Chapter IV: Secret Codes continued

4 Factoring Large Numbers

There are some wonderfully clever ways to factor numbers and we will look at some of them; they allow factoring of fairly large numbers: up to perhaps 160-digit numbers today. We have no idea whether there are better methods lurking somewhere that could do much better. At present therefore, the RSA coding scheme looks good, if large enough primes are used, but who knows?

The textbook method for factoring a number n involves testing each prime up to $n^{1/2}$ to see if it divides n. It requires on the order of $n^{1/2}$ trial divisions, and is thus not particularly interesting.

4.1 The Cycle Method

If n is composite then it can be written as $n = ab$ where a and b are relatively prime and greater than 1. One good method for factoring is called the *cycle* method. It is based on the following several facts:

1. If we can find a z that is a multiple of a or b but not of both, then by using Euclid's algorithm with n and z we can find a non-trivial factor of n since $gcd(n, z)$ is strictly between 1 and n.

2. If a function f maps the set of integers mod n onto itself, with the property that the a-remainder of $f(x)$ depends only on the a-remainder of x, and the b-remainder of $f(x)$ depends only on the b-remainder of x, then f is characterized by a function f_a on the a-remainders alone and a function f_b on the b-remainders alone. Any polynomial f will have this property.

3. If we take a random function g on the space of all functions that map $\{0 \ldots a-1\}$ onto itself, and, starting at some element x, iterate the function, forming x, $g(x)$, $g(g(x))$, $g(g(g(x))), \ldots$, then with high probability we obtain the same value more than once by the time we have iterated $a^{1/2}$ times. (For, there is a probability $1/a$ that we repeat immediately; if we don't, we have a probability $2/a$ of getting a repeat on the second iteration, then $3/a$ on the third, etc. The probability of seeing no repeats after k iterations is therefore $(a-1)\ldots(a-k)/a^k$. Please verify that this probability is almost 0 for $k \gg a^{1/2}$ and almost 1 for $k \ll a^{1/2}$.)

   Thus the function f_a can be expected to start repeating after roughly $a^{1/2}$ iterations.

4. Let $g^{(k)}(x)$ denote the kth iterate of function g on x. If g repeats with $g^{(l)}(x) = g^{(l+j)}(x)$, then $g^{(2k)}(x) = g^{(k)}(x)$ for some $k \leq l + j$. (To see this apply $g^{(j)}$ to both sides of the first equation to get $g^{(l+j)}(x) = g^{(l+2j)}(x)$. Similarly, $g^{(l+2j)}(x) = g^{(l+3j)}(x)$, and so on, so that $g^{(l)}(x) = g^{(l+ij)}(x)$ for all positive i. Set $i = \lceil l/j \rceil$ so that $ij \geq l$. Applying $g^{(ij-l)}$ to both sides of this last equation, we obtain $g^{(ij)}(x) = g^{(2ij)}(x)$.)

5. If $f_a^{(2k)}(x) = f_a^{(k)}(x)$ then $f^{(2k)}(x) - f^{(k)}(x)$ is a multiple of a. On the other hand, if f was truly random, then f_a and f_b have nothing to do with one another, and the probability that $f_b^{(2k)}(x) = f_b^{(k)}(x)$ is very low (on the order of $1/b$). That is, our k almost certainly has the property that $z = f^{(2k)}(x) - f^{(k)}(x)$ is divisible by a but not by b.

Algorithm. Thus our procedure is to choose a suitable function f somehow, and for $k = 1, 2, \ldots$ calculate
$$gcd(n, f^{(2k)}(x) - f^{(k)}(x)).$$

Example. We try to factor 77 with $f(x) = x^2 + 1$ and $x = 2$. Then $f(x) = 5$, $f^2(x) = 26$ and $gcd(77, 26 - 5) = 11$. This is unusually swift success.

This procedure will find factors so long as the function f used obeys the condition described in Fact 2. We expect to have to iterate f_a something like $2a^{1/2}$ times to find a pair $k, 2k$ for which $f_a^{(2k)}(x) = f_a^{(k)}(x)$.

Almost any polynomial is a good candidate to try. We don't really know that any particular function will behave like a "random" function but will usually not be disappointed unless there is some trivial reason why. An obvious function to try is $f(x) = x^2 + 1$. If this one doesn't work (e.g. takes too long to cycle), one can try another function, or use the same function and a different starting point for the iteration.

The method can be expected to require a number of steps on the order of the square root of the smallest prime factor of n, which, if $n = pq$ with p and q primes of approximately the same size, would be on the order of $n^{1/4}$. This means the method can handle 40- and perhaps even 50-digit numbers, but that is about as far as one can go with it.

4.2 The Quadratic Sieve Algorithm (Killian)

The following presentation is partially based on two guest lectures by Joe Killian that he gave in 18.425J in the Spring of 1988.

Dixon. First we present a related algorithm called "Dixon's algorithm". Suppose $n = pq$ where p and q are large primes whose values are unknown to us. Suppose we stumble upon two numbers x and y between 1 and n such that $x^2 \equiv y^2 \bmod n$. Then $(x-y)(x+y) = x^2 - y^2$ is divisible by $n = pq$. If we are lucky, then $x - y$ will be divisible by one of the primes p, q but not the other, so that in calculating $gcd(x - y, n)$, we discover what p is.

In fact, in a certain sense no great luck is required: for *every* x between 1 and n, there is a y that does the job for us. (Actually, this is not quite true; when x is divisible by p or q, the following argument breaks down; but when this happens, we're *really* lucky: we can recover that prime factor just by calculating $gcd(x, n)$.) Fix x between 1 and n, divisible by neither p nor q. Then, in order to have $x^2 \equiv y^2 \bmod n$, it is necessary and sufficient to have $x^2 \equiv y^2 \bmod p$ and mod q. But $x^2 \equiv y^2 \bmod p$ if and only if p divides $x^2 - y^2 = (x-y)(x+y)$, which happens if and only if p divides $x - y$ or $x + y$, which happens if and only if $y \equiv \pm x \bmod p$. Similarly, $x^2 \equiv y^2 \bmod q$ if and only if $y \equiv \pm x \bmod q$. Thus

$$x^2 \equiv y^2 \bmod n \text{ if and only if } y \equiv \pm x \bmod p \text{ and } \bmod q.$$

Using the Chinese Remainder Theorem, we see that there are exactly four y's satisfying this condition: y_1, y_2, y_3, y_4, with $y_1 = x$, $y_2 = n - x$, $y_3 \equiv x \bmod p$ and $\equiv -x \bmod q$, and $y_4 = n - y_3$. Of these, both y_3 and y_4 have the property that $gcd(x-y, n)$ is a prime factor of n.

This suggests a possible strategy: take a number x, square it, then find one of the four square roots of $x^2 \bmod n$; call it y. If $y \not\equiv \pm x$ then we calculate $gcd(x - y, n)$ which is a proper factor of n.

Example: Take $n = 33$ and $x = 7$. Then $7^2 \equiv 16 \bmod 33$. So $y = 4$ (say). Then $gcd(7 - 4, 33) = 3$.

But nobody knows how to efficiently take square roots mod n without already knowing the factorization of n! So we might take a number and square it mod n in the hope that the result is an obvious square in the integers. But there are only $\sqrt{n}$ such numbers in Z_n, so the odds of hitting such numbers are horrendously low.

The fix: sometimes we can multiply "bad" equations together to get a good one. For example try factoring 253. Then

$$195^2 \equiv 75 \bmod 253, \quad \text{and} \quad 112^2 \equiv 147 \bmod 253,$$

but neither gives a square in the integers. But $75 \cdot 147$ is; it's 105^2. Since $195 \cdot 112 \equiv 82 \bmod 253$, multiplying the two equations above yields that

$$82^2 \equiv 105^2 \bmod 253,$$

and we obtain $gcd(105-82, 253) = 23$.

So maybe multiplying non-square equations works. Unfortunately, the numbers in the above example were a bit cooked: $75 = 3 \cdot 5^2$ and $147 = 3 \cdot 7^2$, so of course the product of these numbers will be a square. But suppose we have a set of equations:

$$\begin{aligned} x_1^2 \bmod n = z_1 = 2^{\alpha_{12}} \cdot 3^{\alpha_{13}} \cdot 5^{\alpha_{15}} \ldots \\ x_2^2 \bmod n = z_2 = 2^{\alpha_{22}} \cdot 3^{\alpha_{23}} \cdot 5^{\alpha_{25}} \ldots \\ \vdots \end{aligned}$$

Maybe we can combine some subset of the z_i to give us a perfect square. The question is, how can we determine which equations to multiply? To do this, all we need to do is to keep track of the parity (evenness or oddness) of the α_{ij} with a *characteristic vector* for each z_i, where the j-th bit of the vector indicates the parity of α_{ij}. Finding z_i's that multiply to a square is the same as finding vectors that sum to the 0 vector. When we have such z_i's, we are done.

Example:

$$\begin{array}{ll} z_1 = 10 = 2 \cdot 5 & \text{vector is } 1010 \\ z_2 = 126 = 2 \cdot 3^2 \cdot 7 & \text{vector is } 1001 \\ z_3 = 24 = 2^3 \cdot 3 & \text{vector is } 1100 \\ z_4 = 875 = 5^3 \cdot 7 & \text{vector is } 0011. \end{array}$$

Then $1010+1001+0011 = 0000$, so that $x_1^2 x_2^2 x_4^2 = (x_1 x_2 x_4)^2 = z_1 z_2 z_4 = 2^2 \cdot 3^2 \cdot 5^4 \cdot 7^2 = (2 \cdot 3 \cdot 5^2 \cdot 7)^2 = 1050^2$. We then use Euclid's algorithm to compute $gcd(x_1 x_2 x_4 - 1050,\ n)$ in the hopes of splitting n.

But, to find these parity vectors we must be able to factor the z_i's! Have we just reduced factoring to factoring? Yes, *but*: because we can get away with considering only z_i's which are divisible only by small primes, we can factor them comparatively quickly (using trial division, for instance).

Algorithm: Repeat the following until we have enough good x_i: pick x_i at random, compute $z_i = (x_i)^2 \bmod n$ and factor it (if it factors completely into small primes). Then find a subset of the x_i such that the corresponding z_i form a perfect square. Then calculate gcd as above.

Pomerance. Unfortunately, we might have to try many x_i's before we find enough good ones. The idea behind the Pomerance quadratic sieve algorithm is that we will be a little more clever about the numbers we try. We will deterministically pick a sequence of "small" numbers to use in Dixon's algorithm.

An obvious choice for a sequence of "small" numbers is the sequence 1, 2, 3, ... Unfortunately, after factoring these numbers, we find that multiplying *any* subset of these numbers produces the trivial identity $x^2 = x^2$. Clearly some of the numbers must wrap beyond the modulus n before we can hope to get a non-trivial identity.

The first number that when squared gives a value bigger than n is $\lceil\sqrt{n}\rceil$. This motivates which numbers we are going to choose to try to get our small squares for the quadratic sieve algorithm.

We use the sequence $\lceil\sqrt{n}\rceil, 1+\lceil\sqrt{n}\rceil, 2+\lceil\sqrt{n}\rceil, \ldots$, whose xth term we denote by $S_x = x + \lceil\sqrt{n}\rceil$. We then have the identity:

$$\left(x+\lceil\sqrt{n}\rceil\right)^2 \equiv x^2 + 2x\lceil\sqrt{n}\rceil + \lceil\sqrt{n}\rceil^2 - n \mod n.$$

Thus the numbers produced by squaring S_x are larger then n, but not too much larger: since the third and fourth terms on the right hand side are almost equal, we are left with approximately $x^2 + 2x\lceil\sqrt{n}\rceil$, which is much smaller than n for small x.

Quadratic Sieve. Because the S_x's are fully determined ahead of time, we can factor the squares of many S_x's *together* faster than we could factor them individually. Methods for factoring a whole bunch of numbers together, by taking primes in turn and writing down their multiples in a table, are called *sieve* methods. There is a special sieve that we can use with the S_x's called the *quadratic sieve.* It uses the fact that we can find a square root modulo a prime number fairly quickly.

We will not give here a procedure which finds a square root modulo a prime. Rather we will *assume* we have such a procedure and use it in our quadratic sieve. Consider the equation that determines whether the square of an S_x (or rather, its remainder upon division by n) has a prime factor f:

$$x^2 + 2x\lceil\sqrt{n}\rceil + \left(\lceil\sqrt{n}\rceil^2 - n\right) \equiv 0 \mod f.$$

This is just a quadratic equation in the unknown x. We solve for x by using the quadratic formula and taking square roots modulo the prime f. Let the solutions obtained be x_1 and x_2. But now we can characterize the squares of S_x's which have f as a factor as all those of the form $x_1 + i \cdot f$ or $x_2 + i \cdot f$ for nonnegative integer i. Thus, given a table with an entry for every S_x, we can mark all entries that are congruent to x_1 or x_2 modulo f, and add f to their factor lists. To factor all the numbers on the table, we start with the small primes and repeat this procedure until f has run through all primes in the table.

Another way to think about the sieving procedure is to observe that z_i and z_{i+f} are congruent to one another mod f. For, letting $A = \lceil n\rceil^2 - n$ and $B = 2\lceil n\rceil$, we have $z_i = A + Bi + i^2$ and $z_{i+f} = A + B(i+f) + (i+f)^2$, so that

$$z_{i+f} - z_i = Bf + 2if + f^2,$$

a multiple of f. Thus, if z_i is divisible by f, so are z_{i+p}, z_{i+2p}, etc.

Example. Consider $n = 14,579,877$: Then

$$x^2 + 2x\left\lceil\sqrt{n}\right\rceil + \left(\left\lceil\sqrt{n}\right\rceil^2 - n\right) = x^2 + 7638x + 4884.$$

When we apply the procedure for $f = 7$, we obtain the quadratic

$$x^2 + x + 5 = 0 \bmod 7,$$

whose solutions are $x_1 = 1 \bmod 7$ and $x_2 = 5 \bmod 7$. Thus we know we can add x to the factor lists of $x = 1, 8, 15$, etc., and $x = 5, 12, 19$, etc.

One additional comment. Some numbers in our sequence can have their prime factors with exponents greater than 1, i.e., can be divisible by prime powers. How do we factor these? It turns out we can take square roots modulo a prime power quickly, just as we can for square roots modulo a prime. However, finding square roots modulo $N = pq$ is just as hard as factoring, which (as we have been claiming throughout this chapter) is quite hard indeed.

4.3 The Quadratic Sieve Method (Kleitman)

We can factor n if we can find two distinct remainders mod n that do not sum to n and that have the same square mod n.

Thus one possible way to factor n, is to square random numbers mod n, hoping the square we get is something we know, like, $1, 4, 9, 16, \ldots$.

Dixon suggested a method that is based on this idea.

Square numbers at random and then factor their squares, keeping track of the non-squares factors if they are smaller than some threshold.

Each such square can be described by a zero-one vector, whose components represent primes below the threshold, with a component value equal to one when that prime occurs in the square. For example $126 = 2 \cdot 3^2 \cdot 7$ has the vector $(1, 0, 0, 1)$.

Once one has a set of such vectors that are linearly dependent, we can find a combination of the numbers whose squares is that of a known number. With good luck, this will lead to a factorization.

Thus for example, suppose we find that $a^2 = 2 \cdot 3^2 \cdot 5 \bmod n$, so the vector is $(1, 0, 1, 0)$. Similarly

$$\begin{aligned}
a^2 &= 2 \cdot 3^2 \cdot 5 \bmod n && (1,0,1,0),\\
b^2 &= 5 \cdot 7 \cdot 12^2 \bmod n && (0,0,1,1),\\
c^2 &= 2^4 \cdot 3^3 \cdot 5 \bmod n && (0,1,1,0),\\
d^2 &= 2 \cdot 7 \cdot 5^2 \bmod n && (1,0,0,1).
\end{aligned}$$

Then the sum of the first, second and fourth vectors is 0. Thus the product of the left hand variables is a square. Hence

$$(a \cdot b \cdot d)^2 = (2 \cdot 3 \cdot 5 \cdot 7 \cdot 12 \cdot 5)^2 \bmod n.$$

That is, we can now write

$$\alpha^2 = \beta^2 \bmod n,$$

where $\alpha = a \cdot b \cdot d$ and $\beta = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 12 \cdot 5$. This is equivalent to

$$(\alpha + \beta) \cdot (\alpha - \beta) = 0 \bmod n,$$

by the difference of squares. If $\alpha + \beta \neq 0 \bmod n$ and $\alpha - \beta \neq 0 \bmod n$, then $gcd(\alpha + \beta, n)$ and $gcd(\alpha - \beta, n)$ are nontrivial factors of n.

Pomerance noticed that one can find candidates $a, b, \ldots, d$ whose squares are relatively small by choosing number x of the form least integer greater than square root of $k \cdot n$ for small integers k. Then $x = \sqrt{k \cdot n} + y$, for $y < 1$, so that

$$\begin{aligned} x^2 &= \left(\sqrt{k \cdot n} + y\right)^2 \\ &= k \cdot n + 2 \cdot \sqrt{k \cdot n} \cdot y + y^2. \end{aligned}$$

So

$$x^2 = 2 \cdot \sqrt{k \cdot n} \cdot y + y^2 \bmod n.$$

Thus x^2 is of the order of the $2 \cdot \sqrt{k \cdot n}$ which is already much smaller than n.

Using this method with numbers of this kinds is the best known method for factoring numbers. The best result (factoring a 160 digit number) was obtained as a cooperative effort of thousands of computer users, each helping by investigating a different value of x; between them all they could find a linearly dependent vector and factor the number.

Doing this involves factoring 80 digit numbers namely the squares of our x's, and involved vectors with perhaps 1000 components, corresponding to the first thousand primes.

To analyze how fast this method is, one has to know how many numbers x will have all their non-square prime factors mod n below our threshold. This can be done.

4.4 Elliptic Curve Method

Omitted 1995.

Exercises.

1. Write a program for doing Euclid's Algorithm to find the multiplicative inverse of a mod n when it exists.

2. 10961 is the product of two primes. Find them using the cycle method. (Hint: If all else fails, use $f(x) = x^2 + 1$ with seed $x = 2$.)

3. 22999 is the product of two primes. Find them using Dixon's algorithm with the following sequence of "random" numbers:

$$16597, 21447, 20484, 1554, 8088, 1586, 4693, 23.$$

4. There is a non-trivial polynomial $a(x)$ that divides both of the following polynomials over the 2-element field:

$P(x) = 1 + x^2 + x^3 + x^5 + x^9 + x^{10} + x^{12}$, and
$Q(x) = 1 + x + x^3 + x^4 + x^5 + x^6 + x^7 + x^8 + x^{10} + x^{11} + x^{12}$.

Find it. (Hint: Use a suitable version of Euclid's algorithm that applies to polynomials.)

5. Factor 49163 by the cycle method.

6. Write a program to factor large numbers by the cycle method and use it to factor 8264719.

7. Factor 481 with Pomerance.