

The Number Field Sieve: An Extended Abstract

Joshua E. Hill

June 24, 2010

1 Motivation

So, one day you're walking down the street minding your own business when BANG! A 768-bit number hits you in the head. ¹ The offending number?

$n = 123018668453011775513049495838496272077285356959533479219732245215172640050726$
 $365751874520219978646938995647494277406384592519255732630345373154826850791702$
 $6122142913461670429214311602221240479274737794080665351419597459856902143413$

You perform your normal set of cursory checks; checks that you perform on all numbers that hit any portion of your body:

- You check for “small” prime divisors, up to $\log(n)$. In this case, this means dividing it by the first 99 primes (up to 521). In this case, none of these primes divide n .
- You check to see if n is prime, using a probable prime check such as repeated invocations of the Miller-Rabin primality test using randomly selected bases. A few iterations later (indeed, very likely 1 iteration later) you establish that n is certainly a composite number.
- You check to see if n is a power of a single prime.
- You check to see if n has any “small” factors (less than 20 digits) by using Pollard’s rho algorithm.
- You attempt to find bigger “small” factors (20-25 digits) by using the elliptic curve factorization method.
- You note that the n is 232 digits long, which certainly puts it well beyond the range where the quadratic sieve outperforms the number field sieve (numbers smaller than 110-120 digits).
- You note that n does not appear to be of an “easy” form of $r^e - s$ with r , e , and s small, so you can’t apply the special number field sieve.

¹Or you download it from <http://www.rsa.com/rsalabs/node.asp?id=3723>

Thus, you are left with a number that is best approached using the General Number Field Sieve. You call some colleagues. It's going to be a long night.

2 Introduction

The General Number Field Sieve (GNFS) is (conjectured to be) the fastest known factoring algorithm for factoring hard composite numbers [5] (numbers that are not of one of a variety of special forms. For numbers of various special forms, there are various special factoring algorithms that operate faster than the GNFS, possibly much faster, depending on the form of the number!)

The GNFS is a factoring algorithm that is similar to the continued fraction factorization algorithm (in that the form of the auxiliary numbers is similar) and the quadratic sieve [4] (in that a sieving step is used to find smooth auxiliary numbers which are turned into relations and then input into a matrix reduction step to find subsets of auxiliary numbers that can be multiplied together to get perfect squares). The GNFS is asymptotically faster than either of these techniques, but at the cost of additional conceptual complexity.

As suggested above, the GNFS is comprised of a series of distinct steps. For each step, I will provide a summary of the step, note the algorithms used, provide some indication of the complexity of that step, and finally provide an example of the run time for this step during the recent RSA-768 challenge team's computation. [1]

3 The General Number Sieve Algorithm

The GNFS algorithm acts on a hard composite integer n that is to be factored and produces a divisor of n . It operates by finding congruent squares mod n , which lead to a non-trivial factorization of n . This approach is a generalization of the idea that all odd composite numbers can be represented as a difference of two squares, thus providing a non-trivial factorization. One can't efficiently use this fact to factor, as simply choosing values for x and checking to see if $n - x^2$ is a square is actually less efficient than trial division for most numbers.

One can extend this approach by noting that if $x^2 \equiv y^2 \pmod{n}$ non-trivially (that is, $x \not\equiv \pm y \pmod{n}$) then $\gcd(x - y, n)$ and $\gcd(x + y, n)$ are non-trivial factors of n . If one can generate random values with this relationship, the probability that the resulting values have the trivial relationship is less than $\frac{1}{2}$, so by generating several such values we can factor n . We don't randomly generate values, but heuristically there isn't a particular reason that the candidates generated through our process should be any different, so we expect this probability to be roughly the same.

The question remains how to generate such values of x and y . Both the Quadratic Sieve and the General Number Field Sieve use a two step process to arrive at these numbers. First, numbers of a particular form are sieved for smoothness (and perhaps some other properties that we'll discuss later in the case of the GNFS algorithm), and then a matrix reduction step is used to find subsets of the smooth numbers that can be multiplied together to form the candidate congruent squares. We expect roughly half of these candidates to be non-trivial, so by generating many such candidates we can have a very high probability that we will find a non-trivial factorization.

Though the broadest outline of these two factoring algorithms is similar, they differ significantly in their detail. Most importantly, the Quadratic Sieve operates over the integers only (over $\mathbb{Z} \times \mathbb{Z}$). The General Number Field Sieve operates over the integers and over the ring $\mathbb{Z}[\alpha]$ (so over $\mathbb{Z} \times \mathbb{Z}[\alpha]$, where α is a root of an artfully chosen polynomial $f(X)$). For the final step (where the result should be within the integers) we reduce using the map

$$(\mathbb{Z} \times \mathbb{Z}[\alpha]) \xrightarrow{\phi} (\mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z})$$

where the first (integer) component is simply reduction mod n , and the second component is a homomorphism that we'll call ϕ_2 and establish later.

This difference introduces a great deal of complexity to the general number field sieve that is not present in the quadratic sieve, but it is this difference that allows the General Number Field Sieve to asymptotically out-perform the Quadratic Sieve.

3.1 Selecting the ring $\mathbb{Z}[\alpha]$

In order to start, we must first choose the parameter α . This choice is hugely significant to the runtime of the algorithm, so a good choice of α is vital. We'll choose an irreducible polynomial $f(X)$ (irreducible over \mathbb{Z}) of degree d ($f(X)$ and d are parameters) and let α be a root of this polynomial some extension of \mathbb{Q} . So long as $f(X)$ is monic, we have an easy way of representing $\mathbb{Z}[\alpha]$:

$$\mathbb{Z}[\alpha] \cong \mathbb{Z}[X]/(f(X)) \cong \mathbb{Z} \cdot 1 \oplus \mathbb{Z} \cdot \alpha \oplus \dots \oplus \mathbb{Z} \cdot \alpha^{d-1}$$

Sadly, we don't necessarily have the condition that $\mathbb{Z}[\alpha] = \mathcal{O}_{\mathbb{Q}(\alpha)}$, but we'll deal with this sad fact later.

In order to answer our factoring question (which is, after all, a question about the integers), we need to fix some homomorphism that moves values in $\mathbb{Z}[\alpha]$ into the integers (or something sufficiently similar to the integers, in any case), $\phi_2 : \mathbb{Z}[\alpha] \rightarrow \mathbb{Z}/n\mathbb{Z}$. It is sufficient to establish the value of $\phi_2(\alpha) = m$ (a parameter) and then extend \mathbb{Z} -linearly. We must then choose a value for m so that $f(m) \equiv 0 \pmod{n}$.

This still leaves quite a lot of flexibility for the choice of f (and thus d) and m . One option is to choose d ahead of time (generally from the integers between 3 and 10, though the optimal choice for d tends toward infinity as n tends toward infinity), and then choose $m = \lfloor \sqrt[d]{n} \rfloor$. We can then represent n as a number base m , that is $n = \sum_{i=0}^d c_i m^i$ where c_i is some integer from 0 to $m - 1$. Now let $f(X) = \sum_{i=0}^d c_i X^i$. Clearly $f(m) = n \equiv 0 \pmod{n}$ so this trivially satisfies the congruence requirement. Now, we can assume that this $f(X)$ is irreducible (as if not, we are done: if $f(X) = h(X)g(X)$ then $n = f(m) = h(m)g(m)$, a non-trivial factorization of n).

This does work, but it is generally not the optimal choice for $f(X)$. In the RSA-768 challenge, the team spent 40 processor years² evaluating more than 2^{60} different polynomials in order to select the polynomial f . This selection was made based on various criteria for the polynomial, mainly dealing with the resulting smoothness probability over the relevant range of values (the size of the coefficients,

²quoted processor years are normalized to refer to a single 2.2 GHz AMD Opteron

the number of roots modulo small primes, smoothness of the leading coefficients and the number of real roots).

For reference, the polynomial chosen for the algebraic representation by the RSA-768 challenge team was

$$\begin{aligned}
 f_1(X) = & 265482057982680X^6 \\
 & + 1276509360768321888X^5 \\
 & - 5006815697800138351796828X^4 \\
 & - 46477854471727854271772677450X^3 \\
 & + 6525437261935989397109667371894785X^2 \\
 & - 18185779352088594356726018862434803054X \\
 & - 277565266791543881995216199713801103343120
 \end{aligned}$$

The RSA-768 challenge team also chose a (degree 1) polynomial to represent the integer side of the calculation.

3.2 Sieving and Linear Algebra

Now that we have specified the ring $\mathbb{Z}[\alpha]$, we can specify the sieving process. The sieving step operates on elements of $\mathbb{Z} \times \mathbb{Z}[\alpha]$, locating values that are smooth in their respective rings.

We'll first generate a universe $U = \{(a, b) \in \mathbb{Z} \times \mathbb{Z} : |a| \leq u, 0 < b \leq u, \gcd(a, b) = 1\}$ with some positive parameter u . We'll fix u later; its value will establish the total number of values that we'll examine in our sieve. We'll sieve³ values of the form $(a - bm)$ and $(a - b\alpha)$ (for all $(a, b) \in U$) for smoothness as described below.

We then process the relations derived from the smooth integer / algebraic integer pairs within the linear algebra step.

3.2.1 On the Utility of Smooth Integers

We'll define an element $(x, \gamma) \in \mathbb{Z} \times \mathbb{Z}[\alpha]$ as y -smooth if the factorization of x and $N(\gamma)$ (the norm of γ) involves only primes less than a parameter y . The choice of y will be discussed later.

The general notion underlying our interest in smooth numbers is two-fold. This is easiest to explain in the context of the integer portion of our computation: we view each of our y -smooth integers as a vector in a \mathbb{F}_2 vector-space where the primes act as the basis elements. Formally, we enumerate the primes less than or equal to y as p_1, p_2, \dots, p_k (that is, $k = \pi(y)$, so the $(k + 1)$ -st prime is larger than y). We represent the y -smooth integer $x \in \mathbb{Z}^+$ by its prime factorization, $x = \prod_{i=1}^k p_i^{j_i}$ and then map the value to our vector space

$$\prod_{i=1}^k p_i^{j_i} \rightarrow \sum_{i=1}^k (j_i \pmod{2}) \cdot p_i \cong (j_1 \pmod{2}, j_2 \pmod{2}, \dots, j_k \pmod{2})$$

³Look at that! "sieve" can be used as a transitive verb!

We are ultimately going to multiply a subset of our selected smooth integers in order to construct a square (otherwise said, a subset of our selected smooth integers will multiply to have only even exponents). In our vector space, this is equivalent to finding a subset of vectors that can be added to get 0. Said in the language of linear algebra, we are looking for a linear dependency within our set of vectors. Once we find a linear dependency, we can construct a square integer by multiplying the smooth integers that correspond to the vectors present in the sum that resulted in the linear dependency.

Taking the first k primes, this gives us a k -dimensional vector space, so by choosing more than k vectors, we are guaranteed to find at least one linear dependency. If we didn't limit ourselves to a fixed set of primes, we could have an infinite number of values with no linear dependency (as there are an infinite number of primes!).

To justify our interest in the first k primes (and thus y -smooth integers, rather than integers composed of some other set of k primes), there is a more probabilistic explanation: randomly selected integers with a particular large prime factor are much more rare than integers with a particular smaller prime factor. If we are randomly selecting integers, we are more likely to happen across instances where there are several numbers with the same small prime factor in common than numbers with the same large prime number in common. As we can only arrive at a linear dependency when at least two numbers share the same prime factor, this suggests that the small primes are more significant than the large prime factors in our search.

3.2.2 On the Utility of Smooth Algebraic Integers

The above argument is sufficient to find squares in the integers, but what about $\mathbb{Z}[\alpha]$? The answer is that it's more complicated.

We start out the same basic way. First, we formulate a way of specifying the primes of degree 1 that divide a particular $a - b\alpha$. First, if p (an integer prime) divides $N(a - b\alpha)$, then we expect a prime in $\mathbb{Z}[\alpha]$ over p . We can represent this prime as $\mathfrak{p} = (p; r_p) = (p, r_p - \alpha)$ where $r_p = ab^{-1} \pmod{p}$. Thus, for each a and b and prime p dividing $N(a - b\alpha)$, we can generate the degree 1 prime \mathfrak{p} over p through this relation.

Now that we can (in principal) enumerate the full set of possible prime divisors to $a - b\alpha$, we can restrict ourselves to purely the primes whose norms are less than y , enumerated as $\mathfrak{p}_1, \mathfrak{p}_2, \dots, \mathfrak{p}_l$ so we now have a finite list of primes to sieve over.

We might expect this to proceed just as the integers did from this point, but there are complications. Upon completing our linear algebra step, we again have a set of values such that $\prod_{(a,b) \in S} (a - b\alpha) = \beta$ where the powers for all primes in the product are even. "Grand", we think to ourselves, "a square!".

Sadly, this is not necessarily the case. The obstructions are as follows:

- a) If $\mathbb{Z}[\alpha] \neq \mathcal{O}_{\mathbb{Q}(\alpha)}$, then $\beta \mathcal{O}_{\mathbb{Q}(\alpha)}$ may not be the square of any ideal.
- b) Even if $\beta \mathcal{O}_{\mathbb{Q}(\alpha)}$ is the square of an ideal, that ideal may not be principal.
- c) Even if $\beta \mathcal{O}_{\mathbb{Q}(\alpha)} = (\gamma \mathcal{O}_{\mathbb{Q}(\alpha)})^2$ for some $\gamma \in \mathcal{O}_{\mathbb{Q}(\alpha)}$ (i.e., $\beta \mathcal{O}_{\mathbb{Q}(\alpha)}$ is the square of a principal idea), we may only have $\beta = \gamma^2$ up to multiplication by some unit of $\mathcal{O}_{\mathbb{Q}(\alpha)}$ (that is, $\beta = h\gamma^2$ where h is a unit of $\mathcal{O}_{\mathbb{Q}(\alpha)}$).

d) Even if $\beta = \gamma^2$ in $\mathcal{O}_{\mathbb{Q}(\alpha)}$, we may have $\gamma \notin \mathbb{Z}[\alpha]$.

We'll fully solve (d) first, and then bound the damage done by (a)-(c), and then talk about how to address these issues.

For (d), we can fairly quickly solve the problem by recalling that we can force values from $\mathcal{O}_{\mathbb{Q}(\alpha)}$ to be in $\mathbb{Z}[\alpha]$ by simply multiplying them by $f'(\alpha)$. In our case, we can force the square root of the algebraic component's square into $\mathbb{Z}[\alpha]$ by multiplying the algebraic component by $f'(\alpha)^2$. To maintain the necessary relationship between the integer and algebraic integer components, we then multiply the integer component by $f'(m)^2$.

We are left with the obstructions (a)-(c). We don't have direct solutions that nicely fix everything, but we can estimate the degree of the damage; we define a filtration based on:

- Let V_{-1} be the group generated by all elements of the form $(a - b\alpha)$ with $\gcd(a, b) = 1$.
- Let V_0 be the subgroup of V_{-1} such that if $\beta \in V_0$, then $\beta\mathcal{O}_{\mathbb{Q}(\alpha)}$ has only even exponents at the primes of $\mathbb{Z}[\alpha]$. This subgroup includes all of the candidates that can be produced by the sieve step.
- Let V_1 be the subgroup of V_0 such that if $\beta \in V_1$, then $\beta\mathcal{O}_{\mathbb{Q}(\alpha)}$ has only even exponents at all the primes of $\mathcal{O}_{\mathbb{Q}(\alpha)}$ (that is to say, $\beta\mathcal{O}_{\mathbb{Q}(\alpha)}$ is the square of an $\mathcal{O}_{\mathbb{Q}(\alpha)}$ -ideal). Said alternately, this is the subgroup of V_0 that does not suffer from obstruction (a).
- Let V_2 be the subgroup of V_1 where if $\beta \in V_2$, then $\beta\mathcal{O}_{\mathbb{Q}(\alpha)}$ is the square of a principal ideal of $\mathcal{O}_{\mathbb{Q}(\alpha)}$, say $\beta\mathcal{O}_{\mathbb{Q}(\alpha)} = (\gamma\mathcal{O}_{\mathbb{Q}(\alpha)})^2$ for some $\gamma \in \mathcal{O}_{\mathbb{Q}(\alpha)}$. Said alternately, this is the subgroup of V_1 that does not suffer from obstruction (b).
- Let V_3 be the subgroup of V_2 where if $\beta \in V_3$, then (by the above) $\beta\mathcal{O}_{\mathbb{Q}(\alpha)} = (\gamma\mathcal{O}_{\mathbb{Q}(\alpha)})^2$, and $\beta = \gamma^2$. Note $V_3 = V_0 \cap (\mathbb{Q}(\alpha)^*)^2$. Said alternately, this is the subgroup of V_2 that does not suffer from obstruction (c).

From these definitions, we have the filtration $V_0 \supset V_1 \supset V_2 \supset V_3$. By construction we have that $\dim_{\mathbb{F}_2} V_0/V_1$ measures the size of the obstruction (a). Similarly, $\dim_{\mathbb{F}_2} V_1/V_2$ measures the size of the obstruction (b) and $\dim_{\mathbb{F}_2} V_2/V_3$ measures the size of the obstruction (c). We are interested in examining the impact of all these obstructions on the candidates that we generate (the naïve application of the sieve step produces members of V_0 , but we want members in V_3 , which we can surely convert into solutions that certainly do not suffer from obstruction (d)).

Through a series of algebraic arguments we are able to arrive at the conclusion that $\dim_{\mathbb{F}_2} V_0/V_3 \leq \log n$. This result can be interpreted as giving the expected likelihood that a randomly selected candidate that lies in V_0 also lies in V_3 ; we expect that the proportion of these that lie in V_3 to be at least $\frac{1}{\log n}$.

We thus immediately have a fairly inelegant solution to this problem: just produce quite a lot of candidates and then run some test that will tell you which of your candidates is in the desired V_3 , and is thus truly a square.

We are left with the question of what this test is. If $x \in V_0$, we can verify that $x \in V_3$ by verifying that all characters $\chi : V_0/V_3 \rightarrow \mathbb{F}_2$ are trivial at x . It isn't reasonable to actually try all of the

characters, but we can note that these characters themselves form a vector space of the same dimension as V_0/V_3 , which is to say dimension $\leq \log n$. This implies that if we could find a spanning set for $\text{Hom}(V_0/V_3, \mathbb{F}_2)$ then we could test our candidates by using this spanning set to verify that our candidate is certainly within V_3 . Sadly, we don't have any obvious way of generating such a spanning set deterministically. Instead, we choose randomly from the space $\text{Hom}(V_0/V_3, \mathbb{F}_2)$ and hope for the best.

If a vector space is dimension w and we select $w + s$ random vectors from the space, the probability that our $w + s$ vectors span the vector space is $1 - 2^{-s}$. By choosing a sufficiently large number of random vectors from the space, we can make this as close to 1 as desired.

So, to apply this method, we need some way of generating random quadratic characters. In the integers, the Legendre symbol $\left(\frac{x}{p}\right)$ accomplishes this task so long as $p \nmid x$: we have $\left(\frac{x}{p}\right) = -1$ implies that x is not a square, and if x is not a square then $\left(\frac{x}{p}\right) = -1$ for half the primes p . Thus, we can modify this to be a quadratic character: $\chi_p(x) = -\frac{1}{2} \left(\left(\frac{x}{p}\right) - 1 \right)$.

We construct an analog for this function in $\mathbb{Z}[\alpha]$ by first noting that if q is a prime integer and a and b are selected as above and $\mathfrak{q} = (q, r_q)$ is degree 1, we can define a map $\pi : \mathbb{Z}[\alpha] \rightarrow \mathbb{F}_q$ as $\pi(\alpha) = ab^{-1} \pmod{q}$ (extended \mathbb{Z} -linearly), then define $\chi_{\mathfrak{q}}(x) : \mathbb{Z}[\alpha] \rightarrow \mathbb{F}_2$ as

$$\chi_{\mathfrak{q}}(x) = -\frac{1}{2} \left(\left(\frac{\pi(x)}{q} \right) - 1 \right)$$

We can make sure that this remains well behaved by choosing primes that can't possibly be factors of x , for example primes such as \mathfrak{q} where $N(\mathfrak{q}) > y$.

These quadratic characters can be considered randomly distributed as a consequence of the Chebotarev density theorem, so by selecting a suitable number of these primes, we can (very likely) span the set of quadratic characters.

Now, there are two ways of proceeding from here. We have our test, so we can simply check all of our candidate values against a suitable number of these quadratic characters. This is the approach followed by the RSA-768 challenge team; indeed, they had selected the polynomials to decrease the dimension of V_0/V_3 , which resulted in only having to exclude 52 of the total of 512 final candidates in the quadratic character test stage.

Another option would be to include these quadratic characters directly into the initial sieve step. We'll discuss this possibility in more detail in the next section.

3.2.3 The Sieve

In order to generate a sufficient supply of smooth numbers of the correct form, we sieve for them. For the integer and algebraic integer sides we establish a factor base. This can be abstractly thought of as a set of rules that establish which terms we allow within the candidates that are to "pass through" the sieve.

For the integer side, we want to include the list of allowed primes p_1, p_2, \dots, p_k along with the possible factor -1 (which is either present or not present, and so has exponent 1 or 0, respectively). We call this factor base B_1 .

For the algebraic integer side, the factor base is made up of the full set of allowable first-order primes p_1, p_2, \dots, p_l . If we are interested in avoiding the final quadratic character test, we can also select \hat{l} distinct quadratic characters $\chi_{q_1}, \dots, \chi_{q_{\hat{l}}}$ (where \hat{l} is chosen to be appropriately large to assure a small chance of error) as members of the factor base in the sense that each character will be applied to each component, and the result of the character will be stored in the same way that the power of each prime that is present is stored. (We don't expect to, in any sense, "divide" by the quadratic character elements, just store the results for every quadratic character and deal with them as if they were exponents; these will be stored within the "Quadratic Character Columns" of our final matrix). We call this factor base B_2 .

Now we apply the sieve. For each value in U we populate an integer table made up of the elements $a - bm$ and an algebraic integer table made up of elements $a - b\alpha$. We can quickly establish what power of which primes divide each entry and divide them out. Once we have proceeded through all the primes and prime powers in the factor bases, we are left with a set table of entries that are either units or non-units. If they are non-units, they are rejected as being insufficiently smooth. If they are units, they are accepted as y -smooth. In this process, we can store the "divided out" terms for each entry in our table, resulting in a complete factorization for each of the resulting smooth elements. We accept only values for (a, b) that result in smooth values both in the integers ($(a - bm)$ is smooth) and over the algebraic integers ($(a - b\alpha)$ is smooth).

Once we know which values are being accepted, we can optionally test these accepted values by processing the extra terms in the factor base that are not associated with prime factors (the -1 term for the integer side and the quadratic characters for the algebraic integer side), noting the resulting values as if they were exponents within the quadratic character columns.

We'll identify these smooth values using $U' \subset U$, the particular $(a, b) \in U$ values that passed the sieving step.

The resulting vectors for each candidate are referred to as "relations", as these are the vectors the move into the linear algebra step.

This seems a complicated procedure, but it can be accomplished quite efficiently, asymptotically using $u^{2+o(1)}$ operations total, where u is as defined in section 3.2.

In practice, this calculation can be hugely parallelized to the process scales well across clusters. The RSA-768 challenge team spent over 1300 processor years doing the sieving step, but this investment included significant over-sieving (they produced approximately twice as many candidates as were strictly necessary). This was done to increase the options for the quicker but more delicate linear algebra step.

By the end of the sieving, the RSA-768 challenge team accumulated 64,334,489,730 relations, each represented by 150 bytes (a total of about 9TB of relations). As a consequence of the distributed sieving algorithm used, these relations contained a significant proportion of duplicates (27.4%), which should be discarded. At this point in the calculation, any relation that contained a unique prime (a prime not present in any of the other relations) was also discarded, as were cliques (small sets of relations that share primes that are not in other relations). When this discard step was complete there were 2,458,248,361 relations left (only 3.4% of the initial set of relations!) involving 1,697,618,199 distinct primes. This discard step took 10 processor days .

3.2.4 Linear Algebra

We now have a set of relations that we will form into a matrix and look for linear dependencies between the relations. Once we find such dependencies, we will nearly be done. The naïve approach to this task would be to use Gaussian Elimination. If we had t relations each of size s , this approach would result in a runtime of $O(t^2s)$, which would ruin our ultimate asymptotic performance. Hence, we adopt more advanced algorithms, namely the block Wiedemann or (Montgomery’s variation of the) block Lanczos algorithms, both of which run in time proportional to the dimension and the weight of the matrix (the sum of the hamming weight of the vectors).

In practice, one wants to construct a matrix that will contain a suitably high number of dependencies. This allows us to accept the inevitable loss of some of the resulting linear dependencies. These losses could either be due to the linear dependency corresponding with a trivial relation between the two squares (which has probability heuristically $< \frac{1}{2}$), or due to the expected loss of relations that correspond to algebraic numbers V_0 but not in V_3 (if the quadratic characters aren’t included within the relations). We would also like this matrix to be nearly optimal for whatever algorithm has been selected to reveal the linear dependencies. This matrix creation step is called the merging process.

This merging process took the RSA-768 challenge team about 4 processor years, and resulted in a $192,796,550 \times 192,795,550$ -matrix, which had a weight of 27,797,115,920. The advantage to all this pre-computation is that the matrix step took “only” 460 processor years to compute.

3.3 Square Roots and Other Final Calculations

Once relations are found (and if not included within the factor base, the quadratic character tests rule out non-square results) and we resolve issue (d), we are left with candidates of the form

$$\left(f'(m)^2 \prod_{(a,b) \in S} (a - bm), f'(\alpha)^2 \prod_{(a,b) \in S} (a - b\alpha) \right)$$

where S is a particular subset of U' that is associated with a linear dependency identified in the linear algebra step.

The first value in this tuple is guaranteed to be a square integer, and the second element is very likely a square (recall the quadratic character test is probabilistic!). We can trivially calculate the square root of the integer term (we’ll call this square root x'). Calculating the square root of the algebraic integer is (predictably) rather complicated.

The methods of Montgomery [2] or Nguyen [3] can be used to extract this square root.

Once the square root, γ , is calculated, a quick application of the homomorphism developed at the start gives us

$$\phi(x', \gamma) = (x, \phi_2(\gamma))$$

(where x is just the reduction of $x' \pmod{n}$)

All our work assures us that $x^2 \equiv \phi_2(\gamma)^2 \pmod{n}$. We then check to see if the resulting relationship is trivial, that is we check to see if $x = \pm \phi_2(\gamma) \pmod{n}$; we (heuristically) expect that this occurs less than half the time, so having several candidates is important even at this late stage!

If this congruence is not trivial, then n has non-trivial factors $\gcd(n, x - \phi_2(\gamma))$ and $\gcd(n, x + \phi_2(\gamma))$.

The RSA-768 challenge team had 512 linear dependencies identified from the linear algebra step. Checking the corresponding candidates using the quadratic character excluded 52 of these linear dependencies. They then used Montgomery's method for taking square roots, which took less than a processor-day per solution.

3.4 Performance and Parameter Summary

Table 1: Parameter Summary

Parameter	Description
n	The hard composite number to be factored (§3)
$f(X)$	An irreducible polynomial used to generate the number field we use (§3.1)
α	A root of $f(X)$ (§3.1)
d	The degree of $f(X)$ (§3.1)
m	An integer root of $f \pmod n$ (§3.1)
u	A parameter that establishes the size of our sieving universe (§3.2)
y	The smoothness bound for the sieve (§3.2.1)
\hat{l}	The number of quadratic characters used test for solutions not in V_3 (§3.2.3)

We can make swing at choosing parameters in order to establish the performance of the algorithm. These estimates are largely based on conjecture and heuristic arguments. We seek to maximize the number of smooth numbers available to us; by applying some heuristics and minimizing the resulting equation, we find that

$$d \approx \left(\frac{3 \log n}{\log \log n} \right)^{\frac{1}{3}}$$

The asymptotic cost of the algorithm is on the order of $u^{2+o(1)} + y^{2+o(1)}$ as $n \rightarrow \infty$, where the first term is the cost of the sieve operation and the second term is the cost of the matrix operation. For the purpose of simplification, we could desire to spend roughly equal time in each task, at which point we would choose these parameters so that

$$\log u \approx \log y \approx \left(\frac{8}{9} \right) (\log n)^{\frac{1}{3}} (\log \log n)^{\frac{2}{3}}$$

With these choices, we arrive at the (conjectured) asymptotic run time of

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

A summary of the run time of the high level tasks involved in the RSA-768 challenge team's calculation is available in Table 2 .

Table 2: RSA-768 Processing Time Summary

Step	Normalized Processor Years	Percentage of Task
Selection of $f(X)$ (§3.1)	40	2.2%
Sieving (§3.2.3)	1300	71.8%
Discarding excess relations (§3.2.3)	0.03	< 0.01%
Merging (§3.2.4)	10	0.6%
Linear Algebra (§3.2.4)	460	25.4%
Square root (§3.3)	0.01	< 0.01%
Total	1810.04	

4 Profit!

Roughly 2.5 years later after that harrowing day when that pesky number hit you, you finally have closure.

$$\begin{aligned} \text{RSA-768} = & 3347807169895689878604416984821269081770479498371376856891 \\ & 2431388982883793878002287614711652531743087737814467999489 \times \\ & 3674604366679959042824463379962795263227915816434308764267 \\ & 6032283815739666511279233373417143396810270092798736308917 \end{aligned}$$

It's time to go to the beach.

References

- [1] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen Lenstra, Emmanuel Thomé, Joppe Bos, Pierrick Gaudry, Alexander Kruppa, Peter Montgomery, Dag Arne Osvik, Herman te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit rsa modulus. *Cryptology ePrint Archive*, Report 2010/006, 2010. <http://eprint.iacr.org/>.
- [2] Peter L. Montgomery. Square roots of products of algebraic numbers. In *Mathematics of Computation 1943–1993*, pages 567–561. American Mathematical Society, 1994.
- [3] Phong Nguyen and Ecole Normale Sup'erieure. A montgomery-like square root for the number field sieve. In *In Proc. of ANTS-III, volume 1423 of LNCS*, pages 151–168. Springer-Verlag, 1998.
- [4] Carl Pomerance. Smooth numbers and the quadratic sieve. In J.P. Buhler and P. Stevenhagen, editors, *Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography*. Mathematical Sciences Research Institute Publications, Cambridge University Press, 2008.

- [5] Peter Stevenhagen. The number field sieve. In J.P. Buhler and P. Stevenhagen, editors, *Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography*. Mathematical Sciences Research Institute Publications, Cambridge University Press, 2008.