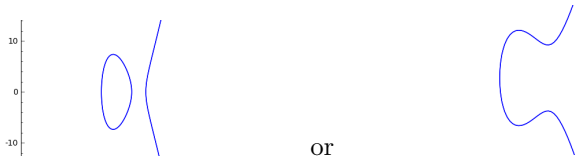
$$E : y^2 = x^3 + Ax + B$$

where $4A^3 + 27B^2 \neq 0$.

Problem 5.1. Show that $4A^3 + 27B^2 \neq 0$ ensures that the elliptic curve has three distinct roots.

Where $y = x^3 + Ax + B$ looks something like

the graph of $y^2 = x^3 + AX + B$ looks like



or

depending whether it has 1 or 3 real roots.

Let $E_{\mathcal{O}} = E \cup \mathcal{O}$ where we view $\mathcal{O}$ as a point at infinity. For two points P and Q on E let $L(PQ)$ be

- the vertical line through P if $Q = \mathcal{O}$,
- the line through P and Q if $P \neq Q \in E$,
- the tangent to E at P if $P = Q$.

Now as the intersection of $y^2 = x^3 + Ax + B$ with a line $y = mx + b$ is the zeros of the cubic equation

$$x^3 + Ax + B = m^2x^2 + 2mbx + b^2$$

and we know that either one or three of these are real, we have that any line that intersects E intesects in one or three points, viewing a tangency as two intersections. Vertical lines, which we cannot express as $y = mx + b$ intersect E in two points, but we consider them also to contain $\mathcal{O}$, so every line that

intersects $E_{\mathcal{O}}$ in at least two points, intersects it in three. Thus for any two points P and Q in $E_{\mathcal{O}}$ there is a third point $L(PQ)_3$ of $L(PQ)$ in E. Where $-P$ is the reflection of P in $y = 0$, let $P \oplus Q = -L(PQ)_3$.

Problem 5.2. Show that $\mathcal{O}$ is the identity element for the operation $\oplus$, and $-P$ is the inverse of P. Show that $\oplus$ is associative and commutative.

So $(E_{\mathcal{O}}, \oplus)$ is a commutative group.

Where $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are on E we find $P_1 \oplus P_2$ explicitly. First, assume that P_1 and P_2 are distinct points on E. Then $L = L(P_1, P_2)$ is

$$L(P_1, P_2) : y = \lambda x + \nu$$

where

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \qquad \text{and} \qquad \nu = y_1 - \lambda x_1.$$

The x-coordinates of $L \cap E$ are the solutions to $(\lambda x + \nu)^2 = x^3 + Ax + B$, which we can write as

$$0 = x^3 - \lambda^2 x^2 + (A - 2\lambda\nu)x + B - \nu^2.$$

We also know that this has solutions x_1, x_2, so (this is where we use that the descriminant is non-zero, and so the roots are distinct) is

$$0 = (x - x_1)(x - x_2)(x - x_3) = x^3 - (x_1 + x_2 + x_3)x^2 + \boxed{\text{lower order stuff}}$$

This yields that $(x_1 + x_2 + x_3) = \lambda^2$, and so

$$x_3 = \lambda^2 - x_1 - x_2.$$

Plugging this into the equation for L we get that

$$y_3 = \lambda x_3 + \nu = \lambda x_3 + (y_1 - \lambda x_1) = \lambda(x_3 - x_1) + y_1.$$

So

$$(x_1, y_1) \oplus (x_2, y_2) = (x_3, -y_3) = (\lambda^2 - x_1 - x_2, \lambda(x_1 - x_3) - y_1).$$

Now, if $P_1 = P_2$, then $L(P_1, P_2)$ is

$$L : y = \lambda x + \nu$$

where to find λ we use implicit differentiation:

$$2y\frac{\mathrm{d}y}{\mathrm{d}x} = 3x^2 + A$$

giving that

$$\lambda = \frac{\mathrm{d}y}{\mathrm{d}x} = \frac{3x^2 + A}{2y} = \frac{3x_1^2 + A}{2y_1}.$$

Again we have that $\nu = y_1 - \lambda x_1$ and so $x_2 = \lambda^2 - x_1 - x_2$ and $-y_3 = \lambda(x_3 - x_1) + y_1 = \lambda(x_1 - x_3) - y_1$.

Summarising:

- $P \oplus \mathcal{O} = P = \mathcal{O} \oplus P$
- $P \oplus -P = \mathcal{O}$
- $(x_1, y_1) \oplus (x_2, y_2) = (x_3, -y_3)$ where

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{if } P_1 \neq P_2 \\ \frac{3x_1^2 + A}{2y_1} & \text{if } P_1 = P_2 \end{cases}$$

and $x_3 = \lambda^2 - x_1 - x_2$ and $y_3 = \lambda(x_1 - x_3) - y_1$.

Problems from the Text

P.339 : 5.1, 5.2, 5.3

5.2 Elliptic Curves over Finite Fields

Now we consider the curve $E : y^2 = x^3 + Ax + B$ with non-zero descriminant ($4a^3 + 27b^2 \neq 0$) over a finite field $\mathbb{F}_p$ rather than over $\mathbb{R}$.

If $x^3 + Ax + B$ is a quadratic residue for some $x \in \mathbb{F}_p$ then it yields a (usually two) points (x, y) on the curve. As half of the elements of $\mathbb{F}_p$ are quadratic residues, it usually turns out, for given A and B, that about half of the values $x^3 + Ax + B$, as x runs over $\mathbb{F}_p$ are quadratic residues. So for given A and B, there are usually around p solutions to

$$E : y^2 = x^3 + Ax + B$$

in $\mathbb{F}_p$. All the arithmetic we did in the previous section to calculate $P \oplus Q$ carries over in $\mathbb{F}_p$, and so with the point $\mathcal{O}$ we again get a group $E(\mathbb{F}_p)$.

Example 5.1. Let $p = 17$, $A = 3$ and $B = 5$. We check that

$$4A^3 + 27B^2 = 4(3^3) + 27(5^2) \equiv_{17} 1 \neq 0$$

so that the group $E = E(\mathbb{F}_{17})$ is well-defined.

To find element of E we compute y^2 and $x^3 + 3x + 5$ for all y and x in $\mathbb{F}_{17}$.

y	y^2
1	1
2	4
3	9
4	16
5	8
6	2
7	15
8	13
9	13
10	15
⋮	⋮

x	x^3+3x+5
1	9
2	2
3	7
4	13
5	9
6	1
7	12
8	14
9	13
10	15
11	9
12	1
13	14
14	3
15	8
16	1

This yields 22 points, $(1, \pm 3), (2, \pm 6), (4, \pm 8) \ldots$. With $\mathcal{O}$, we have a group E of 23 elements.

Note

We predicted there would be about 17 elements, so there is some varience. Actually computing $|E(\mathbb{F}_p)|$ for given A and $B \in \mathbb{F}_p$ requres some work, but Schoof showed that it can be done efficiently in time $O(k^6)$.

Problem 5.3. Using the formula we computed in the previous section compute the following in $E(\mathbb{F}_{17})$ with $A = 3$ and $B = 5$.

i). $(5, 3) \oplus \mathcal{O}$

ii). $(1, 3) \oplus (1, 14)$

iii). $(11, 3) \oplus (9, 9)$

iv). $(11, 14) \oplus (11, 14)$

Above, we computed all the elements of $E(F_{17})$. For some big prime p, this will be untenable, but for our cryptosystems we won't need all elements of $E(\mathbb{F}_p)$, we will be happy with just a couple of them. As about half the values of x yield a quadratic residue $x^3 + Ax + B$, we will choose random x and check that $x^3 + Ax + B$ is a QR.

Example 5.2. To find a point in $E(\mathbb{F}_{23})$ where E is $y^2 = x^3 + 4x + 3$ we choose random x.

Guessing $x = 5$, we must check if $5^3 + 4(5) + 3 = 148 \equiv_{23} 10$ is a QR mod 23.

$$\left(\frac{10}{23}\right) = \left(\frac{2}{23}\right)\left(\frac{5}{23}\right) = \left(\frac{23}{5}\right) = \left(\frac{3}{5}\right) = \left(\frac{5}{3}\right) = \left(\frac{2}{3}\right) = -1.$$

So no! How about with $x = 8$?

Computing $8^3 + 4(8) + 3 \equiv_{23} (-5)8 + 4(8) + 3 \equiv -17 + 9 + 3 = 18$ and checking $\left(\frac{18}{23}\right) = \left(\frac{3}{23}\right)^2\left(\frac{2}{23}\right) = 1$, we find it is. Now to find a point in $E(\mathbb{F}_{23})$ we simply have to compute the square root of 18.

Ah, but darn. Square-roots are hard. Except that in the homework problem 3.38 of the text, we saw that in the case that $p \equiv_4 3$, square roots are easy to compute: if a is a QR, then $y = a^{\frac{p+1}{4}}$ is a square root.

So computing $18^{\frac{24}{4}} \equiv_{23} 18^6 \equiv 5^6 \equiv 2^3 = 8$ we get that $(8, 8)$ and $(8, -8)$ are elements of $E(\mathbb{F}_{23})$.

Problem 5.4. Can one avoid using the Legrange symbol in the case that $p \equiv 3$ mod 4 to check if n is a QR modulo p? Is this faster?

Problem 5.5. Where $A = 5$ and $B = 4$ find a point of $E(\mathbb{F}_{23})$. (Not $\mathcal{O}$, cheater!)

Problems from the Text

P.340: 5.5 , 5.6, 5.7

5.3 Elliptic Curve DLP

An elliptic curve over a finite field is a group, so we can talk of the DLP over an elliptic curve. We recall aspects of the DLP and observe where features of the DLP over elliptic curves differ from over $\mathbb{Z}_p$.

The general DLP is to find a solution x (an integer) to

$$h \equiv g^x$$

in a group G where h and g are element in G.

In $E(\mathbb{F}_p)$ the group action is written additively so the DLP asks for a solution x to

$$Q = xP,$$

where Q and P are in $E(\mathbb{F}_p)$. The solution x is the *elliptic discrete log of Q wrt P* and is denoted $\log_P(Q)$.

Naturally, the solution need not exist, and if it does it is an integer modulo the order d of $E(\mathbb{F}_p)$.

The brute force approch to the DLP requires powering, which, written additively, is computing nP for some integer n and $P \in E(\mathbb{F}_p)$. In this notation fast powering is called double-and-add.

Example 5.3. Where E is the curve $y^2 = x^3 + 4x + 3$ let P be the point $(1, 13)$ in $E(\mathbb{F}_{23})$. We compute $7P$. Indeed, using double-and-add we observe that $7 = 4 + 2 + 1 = 2^2 + 2^1 + 2^0$ so we will compute $P \oplus 2P \oplus 4P$.

First we compute $2P = P \oplus P = (1, 13) \oplus (1, 13) = \cdots$

- $\lambda = \frac{3(1)^2+4}{2(13)} \equiv 7/3 \equiv 7 \cdot 8 \equiv 10$
- $x_3 = \lambda^2 - x_1 - x_2 \equiv 98 \equiv 6$
- $-y_3 = y_1 + \lambda(x_3 - x_1) = 13 + 10(5) - 6$

So $2P = (6, 6)$.

Now we compute $4P = 2(6, 6) = \cdots = (1, 10)$.

Now note here that $4P = -P$ so $5P = \mathcal{O}$ telling us that P has order 5. Using this we can cheat a bit and write

$$7P = P \oplus 2P \oplus 4P = 2P = (6, 6)$$

Problem 5.6. Observe that in the group $E(\mathbb{F}_p)$ inverses are very easy to compute: the inverse of $(1, 13) = (1, -13)$. Use this to speed up the double-and-add algorithm.

5.3.1 Some Notes about DLP over $E(F_p)$

We saw that the element $P = (1, 13)$ in the above example had order 5. Naturally, for cryptosystems based on the DLP, we want to find P of large order. One can show that the group $E(\mathbb{F}_{23})$ that we used above actually has order 20 so is

$$\mathbb{Z}_4 \times \mathbb{Z}_5 \text{ or } \mathbb{Z}_2^2 \times \mathbb{Z}_5.$$

The maximum order of an element is then either 20 or 10 respectivley. Picking random elements, say $P = (7, 2)$ we can calculate

- $4P = (1, 10) \neq \mathcal{O}$ and
- $10P = (16, 0) \neq \mathcal{O}$

to verify that P is an element of order 20. There are algorithms for choosing p, A and B so that there exists an element of order d close to p. (It turns out you don't want exactly p though). Doing all of this is beyond our scope, but can be done relatively efficiently.

By the collision algorithms we saw earlier, the DLP for $E(\mathbb{F}_p)$ with respect to such an element of order d can be solved in time time $O(\sqrt{d})$. For the DLP over $\mathbb{F}_p$ we had methods to reduce this if $p-1$ factored into small primes. Such methods do not work for the DLP over $E(\mathbb{F}_p)$. This is one of the reasons that $E(\mathbb{F}_p)$ is useful.

Problems from the Text

P.340: 5.8, 5.9, 5.10

5.4 Elliptic Curve Cryptography

We look at the Diffe Hellman Key Exchange and the El-Gamal PKC using DLP in $E(\mathbb{F}_p)$ rather than $\mathbb{Z}_p$.

5.4.1 Diffe Hellman

Translating the Diffe Hellman Key Exchange to an elliptic curve version is trivial.

Recall that Diffe Hellman had

- Public: prime p, and primitive element $g \in \mathbb{F}_p^*$
- Alice's key: a
- Bob's key: b
- Alice sends Bob: $A = g^a \in F_p^*$
- Bob sends Alice: $B = g^b \in F_p^*$
- They secretly share $A^b = B^a = g^{ab}$

The elliptic curve version is

- Public: prime p, curve E, and point $P \in E(\mathbb{F}_p)$ of high order
- Alice's key: a
- Bob's key: b
- Alice sends Bob: $K_A = aP$ (We can't use A and B – we use these to describe E.)
- Bob sends Alice: $K_B = bP$

- They secretly share $S = bK_A = aK_B = abP$

Problem 5.7. For the Diffe-Hellman Key Exchange over $E(\mathbb{F}_p)$ where $p = 251$, $A = 57$ and $B = 24$, Alice and Bob agree on the public point $P = (69, 49)$. Alice chooses key $a = 67$ and Bob chooses $b = 97$. What is their shared secret point? (Use a computer!)

Problem 5.8. Show that if Alice and Bob only exchange x-values of their keys K_A and K_B they can still get a shared secret x-value.

This reduces the expansion of the cryptosystem.

5.4.2 El-Gamal

Recall that El-Gamal had

- Public: prime p, and primitive element $g \in \mathbb{F}_p^*$
- Alice chooses private exponent a and publishes public encryption key $A = g^a$
- Bob chooses private b and encrypts m to $(c_1, c_2) = (g^b, mA^b)$
- Alice decrypts (c_1, c_2) to $c_2/(c_1^a) = m$.

We adapt this to $E(\mathbb{F}_p)$ for prime p, so that Bob can send Alice a secret point $m \in E(\mathbb{F}_p)$.

- Public: prime p, curve $E(\mathbb{F}_p)$, and high order element P of $E(\mathbb{F}_p)$
- Alice choose private integer a and publishes $K_A = aP$.
- Bob chooses private integer b encrypts m to $(c_1, c_2) = (bP, m \oplus bK_A)$.
- Alice decrypts to

$$c_2 \ominus ac_1 = m \oplus (baP) \ominus abP = m$$

We write $P \ominus Q$ for $P \oplus -Q$.

Example 5.4. Where $p = 251$, $A = 57$ and $B = 24$ let $P = (197, 154) \in E(\mathbb{F}_p)$. (This P has order 239.)

Alice chooses private $a = 15$ and publishes $K_A = 15P = (36, 17)$.

To encrypt the message $m = (93, 190)$ Bob chooses private key $b = 10$ and computes

$$c_1 = 10P = (110, 79)$$
$$c_2 = m \oplus bK_A = (194, 130).$$

He sends Alice $(c_1, c_2) = ((110, 97), (194, 130))$.

Alice decrypts

$$(93, 190) \ominus a(110, 97) = (93, 190).$$

It is not enough for Bob to send Alice just the x-values of c_1 and c_2:

Problem 5.9. Show that $P \oplus Q$ and $P \ominus Q$ do not generally have the same x value.

However, it is enough that Bob send the x-values of c_1 and c_2 and the signs of the y-values.

Problem 5.10. How would Bob use this to send a secret message $M \in \mathbb{F}_p$ to Alice?

Problems from the Text

P.341: 5.13, 5.15, 5.16, 5.17

7 Digital Signatures

When we sign a document, we are certifying that "I have seen and approved this document." We do not want anybody else to be able to sign a document with our signature, but we want anybody to be able to verify that it was us who signed it. A digital signature does the same for a digital document.

The basic scheme is as follows.

- Alice has a private signing key K_{sign} and a public verification key K_{ver}.
- Alice signs a document D with a signature $S = K_{\text{sign}}(D)$.
- Bob can check that Alice signed D by computing $K_{\text{ver}}(S)$ and checking that it is equal to D.

Eve should not be able to forge (falsify) Alice's signature even if she has access to several different signed documents. Moreover signing and verification should be relatively quick.

In practice, the document D may be quite long so Alice signs a *hash* H of D rather than D itself. A *hash function* is a function

$$H : \cup_{i=1}^{\infty} \mathbb{F}_2^i \to \mathbb{F}_2^N$$

that maps any binary string to a N-bit binary string. Clearly we want that H is difficult to invert when restricted to $\mathbb{F}_2^i$ for any i.

Problem 7.1. Using the modular arithmetic we have seen in this class, make a good Hash function.

RSA gives a simple example of a digital signature.

7.1 RSA Digital Signatures

The RSA Digital Signature is as follows.

i). Alice chooses large primes p and q and computes $N = pq$ She chooses secret exponent s with that is invertible mod $n = (p-1)(q-1)$ and computes its inverse v.

ii). Alice publishes (N, v).

iii). Given a document (or a hash of a document) D in $\mathbb{Z}_N$ she computes a signature $S = D^s$ in $\mathbb{Z}_N$.

iv). To verify that Alice signed it, Bob checks that $S^v \equiv_N D$.

Problems from the Text

P. 458: 7.1, 7.2, 7.3

7.2 El Gamal Digital Signature

Alice makes a digital signature from the El Gamal Cryptosystem for prime p and generator g as follows.

i). She chooses a signing exponent a, computes $v = g^a$ in $\mathbb{Z}_p$, and publishes (a, v).

ii). To sign D she chooses another exponent e and computes

$$s_1 = g^e \mod p \text{ and } s_2 = (D - as_1)e^{-1} \mod p - 1.$$

iii). Bob can verify that it was her by computing

$$v^{s_1} \cdot s_1^{s_2} \equiv_p g^{as_1} \cdot g^{e\cdot(D-aS_1)e^{-1}} = g^d$$

and comparing it to g^D.

Certainly Eve can forge a signature if she gets a, which she can do by finding the discrete log $\log_g(v)$ modulo p. But there is another way.

To forge, she needs, given v and g^D, to find x and y such that

$$v^x \cdot x^y \equiv g^D \mod p.$$

Taking logs, this is

$$log_g(v)x + y \log_g(x) \equiv D \mod p - 1.$$

Again though, to find x and y, it seems that she must take discrete logs. (We cannot prove this though.)

Problems from the Text

P. 459: 7.4, 7.5, 7.6, 7.7

6 Lattices and Cryptography

6.2 The Knapsack Problem

Definition 6.1 (The Knapsack Problem (KP(n)))**.**
Input: A set $M = \{m_1, \ldots, m_n\}$ of n integers, and an integer S.
Output: The subset $M' \subset M$ such that $\sum M' = S$.

Problem 6.1. Is there a subset of $M = (4, 5, 7, 8, 12, 14, 18)$ that sums to 28? to 36?

Usually we write the set M as an increasing sequence $M = (m_1, m_2, \ldots, m_n)$; so $m_i < m_{i+1}$ for all i. For any subsequence M' of a given n term superincreasing sequence M, let $v_{M'} = (v_1, \ldots, v_n)$ be the vector in $\mathbb{F}_2^n$ with

$$v_i = \begin{cases} 0 & \text{if } m_i \notin M' \\ 1 & \text{if } m_i \in M' \end{cases} .$$

Then the sum of elements of M' can be written as the dot product

$$v_{M'} \cdot M = \sum_{i=1}^{n} v_i m_i.$$

Solving KP for M and S comes down to finding a solution x to $x \cdot M = S$.

The brute force algorithm would be to try computing $x \cdot M$ for every vector $x \in \mathbb{F}_2^n$. There are 2^n of these, so this is not a good algorithm.

This problem is well known to be NP-complete, which means that there is no known algorithm for it that has running time better than exponential in n. (And many of us believe that there will never be.) But we can do better than $O(2^n)$.

Consider the following collision algorithm.

i). Compute a list L_A of the values $x \cdot M$ for all vectors x that are 0 in the last $\lfloor n/2 \rfloor$ bits.

ii). Compute a list L_B of the values $S - y \cdot M$ for all vectors y that are 0 in the first $\lfloor n/2 \rfloor$ bits.

iii). Look for a common element $x \cdot M = S - y \cdot M$.

iv). Return solution $x + y$.

As with the collision algorithm for DLP our $O(2^n)$ is reduced to $O(2^{n/2})$.

Problem 6.2. Show that this algorithm works.

6.2.1 Superincreasing Sequences

An increasing sequence $M = (m_1, \ldots, m_n)$ is *superincreasing* if for all i

$$\sum_{j=1}^{i-1} m_j < m_i.$$

For a superincreasing sequence M the function

$$e : \mathbb{F}_2^n \to \mathbb{Z} : v \mapsto v \cdot M$$

is injective, and its inverse map d, solves the Knapsack Problem.

Problem 6.3. Show that if M is superincreasing, then e is injective.

Problem 6.4. The sequence $M = (1, 2, 5, 9, 20, 45, 100)$ is superincreasing. Decide if $S = 182$ is the sum of a subsequence of M.

Explain how to decide if some S is a sum of a subset of a superincreasing sequence M.

This allows us to use such a sequence for a cryptosystem, we simply have to disguise M.

The Merkle - Hellman Subset Sum Cryptosystem works as follows.

i). Alice chooses a secret superincreasing sequence $R = (r_1, \ldots, r_n)$ and relative prime integers A and B with $B > 2r_n$ and $A > B/2$.

ii). Alice computes $m_i = Ar_i \mod B$ for each i and publishes the set $M = (m_1, m_2, \ldots, m_n)$.

iii). Bob encrypts $m \in \mathbb{F}_2^n$ to $S = m \cdot M$.

iv). Alice decrypts S by calculating $S' = A^{-1}S$ and solving KP for set R and sum S'.

Problem 6.5. Show that the solution x to $x \cdot R = S'$ is indeed m.

Problem 6.6. Using the superincreasing sequence $R = (2, 5, 9, 20, 45, 100)$ we take $B = 203$ and $A = 177$. Computing $2 \cdot 177 \equiv_{203} 151$ etc, we publish the encryption key $M = (151, 73, 172, 89, 48, 39)$. Bob encrypts a meassage m to 85. Decrypt the message as Eve, and as Alice.

Problem 6.7. For added security, Alice reordered the encryption key to $M = (39, 48, 73, 89, 151, 173)$. How does she decrypt 85 now.

Problem 6.8. Find the n term superincreasing sequence with the smallest sum. How about if the first term is at least a?

For security reasons we don't want to take r_1 too small. To encrypt a n bit string, we take r_1 having n bits, and so B has about $2n$ bits. The encoded message is thus a $2n$ bit string, so our message expansion is about 2.

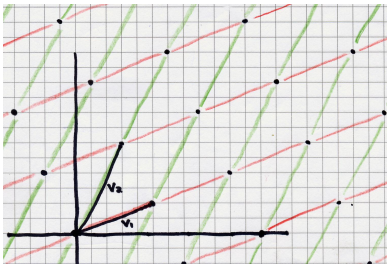
Problems from the Text

P. 422: 6.2, 6.3

6.3 Lattices and the Shortest Vector Problem

For a set $V = \{v_1, \ldots, v_n\}$ of vectors in $\mathbb{R}^m$, the *lattice L generated by V* is the set of linear combinations over $\mathbb{Z}$ of vectors in V. Viewing V as the rows of a matrix M_V, L is the set of vectors/points xM_v for vectors $x \in \mathbb{Z}^n$.

The lattice L_0 generated $v_1 = (5, 2)$ and $v_2 = (3, 6)$ looks like:



One of the basic problems concerned with lattices is the following.

Definition 6.2 (The Shortest Vector Problem SVP)**.** Given a lattice L find a non-zero vector $\ell \in L$ minimising $|\ell|$.

Solving SVP for the lattice L_0 above is simple: $\pm v_1$. This is because we have a nice generating set V for L_0.

A *basis* B of a lattice is any set of independent vectors that generates it. All bases of a lattice have the same number of elements, and this number is the *dimension* of the lattice.

Problem 6.9. Show that $B = (v_1, v_2) = ((5, 2), (3, 6))$ is a basis of L_0. Show that $B' = \{(17, 2), (22, 4)\}$ is another basis of L_0.

Finding the shortest vector in L_0 might have been more difficult if we had been given the basis B'.

But the Gaussian Lattice reduction algorithm works nicely finding a shortest vector. It does more, it gives a basis containing a shortest vector.

The idea is to replace the basis with a smaller one by removing the component of the shorter basis vector from the longer one. Recall that the projection

of a vector a onto a vector b can be computed, using dot products, by

$$\mathrm{proj}_b(a) = \frac{a \cdot b}{b \cdot b} b.$$

For a real number r, $\lfloor r \rceil$ is the integer closest to it. In Sage it is $round(r)$ so the following finds a reduction of the vector u by its integral v component:

```
def reduce(a,b):
  l = (a.dot_product(b)/b.norm()^2).n();
  return (a - round(l)*b)
```

The Gaussiam lattice reduction for a basis (a, b) of a 2-dimensional lattice L is as follows.

i). If $|b| > |a|$ switch a and b.

ii). Let $a' = reduce(a, b)$

iii). If $a' = a$ return (a', b), otherwise let $a = a'$ and repeat the loop.

Example 6.3. Let $a_1 = (284, 176)$ and $b_1 = (87, 54)$ be our basis vectors.

$$\begin{aligned}
a_2 &= a_1 - \left\lfloor \frac{a_1 \cdot b_1}{b_1 \cdot b_1} \right\rceil b_1 \\
&= (284, 176) - \left\lfloor \frac{(284, 176) \cdot (87, 54)}{(87, 54) \cdot (87, 54)} \right\rceil (87, 54) \\
&= (284, 176) - \left\lfloor \frac{34212}{10485} \right\rceil (87, 54) = (284, 176) - 3(87, 54) = (23, 14) \\
b_2 &= b_1 - \left\lfloor \frac{b_1 \cdot a_2}{a_2 \cdot a_2} \right\rceil a_2 = (-5, -2) \\
a_3 &= a_2 - \left\lfloor \frac{a_2 \cdot b_2}{b_2 \cdot b_2} \right\rceil b_2 = (-2, 4) \\
b_3 &= b_2 - \left\lfloor \frac{b_2 \cdot a_3}{a_3 \cdot a_3} \right\rceil a_3 = (-5, -2)
\end{aligned}$$

Done: $\{(2, 4), (5, 2)\}$.

One can show that this converges quite quickly in two dimensions, and so solves SVP in two dimensions. There are generalisations of the Gaussian lattice reduction algorithm that works quite well in higher dimensions, but gets slow in really high dimensions.

This is good, because the SVP solves the KP.

Problem 6.10. For an instance $M = (m_1, \ldots, m_n)$ and S of KP, consider the matrix A_M

$$\begin{bmatrix} 2 & 0 & 0 & \cdots & 0 & m_1 \\ 0 & 2 & 0 & \cdots & 0 & m_2 \\ 0 & 0 & 2 & \cdots & 0 & m_3 \\ \vdots & & & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 2 & m_n \\ 1 & 1 & 1 & \cdots & 1 & S \end{bmatrix}$$

For a solution x to $xM = S$, let $x' = [x \mid -1]$ be the vector we get by appending a -1 at the end. Show that $x'A_M = (\pm 1, \pm 1, \pm 1, \ldots, \pm 1, 0)$.

The length of $(\pm 1, \pm 1, \pm 1, \ldots, \pm 1, 0)$ is at most $\sqrt{n}$. As the m_i are large, of the order $O(2^n)$, other vectors in the lattice (except possibly those with a 0 in the last coordinate), tend to have length of the order $O(\sqrt{2}^n)$, which is much bigger than $\sqrt{n}$. So finding x comes down to solving SVP for the lattice generated by A_M.

Problems from the Text

P.430: 6.37

TEST COVERAGE STOPS HERE.

6.3.1 An upper bound on the size of the shortest vector

For a basis B of a lattice L, the *fundamental domain* $\mathcal{F}$ of B is the parallellopiped contained by the points xM_B for $x \in \mathbb{Z}_2^n$. Its volume is the absolute value of the determinant of M_B.

Problem 6.11. Find the volume of the fundamental domain of the lattice L_0.

Problem 6.12. Show that the volume of the fundamental domain is the same for any basis.

We may therefore define the determinant of a lattice L as $\det(L) = \det(M_B)$, the volume of its fundamental domain with respect to any basis B.

One see's Hadamards inequality by an obvious geometric argument: for any basis $B = \{b_1, \ldots, b_n\}$ of a lattice L,

$$\det(L) \leq \prod_{i=1}^{n} |b_i|.$$

Theorem 6.4 (Minkowski's Theorem)**.** *Let S be a symmetric convex region in R^n (say a sphere or a cube) with volume $\mathrm{Vol}(S) > 2^n \det L$, then S contains a non-zero vector of the lattice L.*

Proof. Let S be a symmetric convex space of volume greater than $2^n \mathrm{Vol}(L)$. Shrinking every point in S by a factor of 2, the volume of the result is greater than that of a fundamental domain of L, so there are two points a and b of S such that $a/2 = \ell_a + w$ and $b/2 = \ell_b + w$ for some lattice points ℓ_a and ℓ_b and some vector w. As S is symmetric $-b$ is in S.

But then the lattice point

$$\ell_a - \ell_b = \frac{a}{2} - \frac{b}{2} = \frac{a-b}{2}$$

is halfway between a and $-b$, so S being convex, it is in S. □

Problem 6.13. Use Minkowski's Theorem to show the for any lattice, the shortest non-zero vector x satisfies

$$|x| \leq \sqrt{n} \det L^{1/n}.$$