

Factorization Methods: Very Quick Overview

Yuval Filmus

February 25, 2010

1 Introduction

In this lecture we introduce modern factorization methods. We will assume several facts from analytic number theory. The analyses we present are not formal, but serve well to explain why the algorithms work. Also, since some of the algorithms are quite intricate, we won't give a full description of them, rather only their flavor.

Our base line algorithm is trial division, which will factor an integer n in time proportional to \sqrt{n} . However, if n has a small prime factor p , then this factor would be found much faster, viz. in time p . Several of the algorithms we consider below have this property.

We will be mostly interested in the difficult RSA case, viz. $n = pq$ with p, q primes of size approximately \sqrt{n} . This case is interesting, since if you can factor n then you can break the corresponding cryptosystem.

2 Pollard's rho

Suppose that x, y are random integers that happen to satisfy $x \equiv y \pmod{p}$ for some factor p of n . With high probability (at least in the RSA case), in fact $(n, x - y) = p$, so such a pair leads to a factorization of n . How do we find such a pair of integers? Suppose that we are supplied with N random integers modulo n . How many such pairs x, y do we expect? About $\binom{N}{2}/p \approx N^2/2p$. Thus, given $N \approx \sqrt{2p}$ random integers, we expect there to be a pair x, y which leads to a factorization of p . The question is how to find such a pair efficiently (testing all pairs will lead to a trivial $O(p)$ algorithm).

The trick is to have the numbers form a pseudorandom sequence. Decide on some pseudorandom *rational function* $\varphi: \mathbb{Z}_n \rightarrow \mathbb{Z}_n$, and form a sequence $x_{i+1} = \varphi(x_i)$, with some arbitrary starting point. Note that since φ is defined only using arithmetic operations, then the sequence law $x_{i+1} = \varphi(x_i)$

holds also modulo p . After about $\sqrt{2p}$ iterations, the sequence will close on itself, roughly at the middle. This eventually periodic structure of the sequence is the origin of the name ρ . The question is how to find two points x_i, x_j that satisfy $x_i \equiv x_j \pmod{p}$.

If we know in advance the size of the smallest factor p of n (as in the RSA case), then we can start with $x_{\sqrt{2p}}$, which should be on the cycle, and compare each further value to it; after we finish the entire cycle, we will reach a point $x_{\sqrt{2p}+\ell}$ satisfying $(x_{\sqrt{2p}+\ell} - x_{\sqrt{2p}}, n) = p$ (the GCD can also be larger, but we expect that to happen with negligible probability). The total running time is at most $2\sqrt{2p}$, which is $O(\sqrt[4]{n})$ in the RSA case.

In case we do not have an estimate of p , we can use one of the following two methods, which are in fact slightly faster than the method presented. The first method (Floyd's) compares x_t to x_{2t} constantly; we detect the cycle when t is larger than the initial segment, and is a multiple of the cycle length. The second method (Brent's) comprises of a list of snapshot times t_i . Each x_t is compared to the most recent snapshot. Taking $t_i = 2^i$ results in a better algorithm than Floyd's.

3 Pollard's $p - 1$

The next algorithm we describe is Pollard's $p - 1$ algorithm, on which one of the fastest algorithms around, Lenstra's ECM, is based. The idea is to use Fermat's little theorem: $a^{p-1} \equiv 1 \pmod{p}$. Consider the RSA setting. Suppose we find an integer N which satisfies $p - 1 \mid N$ but $q - 1 \nmid N$. Then for general a , $a^N \equiv 1 \pmod{p}$ but $a^N \not\equiv 1 \pmod{q}$ (this fails to hold in the lucky but rare case when $(a, n) > 1$). Therefore, in that case $(a^N - 1, n) = p$.

We are going to search systematically for such an N . It is extremely likely that the largest prime factor P of $p - 1$ is different than the largest prime factor Q of $q - 1$. Without loss of generality, $P < Q$. The following N will then factor n :

$$N = \prod_{\pi \leq P} \pi^{\lfloor \log n / \log \pi \rfloor},$$

where the product is over all primes less than P . Pollard's method works by starting with some a and repeatedly raising it to the $\pi^{\lfloor \log n / \log \pi \rfloor}$ power, constantly checking whether for the current exponent N , $(a^N - 1, n)$ is non-trivial. Raising a number to an exponent k can be done using $2 \log k$ multiplications (using repeated squaring), and so if we have to go up to P , the total amount of work is proportional to $P \log P$, or even to P if we take

care to only peruse prime π (the prime number theorem shows that there are $P/\log P$ primes below P).

What is the efficiency of this method? The largest prime factor of a random integer m is about $m^{0.62433}$, where 0.62433 is the Golomb-Dickman constant. Therefore, we expect the method to run in time $p^{0.62433}$ for the smallest prime factor p of n . In the RSA case, this boils down to $n^{0.31217}$. In the worst case, $(p-1)/2$ is also prime (such primes are known as Sophie Germain primes), and then the running time is the same as trial division.

An optimization which is used in practice takes notice of the fact that the second largest prime factor is of size only $m^{0.20958}$. Therefore, we expect there to be only one prime factor larger than $m^{0.20958}$. Thus, once we get to $a^{N(m^{0.20958})}$, we only need to test the numbers $a^{N(m^{0.20958})P}$ with P starting with $m^{0.20958}$. In each step, instead of raising to the power P , we only multiply by a , which is significantly faster. This optimization doesn't considerably improve the asymptotic running time, but is very useful in practice.

We comment on the amusing fact that the expected exponents of the largest prime factors are the same as the expected lengths of the largest cycles in a random permutation, as a fraction of the number of points! The constant 0.62433 is the Golomb-Dickman constant. The classical paper analyzing cycle lengths in permutations is *Ordered Cycle Lengths in a Random Permutation* by Shepp & Lloyd. Knuth & Prado (*Analysis of a simple factorization algorithm*) have done the calculations for factorizations, amazingly getting the same results. See also Granville's *The Anatomy of Integers and Permutations*.

4 Williams' $p+1$

Pollard's method can be adapted to a slightly different setting, where reliance on $p-1$ is replaced by reliance on $p+1$. The trick is to replace ordinary powers with traces of powers in a quadratic field, picking an element of order $p+1$ (the multiplicative group of the field has order p^2-1).

Given a parameter A , define a Lucas sequence V_i by

$$\begin{aligned} V_0 &= 2, \\ V_1 &= A, \\ V_m &= AV_{m-1} - V_{m-2}. \end{aligned}$$

We know that $V_m = C_1\alpha_1^m + C_2\alpha_2^m$, where α_1, α_2 are the roots of the characteristic polynomial $t^2 - At + 1$ (given that $\alpha_1 \neq \alpha_2$). The roots of this

polynomial are

$$\alpha_{1,2} = \frac{A \pm \sqrt{A^2 - 4}}{2}.$$

It is easy to verify that $C_1 = C_2 = 1$, and so we obtain the formula

$$V_n = \left(\frac{A + \sqrt{A^2 - 4}}{2} \right)^n + \left(\frac{A - \sqrt{A^2 - 4}}{2} \right)^n.$$

Now suppose that the entire computation is done modulo p for some prime p . We can think of the two roots α_1, α_2 as elements of the field $\mathbb{Z}_p[\sqrt{A^2 - 4}]$.

If $A^2 - 4$ is a quadratic non-residue, then this is a quadratic field, and V_n is the trace of α_1^n . Moreover, $(A^2 - 4)^{(p-1)/2} \equiv -1 \pmod{p}$, and so

$$\left(x + y\sqrt{A^2 - 4} \right)^p = x - y\sqrt{A^2 - 4} \quad \text{in } \mathbb{Z}_p[\sqrt{A^2 - 4}].$$

Thus $\alpha_1^p = \alpha_2$, and so $\alpha_1^{p+1} = \alpha_2^{p+1} = \alpha_1\alpha_2 = 1$. It immediately follows that $V_{c(p+1)} \equiv 2 \pmod{p}$ for any c .

If, on the contrary, $A^2 - 4$ is a quadratic residue, then $\mathbb{Z}_p[\sqrt{A^2 - 4}]$ reduces to \mathbb{Z}_p . Thus

$$V_{c(p+1)} = \alpha_1^{c(p+1)} + \alpha_2^{c(p+1)} \equiv \alpha_1^{2c} + \alpha_2^{2c} \pmod{p}.$$

Since $\alpha_1\alpha_2 = 1$, we have

$$(\alpha_1^{2c} - 1)(\alpha_2^{2c} - 1) = 2 - \alpha_1^{2c} - \alpha_2^{2c}.$$

Thus $V_{c(p+1)} \equiv 2 \pmod{p}$ if and only if $\alpha_1^{2c} \equiv \alpha_2^{2c} \equiv 1 \pmod{p}$.

Summarizing, for general A which are quadratic non-residues modulo p , we have $V_{c(p+1)} \equiv 2 \pmod{p}$ but $V_{c(q+1)} \not\equiv 2 \pmod{q}$ for a prime $q \neq p$ unless $(q-1)/2 \mid c$. This prompts using an algorithm very similar to Pollard's $p-1$ method: compute V_N for the same values of N used by Pollard's method; if N embodies all primes up to the highest prime dividing $p+1$, then we expect $(V_N - 2, n) = p$.

There are two problems with the method as stated. First, the method only works if $A^2 - 4$ is a quadratic non-residue modulo p . Fortunately, for random A we expect that to hold with probability roughly $1/2$ (experiments verify this), and so we can find a good A by simply trying a few random values.

The second problem is how to compute the sequence V_m efficiently. We use the following identity, which ultimately follows from $\alpha_1\alpha_2 = 1$:

$$\begin{aligned} V_n V_m &= (\alpha_1^n + \alpha_2^n)(\alpha_1^m + \alpha_2^m) \\ &= \alpha_1^{n+m} + \alpha_2^{n+m} + (\alpha_1^{n-m} + \alpha_2^{n-m})(\alpha_1\alpha_2)^m \\ &= V_{n+m} + V_{n-m}. \end{aligned}$$

Using this identity we can mimic exponentiation using repeated squaring. Furthermore, all computations can be done modulo n , so that the values do not blow up.

5 Elliptic Curve Method

The trouble with Pollard's $p-1$ method and Williams' $p+1$ method is that we have to rely on the random fact that $p \pm 1$ has small prime factors. The number $p-1$ itself is simply the order of the group \mathbb{Z}_p^\times . What if we could pick another group related to p of different length?

Such groups appear in the form of *elliptic curves*, as suggested by Lenstra. An elliptic curve is the set of solutions of an equation of the form $y^2 = ax^3 + bx + c$. Each elliptic curve, along with a special point at infinity which we designate O , has an associated group (defined over any field). The group action is defined as follows: to calculate $X + Y$, draw a line between X and Y ; usually the line will touch the curve in a third point Z ; the sum is the conjugate point to Z (with opposite sign of y). This funny operation turns out to be associative. In terms of the (x, y) coordinates, addition is a rational mapping, that is the result is a quotient of two polynomials in all the inputs. Division by zero corresponds to the point at infinity, which is also the identity of the group.

How many points are on the curve mod p (i.e. where $x, y \in \mathbb{Z}_p$)? For a random x , we expect $ax^3 + bx + c$ to be a quadratic residue with probability about half, in which case we get two solutions for y ; if it's a quadratic non-residue, we get no solutions. Thus, we expect the number of points (not counting O) to be about p . The standard deviation is about \sqrt{p} , so we expect the number of points to be roughly $p \pm C\sqrt{p}$ for some small C .

Indeed, Hasse's theorem tells us that the number S of points, including the point at infinity, satisfies $|S - p| \leq 2\sqrt{p}$. For a random curve, we expect the size to vary quite uniformly along this range. This number replaces $p-1$, which is the number of points in the multiplicative group mod p .

How do we adapt Pollard's $p-1$ algorithm to this new setting? We start with a point a on the elliptic curve. Instead of raising it to powers,

we multiply it by repeated addition using the rational addition formula. If N divides the order of the elliptic curve modulo p but not the order of the elliptic curve modulo all other prime factors, then $Na = O$ modulo p but $Na \neq O$ modulo n/p . We mentioned earlier that the point O is obtained when the denominator is zero in the addition formula. In this case the denominator d will be zero only modulo p , and so $(d, n) = p$.

In slightly more detail, the addition formula requires us to calculate inverses of elements modulo n . This is done using the extended GCD algorithm: if $(n, x) = 1$ then this algorithm finds a, b such that $an + bx = 1$, and so b is the inverse of x . When we are about to reach a point which is the identity modulo p but not modulo n/p , the denominator in question will have non-trivial GCD with n , something which we can discover while running the GCD algorithm.

Why have we gone to all this trouble? The reason is that if we look at a large number of integers, we have a fair chance of finding one with only small factors — such numbers are called smooth. Explicitly, the probability that a number m will have all its prime factors at most $m^{1/u}$ is approximately $1/u^u$ (the exact expression is given by Dickman's function). If we take about u^u random elliptic curves and run the above algorithm up to $p^{1/u}$ (i.e. up to $n^{1/2u}$), the total running time will be

$$u^u p^{1/u} = \exp\left(u \log u + \frac{\log p}{u}\right).$$

We want to minimize this quantity. The minimum is obtained when the derivative $\log u + 1 - \log p/u^2$ is zero. Thus approximately $u^2 \approx \log p / \log u$, so that $\log u \approx \frac{1}{2} \log \log p$. We conclude that $u \approx \sqrt{2 \log p / \log \log p}$ and the total running time is about

$$\exp\left(u \log u + \frac{\log p}{u}\right) \approx \exp\left(\sqrt{2 \log p \log \log p}\right).$$

A detail which we did not cover is how to find a random elliptic curve with a point on it. One way is to start with a random point (x, y) and random a, b , and calculate c via $c = y^2 - ax^3 - bx$.

6 Continued Fractions

The next algorithm we present is based on continued fractions. The idea that we present can be developed more formally, and this line of work culminated in Shanks' algorithm SQUFOF.

The basic idea is to use good rational approximations to \sqrt{n} . For every b , we can find a a such that $|a/b - \sqrt{n}| \leq 1/b$. Partial convergents of the continued fraction of \sqrt{n} satisfy the stronger property $|a/b - \sqrt{n}| \leq 1/b^2$. This implies that a^2/b^2 is a good approximation to n :

$$\left| \frac{a^2}{b^2} - n \right| = \left| \frac{a}{b} - \sqrt{n} \right| \left| \frac{a}{b} + \sqrt{n} \right| \leq \frac{2\sqrt{n}}{b^2}.$$

We note that the inequality is only approximate if $a/b > \sqrt{n}$. Next, we multiply both sides by the denominator b^2 :

$$|a^2 - nb^2| \leq 2\sqrt{n}b^2.$$

The probability that the right-hand side is a square (with the correct sign; this corresponds to considering only the odd convergents) is about $\sqrt[4]{n}$. If it is $\pm r^2$, we obtain the equation

$$a^2 \equiv r^2 \pmod{n}.$$

How does this equation help us? Consider the RSA case. How many square roots can a number have? Modulo p it has two square roots, and modulo q it has two square roots, and so four in total. With probability one half, a and r will be equivalent modulo only one of the prime factors of n , say p . In that case, $p \mid a^2 - r^2$ but $q \nmid a^2 - r^2$, so that $(a^2 - r^2, n) = p$. The GCD is easily computable using the Euclidean algorithm, and so the resulting factorization method has complexity $\sqrt[4]{n}$.

There are some technicalities to consider: first, the partial convergents of the continued fraction can be rather large — indeed, they grow exponentially. Second, in order to compute the continued fraction naively, we need to go through high precision calculations, which could also be costly. The second problem is solved by noticing that the residuals are always of the form $\alpha + \beta\sqrt{n}$, and so they can be stored implicitly using such a representation.

An even better representation is a quadratic form $Ax^2 + Bx + C$, the current residue is a root of which. It is well-known that the coefficients A, B, C are bounded (that's the reason the continued fraction is eventually periodic), and one can actually calculate the next residual from the preceding one given quadratic form representation. This representation also allows us to calculate the partial convergents (in fact, we only need the numerators) modulo n , and so the algorithm is practical.

Shanks' SQUFOF is an algorithm based intuitively on these ideas but cast entirely in the language of quadratic forms and reductions thereof.

SQUFOF can guarantee that once a square is found, it will result in a non-trivial factorization. For more details, consult Square Form Factorization or Continued fractions and Parallel SQUFOF.

7 Quadratic Sieve

Pomerance's quadratic sieve also attempts to find two numbers x, y such that $x^2 \equiv y^2 \pmod{n}$, but using a completely different approach. For a positive number $x < n$, denote $\psi(x) = x^2 \pmod{n}$. This function defines a homomorphism, i.e. $\psi(xy) \equiv \psi(x)\psi(y) \pmod{n}$. Our general approach will be to find a set of integers x_i such that $\prod \psi(x_i)$ is a square. Since $\prod \psi(x_i) \equiv \prod x_i^2 \pmod{n}$, this will give us the required factorization.

How do we find a set of integers such that $\prod \psi(x_i)$ is a square? We will use the factorizations of $\psi(x_i)$. That might seem pointless, but we will only consider $\psi(x_i)$ with easy factorizations, more specifically B -smooth numbers (number all of whose prime factors are at most B). How do these factorizations help us? An integer is square if in its prime factorization, all the exponents are even. This leads us to consider *exponent vectors*, which give for each $\psi(x_i)$ the exponents of all primes $\leq B$. In fact, we only care about the parity of the exponents, and so the exponent vectors are bit vectors of length $B/\log B$ (the approximate number of primes).

Multiplying two numbers corresponds to adding their exponent vectors. Therefore, given a list of smooth numbers $\psi(x_i)$ and their exponent vectors, finding a set whose product is a square is tantamount to finding a subset of the exponent vectors summing to the zero vector. This much we can accomplish using Gaussian elimination! The number of exponent vectors needed to guarantee a non-trivial linear combination summing to zero is $B/\log B + 1$.

We make two comments: first, we can get easy squares by reusing the same number twice: indeed, $\psi(x_i)^2$ is a square. However, the resulting equation $x_i^4 \equiv x_i^4 \pmod{n}$ clearly does not lead to factorization. In general, there is no point in reusing the same number twice. Second, our exponent vectors only contain parities, so how do we calculate the square root? This can be done easily and quickly using a binary search approach involving only shifts, adds and comparisons.

Let us describe the algorithm so far: we pick random numbers and test them for B -smoothness; this can be done by trying to factor them (a better method will be described later). When we have B of them, we use Gaussian elimination to find a subset of them $\{x_i\}$ that multiplies to a square. We

then get an equation $\prod x_i^2 \equiv \prod \psi(x_i) \pmod{n}$. We attempt to factor n by computing the GCD $(\prod x_i - \sqrt{\prod \psi(x_i)}, n)$, which we expect to be non-trivial half the time (for the RSA case).

When picking random numbers, it is best to pick them quite close to \sqrt{n} , since $\psi(\lfloor \sqrt{n} \rfloor + k) \approx 2k\sqrt{n} + k^2$ will be quite small by itself. For simplicity, let's assume that these numbers are $O(\sqrt{n})$; in practice they are somewhat bigger, but this can be avoided using different ψ 's (this is known as the MPQS, described below).

How should we choose B ? If $B = n^{1/2u}$, then the total running time of the algorithm is $n^{1/2u}u^u + n^{3/2u}$. This is an expression very similar to the one we got for the ECM, with essentially the same solution, under the substitution $p = \sqrt{n}$. Hence the running time (ignoring the linear algebra part) is

$$\exp(\sqrt{\log n \log \log n}).$$

Using efficient $O(N^2)$ linear algebra (see later), the linear algebra part has similar running time. Note that contrary to the ECM, the running time does not depend on p .

So far we've explained the quadratic part of the algorithm's name; what about the sieve? The sieve is an efficient way of finding smooth numbers. One could test for smoothness by trial division, but this is quite slow. The correct way is to use a sieve somewhat analogous to Eratosthenes'. Consider some small prime p . When does p divide $\psi(\lfloor \sqrt{n} \rfloor + k)$? Recall that

$$\psi(\lfloor \sqrt{n} \rfloor + k) = \lfloor \sqrt{n} \rfloor^2 - n + 2\lfloor \sqrt{n} \rfloor k + k^2.$$

This is a quadratic polynomial in p . If we solve this quadratic equation modulo p , then we can list all k such that $p \mid \psi(\lfloor \sqrt{n} \rfloor + k)$ by taking each of the two solutions K modulo p and jumping ahead p steps at a time. In order to find smooth integers this way, we can make a list of $\psi(\lfloor \sqrt{n} \rfloor + k)$ for a range of k 's, go over all primes p , for each prime go over all the values of k corresponding to multiples of p , and divide them by p . When we've reached our bound B , integers which have been reduced to 1 are smooth. We can then refactor them to calculate the exponent vector.

A more efficient method of executing the sieve without repeated divisions (which are time consuming) subtracts $\log p$ from an initial $\log \psi$ (or its estimate) instead of dividing by p . Numbers whose remaining value is small are probably smooth.

In practice, two further optimizations are made. First, just as in the $p-1$ algorithm, we can have two smoothness thresholds, i.e. require that the numbers will factor almost completely within a bound B_0 , perhaps with

an additional prime factor within B_1 . We then “pair” numbers with the same additional prime factor, or reduce even more general “cycles”.

Second, the matrix we need to solve in the linear algebra part is sparse, and in that case there are methods that outperform Gaussian elimination, including “block Lanczos”. We note that there are also theoretic sub-cubic methods (corresponding to the matrix multiplication exponent ω ; the current champion is $B^{2.376}$), but these aren’t efficient in practice.

A note about MPQS, used to reduce the size of integers required to be smooth, is in order. Instead of considering only $\psi(x) = x^2 - n$, we consider quadratic polynomials of the form $f(x) = Ax^2 + Bx + C$ with $A = \alpha^2$ and discriminant $B^2 - 4AC = n$. We can complete the square:

$$f(x) = (\alpha x)^2 + Bx + C = \left(\alpha x + \frac{B}{2\alpha}\right)^2 - \frac{B^2}{4A} + C = \left(\alpha x + \frac{B}{2\alpha}\right)^2 - \frac{n}{(2\alpha)^2}.$$

We conclude that

$$(2\alpha)^2 f(x) \equiv (2Ax + B)^2 \pmod{n}.$$

If $A \approx \sqrt{n}/2x$ and B is small then $4Af(x) \pmod{n}$ is $O(\sqrt{n})$, i.e. it is small and so has higher chance to be smooth. Since $4A = (2\alpha)^2$ is a square, we can use this just like the relations we’ve described earlier.

How do we find such A, B, C ? Choose a bound for x , and now our goal is to get $A \approx \sqrt{n}/2x$ and B small. Find a prime α close to $\sqrt[4]{n}/\sqrt{2x}$ (using a probabilistic prime checking algorithm), such that n is a quadratic residue with respect to α (this happens with probability half), say $n \equiv B^2 \pmod{\alpha}$ (square roots modulo primes can be effectively found), where we can assume B is odd (the sum of the square roots is α so one is even and the other is odd); note that $B = O(\sqrt[4]{n})$. Thus there exists C such that $B^2 - AC = n$. Since B is odd, $B^2 \equiv 1 \pmod{4}$. If $n \equiv 1 \pmod{4}$ then $4 \mid C$ (because A is odd) and we’re done. If $n = 4m + 3$, then $AC \equiv 2 \pmod{4}$, and so $2C$ is integral. This means that we can use the polynomial $4Ax^2 + 4Bx + 4C$ instead.

We conclude with a toy example, illustrating the factoring of 77:

$$\begin{aligned} 20^2 &= 400 \equiv 15 = 3 * 5 \pmod{77}, \\ 26^2 &= 676 \equiv 60 = 2^2 * 3 * 5 \pmod{77}, \\ 520^2 &= (20 \cdot 26)^2 \equiv (2 \cdot 3 \cdot 5)^2 = 30^2 \pmod{77}, \\ (520 - 30, 77) &= (490, 77) = 7. \end{aligned}$$

8 Number-Field Sieve

In the quadratic sieve we defined a homomorphism ψ and looked for numbers x_i such that $\psi(x_i)$ is a square. We then used an equation $\prod x_i^2 \equiv \prod \psi(x_i) \pmod{n}$ in order to try to factor n . The square on the left came for free, but we had to work hard for the square on the right: we needed to gather enough smooth numbers $\psi(x_i)$ so that we could find a subset multiplying to a square using linear algebra. Pollard's NFS uses two different homomorphisms, and this time there is no square coming for free. However, the size of the numbers that are required to be smooth is lowered from \sqrt{n} to about $\exp((\log n)^{2/3}(\log \log n)^{1/3})$, which makes that event much more probable.

Let d be a small integer, and $m = \lfloor n^{1/d} \rfloor$. There exists a d -degree integer polynomial P with “small” coefficients satisfying $P(m) \equiv 0 \pmod{n}$ (we will later construct it explicitly). Choose a root α of $P(m)$, and consider the number field $F = \mathbb{Z}[\alpha]$. Any number in F can be written as a polynomial in α . Since $P(m) \equiv 0 \pmod{n}$, we can think of m as $\alpha \pmod{n}$. More explicitly, the operation of substituting m for α is a homomorphism from F to \mathbb{Z}_n . Denote this operation by ϕ .

Define $\alpha(x, y) = x + \alpha y$ and $\beta(x, y) = x + my$. Since ϕ is a homomorphism, we have

$$\phi\left(\prod \alpha(x_i, y_i)\right) \equiv \prod \beta(x_i, y_i) \pmod{n}.$$

Our goal is clear now: we would like to find pairs (x, y) such that both $\alpha(x, y)$ and $\beta(x, y)$ are smooth, and then we can attempt to factorize n just like in the quadratic sieve. It is clear what smoothness means for $\beta(x, y)$, but what does it mean for $\alpha(x, y)$?

The norm operator is defined for algebraic number fields as the product of all conjugates (conjugates are obtained by replacing α by any other root of P). The norm is multiplicative, and so if $z \in F$ is a square then so is $N(z)$. The converse is almost true: we can add some more “statistics” (quadratic characters) which are also guaranteed zero for squares, and given we take enough of them, a number whose norm is square and all of whose statistics are zero is most probably square. These statistics are homomorphisms from F to \mathbb{Z}_2 , so they fit nicely to our scheme. The final ingredient, square root extraction in F , is arcane (F need not be a unique factorization domain!) but possible.

So smoothness for an element of F corresponds mainly to smoothness of its norm. If $P(t) = \sum c_i t^i$ then it turns out that the norm of $x - \alpha y$ is $\sum c_i x^i y^{d-i}$ (recall that $d = \deg P$), and this number is small if d, c_i, x, y are

small. In order to explain exactly how small, we need to present a sample polynomial P . Writing n in base m is tantamount to finding an expression $n = \sum c_i m^i$ with $0 \leq c_i < m$. Note that since $m = \lfloor n^{1/d} \rfloor$, we can convert this expansion into a degree d polynomial.

If $|x|, |y| \leq B$ for some bound B then $|\alpha(x, y)| \leq (d + 1)mB^{d+1}$ and $|\beta(x, y)| \leq (m + 1)B$. Since we want both $\alpha(x, y)$ and $\beta(x, y)$ to be smooth, we can think of them as one big number of size roughly $dm^2 B^{d+2}$ which is required to be smooth. One can now calculate the optimal choices of B and d , which are

$$B = \exp\left(\left(\frac{8}{9} \log n\right)^{1/3} (\log \log n)^{2/3}\right),$$

$$d = \exp\left(\sqrt{2/\left(\frac{8}{9}\right)^{1/3}} (\log n / \log \log n)^{1/3}\right).$$

The resulting running time is

$$\exp\left(\left(\frac{64}{9} \log n\right)^{1/3} (\log \log n)^{2/3}\right).$$

This is significantly faster than any other known method.

We mention in passing that for some numbers, a polynomial P can be found which is sparse, and this leads to a better constant in the exponent (replacing $64/9$). The general method is called the GNFS, and the special method in which P is sparse is called the SNFS. In fact, even for the GNFS the constant can be reduced somewhat employing trickery.

More material on the NFS can be found in Pomerance's *The Number Field Sieve*, and in the book *The Development of the Number Field Sieve*.

9 Summary

We've presented several different factorization methods. Let us summarize them and their running time, where n is an integer, and p is its smallest prime factor:

- Pollard's rho algorithm — $O(\sqrt{p})$. Finds $x \equiv y \pmod{p}$ in a pseudo-random sequence.
- Pollard's $p - 1$ algorithm — $O(P)$, where P is largest prime factor of $p - 1$. Calculates $(x^{P!} - 1, n)$ for iteratively increasing P .
- Williams' $p + 1$ algorithm — $O(P)$, where P is largest prime factor of $p + 1$. Calculates $(\text{Tr } \alpha^{P!} - 2, n)$ for iteratively increasing P and an element α of order $p + 1$ in some quadratic field.

- Lenstra’s ECM — $\exp(\sqrt{2 \log p \log \log p})$. Same as Pollard’s $p - 1$, replacing \mathbb{Z}_p^\times with a random elliptic curve with smooth size modulo p .
- Shanks’ SQUFOF — $O(\sqrt[4]{n})$. Uses reduction of quadratic forms to find $x^2 \equiv y^2 \pmod{n}$, then calculates $(n, x \pm y)$.
- Pomerance’s QS — $\exp(\sqrt{\log n \log \log n})$. Finds a host of x such that $\psi(x) = x^2 \pmod{n}$ is smooth (by sieving). Solves a set of linear equations to find a square of the form $\prod \psi(x)$, which is equivalent to $\prod x^2$ modulo n . Proceeds as in SQUFOF.
- Lenstra’s NFS — $\exp(c(\log n)^{1/3}(\log \log n)^{2/3})$. Defines a number field with a homomorphism ϕ into \mathbb{Z}_n^\times . Finds smooth number, in the number field and \mathbb{Z}_n^\times , which are equivalent under ϕ . Solves a set of linear equations to find a set which multiplies to squares on both sides. Proceeds as in SQUFOF and QS.

The methods we’ve presented cannot, as yet, be analyzed formally as stated. However, variants thereof, with the same asymptotic running time, have been analyzed successfully. A variant of the quadratic sieve was analyzed by Pomerance in *Fast, rigorous factorization and discrete logarithm algorithms*. The fastest rigorously analyzed factoring algorithm replaces squares with arbitrary binary quadratic forms. This algorithm is analyzed by Lenstra and Pomerance in A Rigorous Time Bound for Factoring Integers. It runs in time and space $\exp((1 + o(1))\sqrt{\log n \log \log n})$.

A complementary question is the Discrete Logarithm problem, which is the basis of many practical cryptographic primitives (such as the Diffie-Hellman key exchange algorithm). Given a prime p , a generator g of \mathbb{Z}_p^\times and $x \in \mathbb{Z}_p^\times$, find an exponent y such that $x \equiv g^y \pmod{p}$. Some of the algorithms for solving this problem use ideas similar to the ones used in integer factorization.