Chapter 4

Codes and Factoring

In this chapter, we will look at a code based on the number theoretic properties that we have developed. Since this code depends on the fact that it is a lot easier to find big primes than to factor large numbers, we will also study how this is done.

4.1. Codes

Codes are a way of **encrypting** a message so that it is very difficult or impossible to read the message unless you have knowledge of the **key** which *unlocks* the message. The most familiar codes are simple substitution codes. This means that each letter is replaced with another one. For example, consider the permutation of the alphabet given by

```
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
5936071842KSRIUFHQPOWELJGYTADZMVXNBC
```

A message like 'Houston airport, noon, Jan 22' would become

'QGMDZGJKPAYGAZJGGJOKJ33'.

This kind of code is very easy to break with the aid of a computer. In fact, with a longer message, it can be done by hand and is a popular pastime for many people. For example, this message has 5 G's, so one might think this is a vowel.

Actually, computers routinely use codes all the time—not for secrecy, but because computers can only store numbers (base 2). The ASCII code provides a number from 0 to 255 for all digits, upper and lower case letters, and many other symbols. This is how the computer can store text, and how word processors can manipulate it. The modern UTF-8 system extends ASCII and encodes over 1,000,000 characters containing all major alphabets in the world. A character uses up to 4 bytes (32 bits), so there are 2^{32} possibilities. Since there is extra room in this system, certain bits are used to detect and possibly correct errors. This is another important use of encryption that helps ensure accurate transmission of digital data.

It is much more difficult to break the code known as a 'one time pad'. The idea is to code your message by using another message known to the encoder and the intended recipient. First we need a simple way to combine two letters into one. Let us use a 36 letter alphabet consisting of 26 letters and 10 decimal digits. We can think of each letter as representing an element in $\mathbb{Z}_{36}$. That is.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

Then two letters can be combined by addition modulo 36. Let us use the message 'The quick brown fox jumped over the lazy dog' to encode our message 'Pizza 5:30 tonight'. Consider

P	I	Z	Z	A	5	3	0	T	O	N	I	G	H	T
T	H	E	Q	U	I	C	K	B	R	O	W	N	F	O
I	Z	D	P	4	N	F	K	4	F	B	E	3	6	H

To decode this, one needs to know the coding message. If this message is changed every time, for example using different pages of *War and Peace* each time, this is virtually impossible to break without stealing the code. However, if both the sender and the recipient always have their copy of *War and Peace* with them, it might be a giveaway.

One thing these two codes have in common with each other and most other codes is that encoding and decoding use the same information. Another kind of code has been invented which is of quite a different character. Known as public key cryptography, the interesting thing about these codes is that the method for encryption can be made public. For example, the code can be published in the *New York Times* or be listed on an electronic bulletin board. Anyone can send you a coded message. The important point is that knowing how to *encode* does not tell you how to *decode*!

4.2. The Rivest-Shamir-Adelman Scheme

The public key code that we will study was developed at MIT by Rivest, Shamir and Adelman. The key point that makes their code secure is that it is very easy to find large primes (say 200–300 decimal digits), but very difficult to factor large numbers that have a small number of large prime factors. The reason for this will be discussed in the next section.

Here is how it works. Pick two large primes p and q, with 200–300 digits. Set $n = pq$, and notice that $\varphi(n) = (p-1)(q-1)$. Now pick another number r (say 6–10 digits) which is relatively prime to $\varphi(n)$. Publish the two numbers (n, r).

Anyone who wishes to send you a message does the following. First turn your message into a number M by some standard scheme such as ASCII

or any simple scheme that encodes the 36 characters 0–9 and A–Z as a two digit number, possibly also including a–z and some punctuation marks. If necessary, split your message into blocks so the numbers encoded are all less than n. The coded message is

$$C \equiv M^r \pmod{n}.$$

This message can now be published in the personals section of the *New York Times*, or posted somewhere online.

The presumption is that all interested parties know the method of encoding and the message sent. Nevertheless, it is secure! Only you can break the code. To do this, you must know p and q. And you must know the Chinese Remainder Theorem. First, solve the equation

$$rs \equiv 1 \pmod{\varphi(n)}.$$

This has a unique solution by Lemma 2.3.3. Of course, to find s it is necessary to know $\varphi(n)$, and to find it, one must factor n. The key is Euler's Theorem, which tells us that $M^{\varphi(n)} \equiv 1 \pmod{n}$ when $\gcd(M, n) = 1$. In fact, since n is the product of two distinct primes, it turns out that our decoding method works for every M in the interval $0 < M < n$. Since $rs \equiv 1 \pmod{\varphi(n)}$, it can be written as $rs = 1 + k\varphi(n)$. Now compute $C^s \pmod{n}$ using the Chinese Remainder Theorem.

$$C^s \equiv M^{rs} \equiv M^1 (M^{p-1})^{k(q-1)} \equiv M \pmod{p}$$
$$C^s \equiv M^{rs} \equiv M^1 (M^{q-1})^{k(p-1)} \equiv M \pmod{q}$$

By the Chinese Remainder Theorem, $C^s \equiv M \pmod{n}$. Of course this only finds M up to a multiple of n. That is why we begin with a message such that $0 < M < n$.

If you have access to a symbolic manipulation software, try to design your own codes. Exchange messages with a friend, and decode them. Try using these same programs to break your friend's code.

The message is as secure as the difficulty of factoring large numbers (not practical) and the security of the storage location of the key s. (It is not necessary to remember p and q.) The latter consideration does not have anything to do with coding though. Of course, if everyone knows your encoding procedure, what prevents them from sending you a message and signing another name? How can you be sure that the message is really from your friend? The trick is for the sender to use his own code to give the message a **signature**.

It works like this: suppose your code is (n, r) and the sender has a published code (N, R). Let us also suppose that $N < n$. Only the sender knows the decoding key S for the (N, R) code. The sender computes

$$Q \equiv M^S \pmod{N} \quad \text{and} \quad 0 \leq Q < N.$$

Then this is encoded by the (n, r) code by

$$C \equiv Q^r \pmod{n}.$$

Again C is sent. To decode, you compute

$$Q \equiv C^s \pmod{n}.$$

Fortunately, since $N < n$, we know that $0 < Q < n$ without any ambiguity. Now using the sender's published code, compute

$$M \equiv Q^R \pmod{N}.$$

This message must be from our friend because only he/she knows S which enabled the encoding in the first place.

What happens if $N > n$? Try it out on a computer. You will end up with garbage. In this case you must encode using n first:

$$Q \equiv M^r \pmod{n}$$
$$C \equiv Q^S \pmod{N}.$$

This is decoded in the same basic way.

We end this section by discussing two practical aspects of the Rivest-Shamir-Adelman scheme. First, the encryption scheme involves raising M to a potentially large power r. If one computes M^r by naively multiplying M with itself r times, this requires r computations, which is a large number. Instead, the way one performs this computation in practice is expand r in base 2, namely write

$$r = a_0 + 2a_1 + 4a_2 + \ldots + 2^k a_k$$

where each $a_i \in \{0, 1\}$. Then, by repeatedly squaring, one computes M, M^2, $M^4 = (M^2)^2$, $M^8 = (M^4)^2$, ..., $M^{2^k} = (M^{2^{k-1}})^2$ mod n. One then computes the product

$$M^r \equiv M^{a_0}(M^2)^{a_1} \ldots (M^{2^k})^{a_k} \pmod{n}.$$

In total, this requires very few computations. To obtain all powers M^{2^r} for $r \leq s$ involves taking $k - 1$ products, and then obtaining M^r involves $a_0 + \cdots + a_k - 1 \leq k$ products. Thus, this is on the order of $2k \approx 2\log_2(r)$ computations, which is substantially faster than performing r computations.

Second, the Rivest-Shamir-Adelman scheme relies on choosing $n = pq$ with p and q large primes, which raises the question of how one obtains large primes in practice. At the beginning of Section 1.4, we mentioned the famous Prime Number Theorem, which asserts that for large N, there are roughly $\frac{N}{\log(N)}$ prime numbers less than or equal to N. Said differently, if we fix a large number N, the probability that a randomly chosen number $m \in \{1, \ldots, N\}$ is prime is roughly $\frac{1}{\log(N)}$. Therefore, if we choose $\log(N)$ numbers in $\{1, \ldots, N\}$, there is a good chance that at least one of them is prime. The chances are much higher if you avoid multiples of small primes. In practice, one can test primality using the deterministic algorithm by Agrawal-Kayal-Saxena developed in 2004 or the older Miller-Rabin probabilistic test. Notice that even if N is a large number with 500 digits, $\log(N)$

is only about 1000, so finding large primes with this method is quite practical for a computer.

Exercises

1. Show that s works as a key to decode the RSA encoded message provided that

$$rs \equiv 1 \pmod{\operatorname{lcm}(p-1, q-1)}.$$

2. Use computer software to check that $r = 42385687$ and a number 2 lines long:

$n = 91875320684918502380129870007406274898925429401183797214111268335816454459464037326759995364752417$

has a decoder

$s = 56970378770327971563435212231376282085305478725834255953360930453245246857516891597701705638306003$

3. Use computer software to choose two primes p, q with 40–45 decimal digits, and construct an RSA code.

4. Exchange messages with a friend. Code your student id number or a simple message with your code 'signature', then code it up using your friend's code.

5. Try to break your friend's code.

4.3. Primality Testing

How do you tell if a large number is composite or prime without doing a lot of trial divisions? It turns out that you may be able to show that a number is composite without knowing any factors! In 2004, Agrawal-Kayal-Saxena gave a groundbreaking efficient algorithm to determine if a number is prime. Their algorithm uses the fact that if $n \geq 2$ and $\gcd(a, n) = 1$, then n is prime if and only if $(X + a)^n \equiv X^n + a^n \pmod{n}$, see Exercise 1. Checking this particular congruence is not efficient. However they modify it in a way that makes the problem tractable: if $(X + a)^n \equiv X^n + a^n \pmod{n}$ holds, then it is also true that for all r, there are polynomials $f(X), g(X)$ such that

$$(X + a)^n = (X^n + a^n) + (X^r - 1)g(x) + nf(X). \tag{4.3.1}$$

Indeed, we can simply take $g = 0$ and f an appropriate polynomial. Agrawal-Kayal-Saxena show a converse result: they prove that if there exists r and a set of a such that if (4.3.1) holds for some f and g, then n is a prime power. Moreover, they make these choices in such a way that this equation can be checked efficiently. Although the details of their algorithm are beyond the scope of this course, in this section we highlight some other methods to test primality.

One guaranteed way to test if a number p is prime is based on the results of Section 2.10. We showed in Theorem 2.10.6 that if p is prime, then there is a number a such that the set of powers $\{a, a^2, a^3, \ldots, a^{p-1}\}$ modulo p is a permutation of the list $\{1, 2, 3, \ldots, p-1\}$. Conversely, the existence of such an element guarantees that there are $p-1$ different numbers relatively prime to p. This means that p is prime. Indeed, there are $\varphi(p-1)$ such generators. So chances of finding one by trial and error are quite good. The problem, however, with this test is that if p is large, it is time-intensive to compute all powers of a number a. In this section, we discuss more efficient algorithms to test primality.

Like the Rivest-Shamir-Adelman code, the key to a more efficient algorithm comes from Fermat's Theorem. Let us suppose that a large number n is given. We know that if n is prime, then $a^{n-1} \equiv 1 \pmod{n}$. So if $a^{n-1} \not\equiv 1 \pmod{n}$ for some a, then n is definitely composite. For example, if $n = 2096004487$, we can compute $2^{n-1} \equiv 1992692247 \pmod{n}$. Hence n is composite. This does not tell much about how to factor it however.

There are some composite numbers which pass this test for all choices of a which are relatively prime to n. Such numbers are called **Carmichael** numbers. They are much less common than primes. For example, Erdös showed that the sum of their reciprocals converges. However, recent results have shown that they are nevertheless quite plentiful. An example is $n = 561 = (3)(11)(17)$. Notice that if $\gcd(a, 561) = 1$,

$$\begin{aligned} a^{560} &\equiv (a^2)^{280} \equiv 1 \pmod{3} \\ a^{560} &\equiv (a^{10})^{56} \equiv 1 \pmod{11} \\ a^{560} &\equiv (a^{16})^{35} \equiv 1 \pmod{17} \end{aligned}$$

By the Chinese Remainder Theorem, one sees that $a^{560} \equiv 1 \pmod{561}$ for all a relatively prime to 561. In fact, a^{80} would suffice.

Still, without any additional computation, it is possible to improve this test. In our example, $560 = 16(35)$. Consider the computations

$$\begin{aligned} & & 2^{35} &\equiv 263 \pmod{561} \\ 2^{70} &= \left(2^{35}\right)^2 &&\equiv 166 \pmod{561} \\ 2^{140} &= \left(2^{70}\right)^2 &&\equiv 67 \pmod{561} \\ 2^{280} &= \left(2^{140}\right)^2 &&\equiv 1 \pmod{561} \\ 2^{560} &= \left(2^{280}\right)^2 &&\equiv 1 \pmod{561} \end{aligned}$$

We see from this sequence of equations that 67 is a square root of 1 modulo 561. If 561 were a prime, there would be only two square roots, namely ± 1. So this shows conclusively that 561 is composite. In this case however, information about the factors is revealed because

$$0 \equiv 67^2 - 1 = (67-1)(67+1) \pmod{561}.$$

So $\gcd(66, 561) = 33$ and $\gcd(68, 561) = 17$ are factors of 561.

The general procedure, known as the Miller-Rabin test, uses this approach. Moreover, it does not involve any more computation than it requires to get $a^{n-1} \pmod{n}$. Pull out all factors of 2 from $n-1$, say $n-1 = 2^d m$.

Now compute $a^m \pmod n$, and then successively square it to compute $a^{2m} \pmod n$, $a^{4m} \pmod n$, $a^{8m} \pmod n$,..., $a^{n-1} \pmod n$. If 1 does not occur in this list, then n fails our earlier primality test. However, if $a^{n-1} \equiv 1 \pmod n$ and $a^m \not\equiv 1 \pmod n$, then there is a last congruence in this list which is not 1. This will be a square root of 1 in $\mathbb{Z}_n$. If it is not -1, then n is definitely composite. This is because $x^2 = 1$ has only the solutions $x = \pm 1$ in a field, but can have more solutions when n is composite.

It is also easy to check whether n has any small prime factors. The computer can store the product P of all primes less than 1000. Compute $\gcd(n, P)$. If this is not 1, then n is composite. The composite numbers which pass the Miller-Rabin test for half a dozen random choices of a are quite rare. Indeed, a large number n which passes this test and has no small prime factors is almost surely prime. Such numbers have been called industrial grade primes. They are likely to be very hard to factor.

Exercises

1. Recall that $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ is a positive integer. Prove that if $n \geq 2$, then n is prime if and only if $\binom{n}{k} \equiv 0 \pmod n$ for all $1 \leq k \leq n-1$. This result plays an important role in the Agrawal-Kayal-Saxena algorithm.

2. Show that 3053 is not prime by finding a congruence identity that contradicts primality. Do not factor it.

3. Show that 3876721 is not prime by finding a congruence identity that contradicts primality. Do not factor it.

4. Show that 1729 is a Carmichael number. Find a congruence identity that proves that n is not prime. (A factorization of n is **not** a satisfactory substitute.)

5. Show that 5755495201 is a Carmichael number. Find a congruence identity which proves that n is not prime. (A factorization of n is **not** a satisfactory substitute.) You may use computer software.

6. Show that if $p = 6k + 1$, $q = 12k + 1$, and $r = 18k + 1$ are all prime, then pqr is a Carmichael number.

7. Korselt showed that a composite integer n is a Carmichael number if and only if it is square free and for every prime $p|n$, one has $(p-1)|(n-1)$. Prove that if n has this form, then it is a Carmichael number.

4.4. Factoring Algorithms

If you wish to factor a large number using a computer, there are various tricks you can try. No method known today can factor the product of two primes with 200–300 digits before the end of the universe. Nevertheless,

methods and computers will continue to improve. However, experts feel that it will always be significantly easier to find large primes than to factor the product of two of them. So the security of our code is guaranteed if we make our primes stay ahead of the factoring game.

However, most random numbers have small prime factors as well as large ones. Any sensible factoring algorithm starts by taking the gcd of n with the product of the first few primes. In this way, all factors less than, say 1000, may be pulled out. Then test what is left to see if it is composite. If it seems prime, it almost surely is. So now you try to prove that it is prime. If it is composite, the hard work begins. Unfortunately, it is known that on average, numbers do not have very many factors (relative to their size). So most factors are very big.

Most factoring schemes use quite sophisticated mathematics. Here is an elementary idea that goes back to Lagrange. The idea is simple: try to find non-trivial solutions of

$$x^2 \equiv y^2 \pmod{n}.$$

By non-trivial solution, we mean $x \not\equiv \pm y \pmod{n}$. If n is composite, say $n = ab$, then the solutions of

$$\begin{aligned} x - y &\equiv a \pmod{n} \\ x + y &\equiv b \pmod{n} \end{aligned}$$

provide non-trivial solutions. Conversely, if x and y form a non-trivial solution, then $\gcd(x \pm y, n)$ yield proper factors of n.

Lenstra and Pommerance have added some important new ideas to this method. They hope that it will prove to be a better method than others presently known. Their plan is to look for solutions of $x \equiv y \pmod{n}$ so that x and y are both products of only small primes. If enough solutions are found, they can be used to construct a solution of $x^2 \equiv y^2 \pmod{n}$. Let us illustrate this with a small example.

Let $n = 493$. A few trials yield the following equivalences $\pmod{493}$.

$$\begin{aligned} -3 &\equiv 490 = 2 \cdot 5 \cdot 7^2 \\ 7 &\equiv 500 = 2^2 \cdot 5^3 \\ 32 &\equiv 525 = 3 \cdot 5^2 \cdot 7 \\ -7 &\equiv 486 = 2 \cdot 3^5 \end{aligned}$$

All of these equations contain only powers of -1, 2, 3, 5, and 7. Using only the total parity of the exponents, we can represent these equations by vectors. For example, the first equation has one $-$ sign, and odd powers of 2, 3, and 5; but an even power of 7. This yields the vector $(1, 1, 1, 1, 0)$. Altogether we obtain

$$\begin{gathered} (1, 1, 1, 1, 0) \\ (0, 0, 0, 1, 1) \\ (0, 1, 1, 0, 1) \\ (1, 1, 1, 0, 1) \end{gathered}$$

In order to get squares, we wish to combine them so that all the parities are even. Combining the first, second and fourth achieves this. Multiplying the

three equations together yields

$$3 \cdot 7^2 \equiv 2^4 \cdot 3^5 \cdot 5^4 \cdot 7^2.$$

Cancellation yields $1 \equiv 2^4 3^4 5^4$. This provides a solution to $x^2 \equiv y^2$ with $x = 1$ and $y = 900$. Computing $\gcd(493, 901) = 17$ and $\gcd(493, 899) = 29$ provides a complete factorization.

Exercises

1. Use computer software to find enough congruences to factor 1643 by the method described in this section.

2. **(Factoring algorithms and primality testing)** Using computer software commands for the gcd and mod, but not a complete factoring command, interactively factor $n = 217601977016409565782951 60$, and report on the steps as you go along.

(i) Test for prime factors up to 1000 and factor them out. Let the large factor remaining be called m.

(ii) Compute $3^{m-1} \pmod{m}$. What does this tell you?

(iii) One must use a brute force method to factor m. You may use that 999983 is a factor. Let the other factor be called q.

(iv) Repeat (i) for $q - 1$. This yields a prime factorization. Why?

(v) Prove that q is prime by finding a primitive root, say r.

Notes on Chapter 4

The use of codes for the purpose of secure transfer of information has a long history. The primary uses were for military and political purposes, at least initially, as these parties had great resources. During World War II, codemaking and codebreaking were crucial parts of the war effort. This was the beginning of the use of calculating machines, and led to the computer revolution. The advent of computers has made the need for security in the transmission of messages something that is of importance to all of us.

Computers also provided the means to use more sophisticated methods both for encryption and the breaking of these codes. A central issue was always how two parties could share information about a code that was safe from prying eyes. A major breakthrough was made by Diffie, Hellman and Merkle [**10**] which allowed a public exchange between two parties to agree on a common key without revealing it to any eavesdropper. Diffie proposed that one could develop an asymetrical code with a public key for encryption that only the constructor could decode. This was accomplished by Rivest, Shamir and Adleman [**32**] at MIT in 1978. It has since come out that the codebreaking division of GCHQ, the British signals intelligence agency, came up with a similar method to that of Diffie-Hellman-Merkle almost a decade

earlier, but it was kept secret until recently. Since then, other methods have been developed for public key codes.

Simon Singh's book [**36**] is an interesting, non-technical introduction to codes and codebreaking.

The use of codes to allow for accurate transmission over noisy signals goes back to work of Hamming [**16**] in 1950. Nowadays, when large data files such as computer operating systems and other software are routinely downloaded over the internet, the accuracy of transmission becomes as important as security.

Computing the list of prime numbers goes back to ancient times. However the testing of large integers to decide if they are prime, primality testing, is a modern idea relying on computers. The Miller-Rabin test [**26**, **29**] dates from the mid-1970's. The first definitive algorithm to test for primality is due to Agrawal, Kayal and Saxena [**1**]. Charmichael numbers were introduced by Charmichael in 1910. There are infinitely many such numbers [**3**], but the sum of their reciprocals is finite [**11**]; so they are rare compared to prime numbers.

Factoring of composite numbers is considered to be very difficult, which is why the RSA scheme is thought to be secure. The modification of Lagrange's ideas from Section 4.4 is due to Lenstra and Pomerance [**23**]. The possibility of quantum computers and an algorithm of Peter Shor [**33**] would make factoring practical, and would break the RSA code. Other algorithms for encryption that are secure against quantum computation have been developed, but are not yet in widespread use.