

21. Factoring Numbers I

1. Introduction

The RSA type encoding can be broken if an intruder can factor the number N of its public key. We here describe some of the methods that people have devised for factoring numbers.

Nowadays by distributing computations over many processors and many machines it is possible to perform up to roughly 10^{15} operations in attempting to do such a thing.

The mindless way to factor a large number N is to do it by testing every prime up to the square root of N to see if it is a factor. Even with the massive efforts required to perform 10^{15} operations and even if testing a prime to see if it is a factor could be accomplished with one operation, you could not factor a number much above 10^{30} this way.

There is a second algorithm, which we will call the tortoise and hare algorithm that is somewhat better, but not great, that is much more fun, and we will describe it.

Finally there are two methods which have roughly the same prospects and are considered to be the best available, and we will describe them briefly.

We begin by asking, what information can lead us to a factor of N . We will suppose that N is a product of two distinct primes p and q .

(In all our discussions we will ignore the possibility that N is not a prime but a power of primes, like 9, 27, 125 and so on. For any N it is easy to check whether each of its k -th roots are integers for k up to $\log_{1000}N$ and whether N is divisible by any number up to 1000, and that is all we need do to investigate this possibility. Therefore we will assume here that N is the product of two relatively prime numbers ab . The hardest case will be when a and b are prime, so we shall assume that as well.)

By the Chinese Remainder Theorem, we know that we can describe any remainder $x \bmod N$ by its remainders $\bmod p$ and $\bmod q$, $(x \bmod p, x \bmod q)$.

If we should stumble across an x whose pq remainders are of the form $(0 \bmod p, a \bmod q)$ and a is not 0, we can factor N by applying Euclid's algorithm to x and N .

x of this form will be a multiple of p , and so is N so the gcd of the two, which Euclid's algorithm will disclose, will be p , and we will have factored N .

Thus we want to seek x of the form $(0 \bmod p, a \bmod q)$.

Two of our methods of factoring are based on clever ways to seek such numbers. The third is based on looking for pairs of numbers whose squares, mod N are the same.

Suppose we can find two remainders mod N , x and y , that take the form $(a \bmod p, b \bmod q)$ and $(a \bmod p, (q-b) \bmod q)$. These numbers do not sum to N , but have the same square.

If we take their difference, or their sum, we get $(0 \bmod p, 2b \bmod q)$ or $(2a \bmod p, 0 \bmod q)$ and we will have a number either way of the form that allows us to factor N .

2. The Tortoise and Hare Algorithm

This approach is based upon the fact, that if you wander at random among p vertices, you can expect to visit a vertex a second time after roughly $p^{1/2}$ steps.

This claim is related to the fact that with 23 people there is an even chance that two of them have the same birthday.

For, the probability that you visit a new vertex on your next step after you have been to k of them is $(1-k/p)$. And similarly if a new person, x , enters a room after k are already present and all have different birthdays, the probability that x has birthday different from all is $(1-k/365)$.

And here is the method. We choose a polynomial f like $x^2 + x + 1$, and define $f^{(k)}(x)$ to be its k -th iterate mod N : that is, we define

$$f^{(k)}(x) = f(f^{(k-1)}(x)) \bmod N.$$

The Chinese remainder theorem tells us that the representation of remainders mod N by pairs of remainders mod p and mod q is preserved under arithmetic operations like those involved in the formation and iteration of f .

As a result we can keep track of the behavior of these iterates mod p and mod q separately.

And what will that behavior be like?

Well, mod either p or q this function, starting anywhere, can be expected to wander around, in some unknown fashion, until it hits an integer that it has hit before. After that it will follow its old path around the cycle formed by its steps between these hits on forever.

For $p=23$, for example, the function $1 + x + x^2$ goes from 1 to 3 to 13 to 22 and then to 1 and around again.

While this is happening mod p , something similar will happen mod q , but quite probably will cycle at some slightly different step.

For example, the same equation for $q=19$ goes from 1 to 3 to 13 to 12 to 5 and then to 12 then 5 12 5 12 on forever.

How do we exploit this behavior?

Imagine that we have a tortoise and a hare that race each other by iterating f . The tortoise makes one iteration each step, while the hare makes 2.

This means the hare will enter the cycle mod p or mod q before the tortoise does, but once the tortoise enters the cycle the hare will catch up to the tortoise by one vertex of the cycle, until they meet. They will meet again in a number of steps given by the length of the cycle, on forever.

Notice that once the tortoise enters the cycle, the hare will catch up in a number of steps that is at most the length of the cycle. Thus, at worst, the tortoise and hare will meet (that is, agree) mod p or mod q in a number of steps given by the step number at which the iteration hit it a value that it had previously hit.

So our algorithm consists of forming $f^{2k}(x_0) - f^k(x_0) \pmod N$ for each k , and applying Euclid's algorithm to N and this difference.

It so happens that each of these actions is extremely easy to do on a spreadsheet, so you can factor by this method very conveniently.

Since we guess it will take roughly $p^{1/2}$ or $q^{1/2}$ steps for a cycle to be formed, and since one of p and q has to be less than $N^{1/2}$, this method can be expected to take on the order of $N^{1/4}$ steps, and this is a bit too much for really large n , (above 50 or 60 decimal digits).

By the way if you guess wrong and happen to choose a function for which you do not get cycling after $N^{1/4}$ steps, you can suspect that N is a prime, or try a different function. (More than likely there is a bug in your algorithm though!)

However you can factor numbers whose size is half the precision of your computation on a spreadsheet with hardly any effort.

Here is what the first few columns should look like

factor N	33897117	$1 + x + xx$		
	1	$=\text{MOD}(1+B2+B2*B2, \$B\$1)$	=B2	=B1
	$=\text{MOD}(1+C2+C2*C2, \$B\$1)$	$=\text{MOD}(1+B3+B3*B3, \$B\$1)$	$=\text{MOD}(1+D2+D2*D2, \$B\$1)$	=E2
	$=\text{MOD}(1+C3+C3*C3, \$B\$1)$	$=\text{MOD}(1+B4+B4*B4, \$B\$1)$	$=\text{MOD}(1+D3+D3*D3, \$B\$1)$	=E3
	$=\text{MOD}(1+C4+C4*C4, \$B\$1)$	$=\text{MOD}(1+B5+B5*B5, \$B\$1)$	$=\text{MOD}(1+D4+D4*D4, \$B\$1)$	=E4
	$=\text{MOD}(1+C5+C5*C5, \$B\$1)$	$=\text{MOD}(1+B6+B6*B6, \$B\$1)$	$=\text{MOD}(1+D5+D5*D5, \$B\$1)$	=E5

=MOD(1+C6+C6*C6,\$B\$1)	=MOD(1+B7+B7*B7,\$B\$1)	=MOD(1+D6+D6*D6,\$B\$1)	=E6
=MOD(1+C7+C7*C7,\$B\$1)	=MOD(1+B8+B8*B8,\$B\$1)	=MOD(1+D7+D7*D7,\$B\$1)	=E7
=MOD(1+C8+C8*C8,\$B\$1)	=MOD(1+B9+B9*B9,\$B\$1)	=MOD(1+D8+D8*D8,\$B\$1)	=E8
=MOD(1+C9+C9*C9,\$B\$1)	=MOD(1+B10+B10*B10,\$B\$1)	=MOD(1+D9+D9*D9,\$B\$1)	=E9
=MOD(1+C10+C10*C10,\$B\$1)	=MOD(1+B11+B11*B11,\$B\$1)	=MOD(1+D10+D10*D10,\$B\$1)	=E10
=MOD(1+C11+C11*C11,\$B\$1)	=MOD(1+B12+B12*B12,\$B\$1)	=MOD(1+D11+D11*D11,\$B\$1)	=E11
=MOD(1+C12+C12*C12,\$B\$1)	=MOD(1+B13+B13*B13,\$B\$1)	=MOD(1+D12+D12*D12,\$B\$1)	=E12
=MOD(1+C13+C13*C13,\$B\$1)	=MOD(1+B14+B14*B14,\$B\$1)	=MOD(1+D13+D13*D13,\$B\$1)	=E13
=MOD(1+C14+C14*C14,\$B\$1)	=MOD(1+B15+B15*B15,\$B\$1)	=MOD(1+D14+D14*D14,\$B\$1)	=E14
=MOD(1+C15+C15*C15,\$B\$1)	=MOD(1+B16+B16*B16,\$B\$1)	=MOD(1+D15+D15*D15,\$B\$1)	=E15

And then to the right of these:

=ABS(B3-D3)	=MOD(E3,F3)	=MOD(F3,G3)	=MOD(G3,H3)
=ABS(B4-D4)	=MOD(E4,F4)	=MOD(F4,G4)	=MOD(G4,H4)
=ABS(B5-D5)	=MOD(E5,F5)	=MOD(F5,G5)	=MOD(G5,H5)
=ABS(B6-D6)	=MOD(E6,F6)	=MOD(F6,G6)	=MOD(G6,H6)
=ABS(B7-D7)	=MOD(E7,F7)	=MOD(F7,G7)	=MOD(G7,H7)
=ABS(B8-D8)	=MOD(E8,F8)	=MOD(F8,G8)	=MOD(G8,H8)
=ABS(B9-D9)	=MOD(E9,F9)	=MOD(F9,G9)	=MOD(G9,H9)
=ABS(B10-D10)	=MOD(E10,F10)	=MOD(F10,G10)	=MOD(G10,H10)
=ABS(B11-D11)	=MOD(E11,F11)	=MOD(F11,G11)	=MOD(G11,H11)
=ABS(B12-D12)	=MOD(E12,F12)	=MOD(F12,G12)	=MOD(G12,H12)
=ABS(B13-D13)	=MOD(E13,F13)	=MOD(F13,G13)	=MOD(G13,H13)
=ABS(B14-D14)	=MOD(E14,F14)	=MOD(F14,G14)	=MOD(G14,H14)
=ABS(B15-D15)	=MOD(E15,F15)	=MOD(F15,G15)	=MOD(G15,H15)
=ABS(B16-D16)	=MOD(E16,F16)	=MOD(F16,G16)	=MOD(G16,H16)

The output looks like

```

33897117 1 + x + xx)
  1      3      1 33897117
13      183     3 33897117      10      7      3      1      0 #DIV/0! #DIV/0! #DIV/0!
33673 15299742      13 33897117      33660      1497      726      45      6      3      0 ! !
19995619 33277233      183 33897117 19995436 13901681 6093755 1714171 951242 762929 2E+05 9677 4450
31732378 19730835      33673 33897117 31698705 2198412 920937 356538 207861 148677 59184 30309 28875
23066836 2119890 15299742 33897117 7767094 2828741 2109612 719129 671354 47775 2504 199 116
25445716 29810436 19995619 33897117 5450097 1196535 663957 532578 131379 7062 4263 2799 1464
17448766 21896967 33277233 33897117 15828467 2240183 147186 32393 17614 14779 2835 604 419
25138633 27004455 31732378 33897117 6593745 928392 95001 73383 21618 8529 4560 3969 591
7045297 24395184 19730835 33897117 12685538 8526041 4159497 207047 18557 2920 1037 846 191
21352090 19489635 23066836 33897117 1714746 1316943 397803 123534 27201 14730 12471 2259 1176
18976879 1463031 2119890 33897117 16856989 183139 8201 2717 50 17 16 1 0
27717028 18979434 25445716 33897117 2271312 2098749 172563 27993 4605 363 249 114 21
193564 10901376 29810436 33897117 29616872 4280245 3935402 344843 142129 60585 20959 18667 2292
5413102 27724314 17448766 33897117 12035664 9825789 2209875 986289 237297 37101 14691 7719 6972

```

12661243	27719085	21896967	33897117	9235724	6189945	3045779	98387	94169	4218	1373	99	86
21164344	25377987	25138633	33897117	3974289	2102805	1871484	231321	20916	1245	996	249	0

Notice what happens in the last row here.

3. The Quadratic Sieve Method of Factoring

This method involves a systematic search for two remainders that do not sum to N that have the same square mod N .

The basic plan is to square many numbers, and pick out those whose squares have only small prime factors.

When you accumulate enough of these, you will be able to find a product of them z , mod N that is a perfect square of small prime factors.

This means a number that can be written as $p_1^{2s_1} * p_2^{2s_2} \dots p_k^{2s_k}$.

We can then identify $p_1^{s_1} * p_2^{s_2} \dots p_k^{s_k}$ as a square root of z , and also we have a square root of z that is the product of the numbers whose squares had product z mod N .

These two square roots could be the same, or negatives of one another, but they are just as likely to be completely different, in which case they allow you to factor N . After you accumulate enough such z you will be able to factor N with very high probability.

We can describe this algorithm by describing **how to get lots of squares of numbers mod N efficiently**, and **how to find a product of them that is a perfect square**.

The awkward part of dealing with large numbers is that it takes time and effort to do anything with them. However, one thing that is relatively easy to do is to divide one by a small number and examine the remainder obtained.

So here is the plan for getting lots of squares with only small prime factors.

We evaluate the greatest integer less than $N^{1/2}$ (or $(kN)^{1/2}$ for small k); call it y . Then numbers of the form $y+j$ for reasonably small j will have squares that are of the form $y^2 + jy + j^2$.

But we have $y^2 < N$ and $(y+1)^2 > 0$. Since the difference between these two squares is $2y+1$ we know that both y and $(y+1)^2$ are numbers that mod N are of the order of the square root of N .

This means that all the numbers of the form $y^2 + 2jy + j^2$ will have order $N^{1/2}$ and will be perfect squares mod N .

For any prime p , we determine $(y+1)^2$ and $y \bmod p$ and these allow us to determine what all the numbers of the form $(y+j)^2 \bmod p$, by manipulating only small numbers.

So here is the plan. You pick a few million numbers of the form $(y+j)^2$, and find what each of them are mod each prime p up to say a million.

From this information you can divide each square by each prime p that divides it until the result is no longer a multiple of p . If you reduce the result to 1 in this way, then the number you are dealing with is the product of powers of your small primes.

Now you accumulate a large number of such squares. As we saw in the previous Chapter, the proportion of numbers having this property is roughly $\log M / \log N^{1/2}$, where M is the size of the largest prime considered small here.

Each of the squares s obtained can be described by a vector each of whose components corresponds to a prime p , and whose entry is a 1 if s/p^{2k+1} is an integer that is not a multiple of p , and 0 otherwise.

You will find a perfect square if you can find a linear dependence among these vectors, mod 2.

To see this let us take a mini-example

Suppose we found squares that mod N were 10, 15, 18 and 3.

We would describe these as the following vectors, with component order 2,3,5.

(101), (011), (100), and (010)

.

The standard technique called row reduction, allows us to find a linear dependence.

It can be described as follows. Assign to each vector v its largest non-zero component unless that component has already been assigned to another vector w ; in that case replace v by $v+w$ and repeat using this step starting with $v+w$.

In our case the first vector is assigned to the third component. The second is replaced by its sum with the first (110) which is assigned to the second component. The third is assigned to the first component.

The fourth vector can be assigned no component and this means we have a linear dependence. It is first replaced by its sum with the first and second (100), and this is replaced by its sum with the third (000).

You would try to assign it to the second component, but that already has the sum of the first two vectors assigned to it. Adding the fourth to that leaves the first component but that has been assigned to the third vector. Adding that in gives the (0) vector, and the linear dependence is the sum of all four vectors, which is $(0) \pmod 2$.

This tells us that $10 \cdot 15 \cdot 18 \cdot 3$ is a perfect square, and in fact it is the square of 90. It also would be the square of the product of the four numbers that were used to produce these squares.

And that is the quadratic sieve method for factoring.

When you find two numbers with the same square, of course you will use Euclid's algorithm starting with N and the difference (or sum) of the two, and obtain a prime factor of N as the gcd.

Exercise: 1. Form the product of two 4 digit primes and factor it with the tortoise hare algorithm. Try again with the product of 5 digit primes.
2. Explain the quadratic sieve algorithm in your own words. Which steps will take the most time?