# The number field sieve for integers of low weight

Oliver Schirokauer

Department of Mathematics
Oberlin College
Oberlin, OH 44074

**Abstract.** We define the weight of an integer $N$ to be the smallest $w$ such that $N$ can be represented as $\sum_{i=1}^{w} \epsilon_i 2^{c_i}$, with $\epsilon_1,\ldots,\epsilon_w \in \{1,-1\}$. Since arithmetic modulo a prime of low weight is particularly efficient, it is tempting to use such primes in cryptographic protocols. In this paper we consider the difficulty of the discrete logarithm problem modulo a prime $N$ of low weight, as well as the difficulty of factoring an integer $N$ of low weight. We describe a version of the number field sieve which handles both problems. Our analysis leads to the conjecture that, for $N \to \infty$ with $w$ fixed, the worst-case running time of the method is bounded above by $\exp((c+o(1))(\log N)^{1/3}(\log\log N)^{2/3})$ with $c<((32/9)(2w-3)/(w-1))^{1/3}$ and below by the same expression with $c=(32/9)^{1/3}((\sqrt{2}w-2\sqrt{2}+1)/(w-1))^{2/3}$. It also reveals that on average the method performs significantly better than it does in the worst case. We consider all the examples given in a recent paper of Koblitz and Menezes and demonstrate that in every case but one, our algorithm runs faster than the standard versions of the number field sieve.

**Mathematics subject classification (2000):** Primary: 11Y16 Secondary: 11T71, 11Y05, 11Y40

**Keywords:** discrete logarithm, integer factorization, number field sieve

**Acknowledgement.** The author wishes to thank Neil Koblitz and Alfred Menezes for inquiring about a low-weight version of the number field sieve and for their subsequent comments on this paper.

## 1. Introduction.

We define the weight of an integer $N$ to be the smallest $w$ with the property that there exists a representation of $N$ as the sum

$$\sum_{i=1}^{w} \epsilon_i 2^{c_i},$$

with $\epsilon_1, \ldots, \epsilon_w \in \{1, -1\}$. In 1999, Solinas observed that arithmetic modulo a prime can be made more efficient if the prime is chosen to be of small weight [15]. More recently, Koblitz and Menezes, in their investigation of the impact of high security levels on cryptosystems which are based on the Weil and Tate pairings on elliptic curves over finite fields, have asked whether there is a downside to using a field in this context whose characteristic has small weight ([7]). In particular, they raise the concern that discrete logarithms modulo a prime $N$ might be easier to compute with the number field sieve (NFS) when $N$ is of low weight than they are in general. The increase in vulnerability may then offset any efficiency advantages gained by implementing the protocol with a low weight prime.

The concern about primes of low weight is well founded. Indeed, if $N$ is a prime of the form $br^c + s$, where $b, r$, and $s$ are all small, then the special number field sieve (SNFS) can be used to compute discrete logarithms mod $N$. For $b, r$, and $s$ bounded and $c \to \infty$, this algorithm has a conjectural running time of

$$L_N[1/3; (32/9)^{1/3} + o(1)],$$

where

$$L_N[s; c] = e^{c(\log\,N)^s (\log\log\,N)^{1-s}}.$$

By comparison, the general NFS has a conjectural running time of

$$L_N[1/3; (64/9)^{1/3} + o(1)],$$

for $N \to \infty$. The suggestion, which is supported by the running time analyses of the methods, is that asymptotically the time it takes to compute a discrete logarithm modulo a general prime $N$ is about the time required for a special prime of size $N^2$.

The primes under consideration in the present paper are not necessarily of the form usually handled by the SNFS. However, their low weight does mean that an extension of the SNFS can be used, which in some cases will be faster than the general NFS. In the next section, we describe this algorithm. Because our results are applicable to the problem of factoring as well as to the discrete logarithm problem, and in order to avoid discussion about the existence of primes of fixed weight, we formulate our algorithm as a general method which takes an arbitrary odd integer $N$ as input and which can be incorporated either into a method to factor $N$ when $N$ is composite or a method to compute discrete logarithms mod $N$ when $N$ is prime.

In §3, we analyze the algorithm of the previous section and conjecture that if the weight of $N$ is a fixed value $w$, then the running time of the algorithm is bounded above by $L_N[1/3; c + o(1)]$, where

$$c < \left(\frac{32}{9}\right)^{1/3} \left(\frac{2w-3}{w-1}\right)^{1/3}$$

and the $o(1)$ is for $N \to \infty$. Thus, for fixed $w$, our algorithm is asymptotically faster than the general NFS. We also exhibit an infinite set of integers $N$ for which the algorithm has a conjectural running time of

$$L_N[1/3; (32\tau^2/9)^{1/3} + o(1)],$$

where

$$\tau = \frac{\sqrt{2}w - 2\sqrt{2} + 1}{w-1}.$$

The reader can consult Figure 3.17 to see that the values of $\tau^2$ and $(2w-3)/(w-1)$ are very close. Though the algorithm we present is designed for low-weight inputs, it turns out that the weight of the input is only a weak indicator of the running time of the method. Thus the worst-case running times that we conjecture are of limited value. To partially remedy the situation, we also consider in §3 what happens on average when we restrict to inputs of a fixed weight.

In the fourth and final section of the paper, we turn to practical considerations. To get a better sense of the speed-up afforded by small weight inputs, we compare the size of the numbers that are tested for smoothnes in our algorithm with those arising in other versions of the NFS. The discrete logarithm problems we use for this investigation are the seven examples appearing in [7]. Of these, four involve a finite field of degree two over its prime field. In all but one of the seven examples, the method we describe in §2 is superior to other versions of the NFS, and in many of these cases, dramatically so.

## 2. The algorithm.

Whether it is being used to factor an integer or to compute discrete logarithms in a prime field, the number field sieve (NFS) is comprised of two major components. One is an extensive search, by means of a sieve, for a large number of smooth elements. The other is a massive linear algebra computation. These steps are preceded by the construction of a polynomial $f$ with integer coefficients, which determines among other things what numbers are to be candidates for smoothness in the sieving step. Our purpose in this section is to describe and analyze a method for choosing $f$ which is particularly well-suited to the case that the integer to be factored or the characteristic of the field in which discrete logarithms are to be computed, is of low weight. For the sake of completeness, and in order to understand the impact of the choice of polynomial, we provide a brief summary of the NFS in its entirety. We begin with the piece of the algorithm that is common to its application to both the factoring and discrete logarithm problems.

**Algorithm 2.1.** This algorithm takes as input

(i) an odd integer $N$ of weight $w$, represented as the sum

$$\sum_{i=1}^{w} \epsilon_i 2^{c_i},$$

with $c_w > c_{w-1} > \ldots > c_1 = 0$ and $\epsilon_1, \ldots, \epsilon_w \in \{1, -1\}$, and

(ii) parameters $e, k, B, M \geq 2$, with $e$ integral.

Its purpose is to produce a matrix $A$ which can be used, as described subsequently, for factoring $N$ or computing discrete logarithms mod $N$.

**Step 1.** For $i = 1, \ldots, w$, let $\bar{c}_i$ be the least non-negative residue of $c_i$ mod $e$. In addition, let $\bar{c}_{w+1} = e$. Assume that $\sigma$ is a permutation on the set $\{1, \ldots, w, w+1\}$ which fixes $w+1$ and has the property that the sequence

$$\bar{c}_{\sigma(1)}, \bar{c}_{\sigma(2)}, \ldots, \bar{c}_{\sigma(w)}, \bar{c}_{\sigma(w+1)} \tag{2.2}$$

is non-decreasing, and let $J$ be the largest number such that $\bar{c}_{\sigma(J+1)} - \bar{c}_{\sigma(J)}$ is greater than or equal to the difference between any other pair of consecutive terms in (2.2). Finally, write $\mu$ for the quantity $e - \bar{c}_{\sigma(J+1)}$, and let

$$f = \sum_{i=1}^{w} \epsilon_i 2^{a_i} x^{b_i},$$

where

$$a_i = \bar{c}_i + \mu \quad \text{and} \quad b_i = \lfloor c_i/e \rfloor, \qquad \text{if} \quad \sigma^{-1}(i) \leq J,$$

and

$$a_i = \bar{c}_i - \bar{c}_{\sigma(J+1)} \quad \text{and} \quad b_i = \lfloor c_i/e \rfloor + 1, \qquad \text{if} \quad \sigma^{-1}(i) \geq J+1.$$

It is a straightforward matter to verify that

$$\max |a_i| = \bar{c}_{\sigma(J)} + \mu = e - (\bar{c}_{\sigma(J+1)} - \bar{c}_{\sigma(J)})$$

and that

$$f(2^e) = 2^\mu N.$$

In the case that $N$ is prime, it follows from a result in [5], that $f$ is irreducible. In the case that $N$ is composite and $f$ factors into a product $g_1 \ldots g_m$ of non-constant irreducible polynomials, we either obtaining a splitting of $N$ by plugging $2^e$ into the $g_i$ or we find that $g_i(2^e)$ is a multiple of $N$ for some $i$. In the former case, our problem is solved and in the latter, replacing $f$ by $g_i$ only improves the algorithm that follows. For this reason, we assume from this point onwards that $f$ is irreducible.

Let $\alpha \in \mathbb{C}$ be a root of $f$, let $\eta = 2^{a_w}\alpha$, and let $\mathcal{O}$ be the ring of integers of $\mathbb{Q}(\alpha)$. Note that $\eta$ is a root of the monic polynomial $g = \sum_{i=1}^{w} 2^{a_i + a_w(b_w - b_i - 1)} x^{b_i} \in \mathbb{Z}[x]$ and hence $\eta \in \mathcal{O}$. Let $\Delta$ be the discriminant of $f$, and let $T$ be the set of first degree prime ideals in $\mathcal{O}$ which lie over a rational prime which is prime to $\Delta$ and at most $B$. Finally, let $d$ be the degree of $f$.

**Step 2.** Recall that an integer is said to be $B$-smooth if each of its prime factors is at most $B$. The goal of this step is to find all pairs $(a, b)$ of relatively prime integers satisfying $|a| \leq M$ and $0 < b \leq M$ such that

$$(a - b2^e) f(a/b) b^d \tag{2.3}$$

is $B$-smooth and prime to $\Delta$. This can be accomplished by means of a sieve (see [2]). Call the set of such pairs $U$. If $|U| < \pi(B) + |T| + k$, then the algorithm is not successful and terminates.

**Step 3.** For each $(a, b) \in U$ and each prime ideal $\mathfrak{q}$ in $T$, compute $v_\mathfrak{q}(2^{a_w}a - b\eta)$ where $v_\mathfrak{q}$ is the valuation corresponding to $\mathfrak{q}$. Note that in the case that $\mathfrak{q}$ lies above an odd rational prime $q$, the valuation of $2^{a_w}a - b\eta$ is equal to the order of $q$ at $b^d f(a/b)$ if $\eta$ is congruent to $2^{a_w}a/b$ mod $\mathfrak{q}$ and is 0 otherwise. See §12 of [2] for a discussion of how to handle the case that $\mathfrak{q}|(2)$. Compute also the valuations $v_q(2^{a_w}(a - b2^e))$ where $q$ runs through the rational primes at most $B$. Let $v(a,b)$ be the vector with components $v_{\mathfrak{q}}(2^{a_w}a - b\eta)$ for $\mathfrak{q} \in T$ and $v_q(2^{a_w}(a - b2^e))$ for $q \leq B$, and let $A$ be the matrix whose rows are the vectors $v(a,b)$. This completes the description of Algorithm 2.1.

The matrix $A$ produced by Algorithm 2.1 can be incorporated into a factoring algorithm or a discrete logarithm algorithm. We briefly describe both of these applications. The many details that are omitted can be found in the references given. In what follows, we let $\phi : \mathbb{Z}[\eta] \to \mathbb{Z}/N\mathbb{Z}$ be the ring homomorphism which sends $\eta$ to $2^{a_w}2^e$.

**Factoring with** $A$. (See [2], [16].) To factor the integer $N$, we reduce the matrix $A$ mod 2 and augment it with quadratic character columns. Each of these columns is associated to a degree one prime ideal $\mathfrak{q}$ of $\mathcal{O}$ whose norm is greater than $B$ and is formed by placing a 0 in the row corresponding to a pair $(a,b)$ if $2^{a_w}a - b\eta$ is a square mod $\mathfrak{q}$ and placing a 1 otherwise. It is shown in [2] that if the quadratic characters being used are suitably independent, which we expect them to be, then the number of columns needed is bounded by $c \log N$, for some constant $c$. We also add to $A$ a column containing a 0 in each row for which the corresponding $(a,b)$ satisfies $a - b2^e > 0$ and containing a 1 in the remaining rows. Proceeding under the assumption that $k$ is chosen to be at least $c \log N + 2$, we see that there should exist a linear dependency mod 2 among the rows of our modified matrix. We find such a dependency using, for instance, the algorithm of [16]. Those rows appearing in the relation correspond to a subset $U'$ of $U$ having the property that the products

$$\prod_{(a,b)\in U'} 2^{a_w}a - b\eta \tag{2.4}$$

and

$$\prod_{(a,b)\in U'} 2^{a_w}(a - b2^e) \tag{2.5}$$

are squares in $\mathcal{O}$ and $\mathbb{Z}$ respectively. Let $g'$ denote the derivative of $g$. We next multiply (2.4) by $g'(\eta)^2$ in order to ensure that the resulting square, call it $\mu$, has a squareroot in $\mathbb{Z}[\eta]$. We also multiply (2.5) by $g'(2^{a_w}2^e)^2$ and call the resulting square $\nu$. By construction $\phi(\mu) = \phi(\nu)$. We now compute a squareroot $\sigma$ of $\mu$, a squareroot $t$ of $\nu$, and finally $\gcd(s - t, N)$, where $s \in \mathbb{Z}$ satisfies $\phi(s) = \phi(\sigma)$. Since $N|(s^2 - t^2)$, we expect with probability at least 1/2 that this gcd will be a non-trivial divisor of $N$.

**Computing discrete logarithms with** $A$. (See [13], [14].) Assume that $N$ is prime. Let $\tau$ be an element in $\mathbb{Z}[\eta]$ which factors over $T$ and whose image under $\phi$, call it $t$, is primitive. To compute base-$t$ discrete logarithms in $\mathbb{Z}/N\mathbb{Z}$ with $A$, we first negate the values in the columns corresponding to the rational primes $q$. Next we add a row to $A$, containing the values $v_{\mathfrak{q}}(\tau)$ for $\mathfrak{q} \in T$ and 0's in the components corresponding to the rational primes $q$. Finally, we reduce the matrix and augment it with new columns, as we did in the case of factoring. This time, however, the reduction is modulo a prime divisor $l$ of $N-1$, and the augmentation is with columns containing the values in $\mathbb{Z}/l\mathbb{Z}$ of character maps defined on the set of elements of $\mathcal{O}$ prime to $l$ (see [11], [14]). The number of characters required is equal to the unit rank $r$ of $\mathcal{O}$. We would like the modified matrix that we obtain, call it $A'$, to be of full rank. Thus, $k$ should be chosen sufficiently large to make it likely that this condition is met. Assuming that it is satisfied, we solve the matrix equation $A'x \equiv v \bmod l$, where $v$ is the vector with a 1 in the component corresponding to the row of $A'$ containing the valuations of $\tau$, and with 0's elsewhere. The linear algebra can again be performed by the method in [16].

We claim that the computation is likely to produce, for all rational primes $q \leq B$, the residue mod $l$ of $\log_t(\phi(q))$. Indeed, as explained in [14], under certain assumptions which are likely to hold, there exist integers $l_{\mathfrak{q}}$, for $\mathfrak{q} \in T$, and integers $x_1, \ldots, x_r$ such that for all pairs $(a, b)$,

$$\sum_{\mathfrak{q}\in T} v_{\mathfrak{q}}(2^{a_w}a - b\eta)l_t(\mathfrak{q}) + \sum_{i=1}^{r} \lambda_i(2^{a_w}a - b\eta)x_i \equiv \sum_{q\leq B} v_q(2^{a_w}(a - b2^e))\log_t(\phi(q)) \bmod l,$$

where the $\lambda_i$ are the character maps mentioned earlier. Additionally, the same values satisfy the congruence

$$\sum_{\mathfrak{q}\in T} v_{\mathfrak{q}}(\tau)l_t(\mathfrak{q}) + \sum_{i=1}^{r} \lambda_i(\tau)x_i \equiv 1 \bmod l.$$

Thus, the system $A'x \equiv v \bmod l$ has a solution. Since $A'$ is of full rank, our computation finds this, and only this, solution, and we obtain the residue mod $l$ of $\log_t(\phi(q))$ for all $q \leq B$. We now can determine the residue mod $l$ of $\phi(u)$ for any $B$-smooth integer $u$ since such an integer is, by definition, a product of primes at most $B$.

The restriction to a prime divisor $l$ of $N-1$ is necessitated by the character maps $\lambda_i$ ([11]). However, the method just described can be extended to the case of a prime power divisor of $N-1$. With the aid of the Chinese remainder theorem, we can then compute the logarithm of $\phi(u)$ for any $B$-smooth integer $u$. In fact, it is possible to use the values $l_{\mathfrak{q}}$ for $\mathfrak{q} \in T$ to compute the logarithm of $\phi(u)$ for any $u$ which factors over $T$ ([14]). For a discussion of how to reduce the general discrete logarithm problem to the cases just described, see [4] and [12].

## 3. Running time.

In this section we provide a worst-case running time analysis of Algorithm 2.1 for inputs of a fixed weight. Because the algorithm runs in widely varying times for different inputs of the same weight, we also consider the performance of the method on average for inputs of a fixed weight.

Our starting point is the special number field sieve (SNFS) described, for instance, in [10]. This is an algorithm which was originally designed to factor integers $N$ of the form $br^c + s$, with $b, r$ and $s$ small, and which has been adapted to the computation of discrete logarithms mod $N$ for primes $N$ of the same form. Its conjectural running time is

$$L_N[1/3; (32/9)^{1/3} + o(1)] \tag{3.1}$$

for bounded $b, r$ and $s$ and $c \to \infty$. This is a significant improvement over the running time of the general number field sieve which runs conjecturally in time

$$L_N[1/3; (64/9)^{1/3} + o(1)] \tag{3.2}$$

for $N \to \infty$.

In the case that $w = 2$, the method for finding $f$ in Algorithm 2.1 is essentially the same technique as used in the SNFS. Indeed, Algorithm 2.1 can be viewed as a generalization of the SNFS to larger values of $w$. As $w$ grows, we expect the algorithm's running time to span the distance between the values in (3.1) and (3.2). Our first task is to determine precisely how this occurs. The conjecture that follows is closely modeled after those appearing in §11 of [2].

**Conjecture 3.3.** *Fix $w$, let $\theta = (2w-3)/(w-1)$, and assume $k = O(\log N)$. It is possible to pick $e, B$, and $M$ satisfying*

$$\begin{aligned} e &= (2\theta/3 + o(1))(\log N)^{2/3}(\log\log N^{1/3}), \\ B = M &= L_N[1/3; (4\theta/9)^{1/3} + o(1)] \end{aligned} \tag{3.4}$$

*for $N \to \infty$, such that on input of an integer $N$ of weight $w$, and parameters $e, k, B$, and $M$, Algorithm 2.1 succeeds in producing a matrix $A$ in time at most*

$$L_N[1/3; (32\theta/9)^{1/3} + o(1)] \tag{3.5}$$

*for $N \to \infty$.*

Let $N$ be represented as in Algorithm 2.1. That is, $N = \sum_1^w \epsilon_i 2^{c_i}$, with $c_w > c_{w-1} > c_1 = 0$ and $\epsilon_i \in \{1, -1\}$. The running time of Conjecture 3.3 is obtained by choosing $e$ so that the least non-negative residue of $c_w$ modulo $e$ is close to 0 or $e$. The obvious way to accomplish this is to select $d$ first and then choose $e$ close to $c_w/d$. For our purposes, we choose $d$ satisfying

$$d = \Big(\frac{(3\theta + o(1))\log N}{2\log\log N}\Big)^{1/3}, \tag{3.6}$$

and let $e = \lfloor c_w/d \rfloor$. Then $e$ satisfies the condition in (3.4). Moreover, for $N$ sufficiently large, we have $d(d+1) \le c_w$, from which it follows that the degree of the polynomial $f$ formed in Step 1 of Algorithm 2.1 is $d$. Since the least positive residue of $c_w$ mod $e$ is less than $d$ and since for $N$ sufficiently large, $d - 1 < e/w$, we conclude that for $N$ sufficiently large, the largest gap $\gamma$ in sequence (2.2) is at least

$$e\Big(\frac{1}{w-1} - \nu\Big),$$

where

$$\nu = \frac{d-1}{e(w-1)}.$$

Recall that the coefficients of $f$ are bounded in absolute value by $2^{e-\gamma}$. Thus, for $N$ sufficiently large, the numbers which are being tested for smoothness in Step 2 of Algorithm 2.1, that is the numbers given in (2.3), are bounded by the quantity

$$2dM^{d+1}(2^e)^{2-\frac{1}{w-1}+\nu}.$$

Letting $\theta = (2w-3)/(w-1)$ and using the fact that $e = \lfloor c_w/d \rfloor$, we see that this bound is at most $2dM^{d+1}(2^{c_w})^{(\theta+\nu)/d}$ which is less than or equal to

$$2dM^{d+1}(2N)^{\frac{\theta+\nu}{d}}. \tag{3.7}$$

The remainder of our argument in support of Conjecture 3.3 is the same as the one given following Conjecture 11.4 in [2]. For the sake of completeness but with no claim of originality, we include it here.

For any positive real numbers $x, y$, let $\psi(x,y)$ denote the number of integers at most $x$ which are $y$-smooth. It is proven in [3] that for any $\epsilon > 0$,

$$\frac{\psi(x, x^{1/w})}{x} = w^{-w(1+o(1))} \tag{3.8}$$

for $w \to \infty$, uniformly for $x \geq w^{w(1+\epsilon)}$. It follows that in the case that

$$M_0 = B_0 = L_N[1/3; (4\theta/9)^{1/3}]$$

and $z_0$ equals the quantity in (3.7) with $M$ equal to $M_0$, we have

$$\frac{M_0^2 \psi(z_0, B_0)}{z_0} = B_0^{1+o(1)} \tag{3.9}$$

for $N \to \infty$. Note that we have relied here on the fact that $d$ satisfies (3.6) and that $\nu = o(1)$ for $N \to \infty$. Since the number of pairs $(a, b)$ considered in Step 2 of Algorithm 2.1 is approximately $12M_0^2/\pi^2$ when $M = M_0$, we see that if the numbers tested for smoothness in Algorithm 2.1 behave with respect to the property of being $B_0$-smooth like random numbers at most $z_0$, then Algorithm 2.1, upon input of parameters $e, k, B_0$ and $M_0$, produces a set $U$ of size at least $B_0^{1+o(1)}$. Since the threshold $\pi(B) + |T| + k$ given in Step 2 of Algorithm 2.1 is also equal to $B_0^{1+o(1)}$ in this case, we might expect that $|U|$ is sufficiently large, and that the algorithm succeeds, with parameters close to those given. In fact, they need only be adjusted slightly.

Let $\epsilon$ be any positive real number, let

$$M = B = L_N[1/3; (1+\epsilon)(4\theta/9)^{1/3}],$$

let $d$ and $e$ remain unchanged, and let $z$ equal (3.7) for these values. Observe that

$$\frac{\log z}{\log B} \leq \frac{\log z_0}{\log B_0}$$

and that $(\log z)/\log B \to \infty$ as $N \to \infty$. It follows, then, from (3.8) and (3.9) that

$$\frac{M^2 \psi(z, B)}{z} \geq M_0^{2\epsilon} M_0^2 \Big(\frac{\psi(z_0, B_0)}{z_0}\Big)^{1+o(1)} = M_0^{2\epsilon} B_0^{1+o(1)} = B^{1+o(1)(1+2\epsilon)/(1+\epsilon)}.$$

We now see that no matter how the $o(1)$ tends to 0 in the equation $\pi(B)+|T|+k = B^{1+o(1)}$, there exists a constant $C_\epsilon$ such that

$$\frac{(12/\pi^2) M^2 \psi(z, B)}{z} \geq \pi(B) + |T| + k \tag{3.10}$$

for $N > C_\epsilon$.

We consider what happens as $\epsilon$ goes to 0. That is, for each $N$ sufficiently large, we choose a positive real number $\epsilon(N)$ so that $N > C_{\epsilon(N)}$ and so that $\epsilon(N) \to 0$ as $N \to \infty$. Then we see that for $N$ sufficiently large, Algorithm 2.1 should produce a large enough set $U$ upon input of $e$ as above and $M = B = L_N[1/3; (1+\epsilon(N))(4\theta/9)^{1/3}]$. Since $\epsilon(N) \to 0$ as $N \to \infty$, our choices for $M$ and $B$ satisfy the condition in (3.4). Moreover, it is easily verifed that for $M$ and $B$ as such, the time required for Steps 2 and 3 of the algorithm is equal to $M^{2+o(1)}$. This quantity is equal to (3.5) and our argument in support of Conjecture 3.3 is complete.

The running time given in Conjecture 3.3 is best possible in the following sense. Let $z$ again denote the quantity in (3.7). Then it is possible to show rigorously that if (3.10) holds and if $k$ is chosen sufficiently small so that $\pi(B) + |T| + k = B^{1+o(1)}$, then $M$ is at least $L_N[1/3; (4\theta/9)^{1/3} + o(1)]$. See §10 of [2] for details.

Conjecture 3.3, however, does not give the right answer. It turns out that it is not possible to find an infinite set of integers for which the bound in (3.7) is attained. Indeed, one might expect that for fixed $w$, Algorithm 2.1 would run in time (3.5) for $N$ of the form

$$2^c + \sum_{i=0}^{w-2} 2^{\left(\frac{i}{w-1}\right)\gamma} = 2^c + \frac{1-2^\gamma}{1-2^{\gamma/(w-1)}},$$

where $\gamma$ is the floor of

$$\frac{c}{\left\lfloor \left(\frac{3\theta \log 2^c}{2 \log\log 2^c}\right)^{1/3} \right\rfloor}.$$

Certainly, for these numbers, if $d$ is chosen to be the denominator of the above expression and $e$ is set equal to $\gamma$, then the bound (3.7) is the right one to use and the algorithm runs in time (3.5). However, choosing $d$ to be slightly smaller, and again letting $e = \lfloor c/d \rfloor$, improves the asymptotic running time by forcing the largest gap in sequence (2.2) to be a little bigger than $e/(w-1)$.

More generally, we have the following strategy. Assume we are given as input an integer $N$ of weight $w$, represented as usual as $\sum_{i=1}^{w} \epsilon_i 2^{c_i}$, with $\epsilon_i \in \{1, -1\}$ and $c_w$ maximal. We choose $d$ divisible by $w$, and proceed by producing two polynomials according to the method described in Step 1 of Algorithm 2.1, one with $e$ equal to $e_1 = \lfloor c_w/d \rfloor$ and the other with $e$ set equal to $e_2 = e_1 w/(w-1)$. We then continue the algorithm with the polynomial which produces the smaller smoothness candidates. The rationale for this approach is the following observation. If in the case that $e = e_1$, the largest gap in (2.2) is close to $e_1/(w-1)$ then the least non-negative residue mod $e_1$ of each $c_i$ is necessarily close to a multiple of $e_1/(w-1)$. It follows that in the case that $e = e_2$, the largest gap is close to $2e_1/(w-1)$. Note that because $w|d$, we are assured that in this case the least non-negative residue of $c_w$ is again less than $d$. We find then that when $e_1$ is not a good choice, the gap in (2.2) corresponding to $e_2$ is significantly better than the worst case. When we optimize this method, we obtain for all $w$, a running time which is slightly better than that of Conjecture 3.3. We leave the details to the interested reader and offer instead the following result which shows that the room for improvement over Conjecture 3.3 is quite small.

**Conjecture 3.11.** *Fix $w$, and let*

$$\tau = \frac{\sqrt{2}w - 2\sqrt{2} + 1}{w-1}.$$

*Then there exists an infinite set $S$ of integers of weight $w$ with the property that the time required by Algorithm 2.1 to produce a matrix $A$ is at least*

$$L_N\Big[\frac{1}{3}; \Big(\frac{32\tau^2}{9}\Big)^{1/3} + o(1)\Big] \tag{3.12}$$

*for $N$ in $S$ tending to $\infty$.*

We argue in support of Conjecture 3.11 by exhibiting a set $S$ with the stated property. Fix $w$ and a positive real number $\rho$, and for any integer $c \geq 2$, let

$$h = h(c) = \rho\Big(\frac{\log 2^c}{\log\log 2^c}\Big)^{1/3}.$$

In the case that $c \geq (w-1)(h+1)$, let

$$N_{c,\rho} = 2^c + \sum_{i=0}^{w-2} 2^{i \cdot \lfloor \frac{c}{(w-1)(h+1)} \rfloor}. \tag{3.13}$$

Finally, let

$$\zeta = \frac{\sqrt{2}+1}{2(\frac{2}{3}\tau)^{1/3}}.$$

Our set $S$ then is the set of numbers $N_{c,\zeta}$. To determine what happens when they are input into Algorithm 2.1, we consider various possibilities for the degree $d$ of the polynomial $f$

produced in Step 1. Critical to our approach is the observation that, as a consequence of the way in which $f$ is formed, the parameter $e$ and the degree $d$ satisfy

$$\frac{c}{d+1} \leq e \leq \frac{c}{d-1}.$$

Until otherwise indicated, we assume that $N$ is of the form given in (3.13) and for $i = 1, \ldots w-1$, we use the notation $c_i$ in place of $(i-1)\lfloor c/((w-1)(h+1))\rfloor$.

*Case 1*: Assume that $d$ is chosen so that $d \leq h$. Then $e \geq \frac{c}{d+1} \geq \frac{c}{h+1}$. It follows that

$$\begin{aligned} e - c_{w-1} &\geq \frac{c}{h+1} - (w-2)\left\lfloor \frac{c}{(w-1)(h+1)} \right\rfloor \\ &\geq (w-1)\left\lfloor \frac{c}{(w-1)(h+1)} \right\rfloor - (w-2)\left\lfloor \frac{c}{(w-1)(h+1)} \right\rfloor \\ &\geq \left\lfloor \frac{c}{(w-1)(h+1)} \right\rfloor. \end{aligned}$$

Because of the special form of $N$, the maximal gap in sequence (2.2) is equal either to $\lfloor c/((w-1)(h+1))\rfloor$ or to $e - c_{w-1}$, depending on where the least non-negative residue of $c$ mod $e$ appears in the sequence. In both cases, this maximal gap is at most $e - c_{w-1}$, and we conclude that the maximum absolute value of the smoothness candidates in Step 2 is at least

$$M^{d+1}(2^e - 1)2^{e-(e-c_{w-1})} = M^{d+1}(2^e - 1)2^{(w-2)\left\lfloor \frac{c}{(w-1)(h+1)} \right\rfloor}.$$

This quantity is, in turn, bounded below by

$$l_1 M^{d+1}(2^c)^{\frac{1}{d+1} + \frac{w-2}{(w-1)(h+1)}},$$

for some constant $l_1$ which does not depend on $c$.

*Case 2:* Assume that $d$ is chosen so that

$$h < d \leq (w-1)h.$$

Then $e \leq \frac{c}{d-1} < \frac{c}{h-1}$. If $e$ is in addition less than $c_{w-1}$ then the largest gap in sequence (2.2) is at most $\lfloor c/((w-1)(h+1))\rfloor$. If $e$ is greater than $c_{w-1}$ then it may be that the largest gap in (2.2) is equal to $e - c_{w-1}$ and that this difference is greater than $\lfloor c/((w-1)(h+1))\rfloor$. However,

$$e - c_{w-1} \leq \frac{c}{h-1} - (w-2)\left\lfloor \frac{c}{(w-1)(h+1)} \right\rfloor \leq \frac{c}{(w-1)(h+1)} + \frac{2c}{h^2-1} + w - 2.$$

We conclude that the maximum absolute value of the smoothness candidates in Algorithm 2.1 is at least

$$M^{d+1}(2^e-1)2^{e-\frac{c}{(w-1)(h+1)}-\frac{2c}{h^2-1}-(w-2)},$$

which, in turn, is at least

$$l_2 M^{d+1}(2^c)^{\frac{2}{d+1}-\frac{1}{(w-1)(h+1)}-\frac{2}{h^2-1}}, \tag{3.14}$$

for some constant $l_2$. Note that the upper bound on $d$ ensures that the exponent on $2^c$ in (3.14) is at least

$$\frac{1}{d+1}-\frac{2}{h^2-1}=\frac{1+o(1)}{d+1},$$

where the $o(1)$ is for $c\to\infty$.

*Case 3:* In the case that $d+1>(w-1)h$, we draw no conclusions about the coefficients of the polynomial $f$ and simply observe that the maximum absolute value of the numbers tested for smoothness in Algorithm 2.1 is at least $M^{d+1}(2^e-1)$. In this case, there exists a constant $l_3$ such that this quantity is at least

$$l_3 M^{d+1}(2^c)^{\frac{1}{d+1}}.$$

We now rely on the proof of Lemma 10.12 in [2]. Let $M, B$, and $z$ be functions of $c$ and $d$ with the property that $M^2\psi(z,B)/z\geq g(B)$, where $g(B)\geq 1$ and $g(B)=B^{1+o(1)}$ for $B\to\infty$. Assume that $z\to\infty$ as $c\to\infty$. Then an optimization theorem which appears earlier in §10 of [2] implies that

$$M^2\geq L_z[1/2;\sqrt{2}+o(1)]$$

for $c\to\infty$. Let $D=\{(c,d)\,|\,d+1<c/2\}$, and let $\beta$ be a real-valued function on $D$. In addition, assume that

(i) $c\beta\geq(\log 2)^{-1}$ for $c$ sufficiently large and

(ii) $d+\log(2^{c\beta})\to\infty$ as $c\to\infty$.

In the case that

$$z=lM^{d+1}2^{c\beta},$$

where $l$ is a constant, Lemma 10.12 then supplies the necessary manipulations to conclude that for $(c,d)\in D$,

$$2\log M\geq(1+o(1))\Big(d\log d+\sqrt{(d\log d)^2+2\log(2^{c\beta})\log\log(2^{c\beta})}\Big), \tag{3.15}$$

where the $o(1)$ here, and for the remainder of this discussion, is for $c \to \infty$ uniformly in $d$.

For our purposes, we let $l$ be the minimum of $l_1, l_2$, and $l_3$, and let

$$\beta = \begin{cases} \frac{1}{d+1} + \frac{w-2}{(w-1)(h+1)} & \text{if } d \le h \\ \frac{2}{d+1} - \frac{1}{(w-1)(h+1)} - \frac{2}{h^2-1} & \text{if } h < d \le (w-1)h \\ \frac{1}{d+1} & \text{if } (w-1)h < d. \end{cases}$$

Since $\beta \ge (1+o(1))/(d+1)$, conditions (i) and (ii) above are met.

From our investigation of Cases 1-3, we see that when Algorithm 2.1 is run with input $N = N_{c,\rho}$ subject to the constraint that the degree of $f$ is some value $d$, the largest smoothness candidate that arises is at least $z = lM^{d+1}2^{c\beta}$ in absolute value. Since we are interested in a lower bound on the running time, we can assume that all the candidates are, in fact, bounded by $z$ in absolute value. If we then assume that these numbers behave with respect to $B$-smoothness as random numbers bounded by $z$, we find that any value of $M$ which is sufficiently large that Algorithm 2.1 succeeds must satisfy

$$\frac{(12/\pi^2)M^2\psi(z,B)}{z} \ge \pi(B) + T + k,$$

where $T$ and $k$ are as in the description of the algorithm. Since $\pi(B) + T + k \ge B^{1+o(1)}$, we obtain (3.15) for the $\beta$ given, so long as we restrict the domain of $\beta$ to $D$. However, this restriction is of no consequence. The parameters adopted in Conjecture 3.3 lead to smoothness candidates which are at most $L_N[2/3; \xi + o(1)]$ for some constant $\xi$. Since these candidates are at least $2^{d+1}$ in size, regardless of what polynomial $f$ of degree $d$ is used, we see that the optimal choice of $d$ must satisfy $2^{d+1} \le L_N[2/3; \xi + o(1)]$. We conclude that for $c$ sufficiently large,

$$2^{d+1} < N^{\frac{1}{3}} < (2^{c+1})^{1/3} \le 2^{\frac{c}{2}}.$$

Thus for $c$ sufficiently large, the only pairs $(c,d)$ of interest are in $D$.

It remains to minimize (3.15), a task which is accomplished by having $(d\log d)^2$ be of the same order as the term accompanying it under the square-root sign. In each of the three cases appearing in the definition of $\beta$, we find as a consequence that $d$ must satisfy

$$d = (\delta + o(1))(\log 2^c/\log\log 2^c)^{1/3}$$

for some constant $\delta$. Substituting in for $d$, we determine that

$$2\log M \ge (m + o(1))(\log 2^c)^{1/3}(\log\log 2^c)^{2/3}, \tag{3.16}$$

where $m$ is the minimum value of

$$f_\rho(\delta) = \begin{cases} \frac{\delta}{3} + \left(\frac{\delta^2}{9} + \frac{4}{3\delta} + \frac{4(w-2)}{3(w-1)\rho}\right)^{\frac{1}{2}}, & \text{if } \delta \le \rho \\ \frac{\delta}{3} + \left(\frac{\delta^2}{9} + \frac{8}{3\delta} - \frac{4}{3(w-1)\rho}\right)^{\frac{1}{2}}, & \text{if } \rho \le \delta \le (w-1)\rho \\ \frac{\delta}{3} + \left(\frac{\delta^2}{9} + \frac{4}{3\delta}\right)^{\frac{1}{2}}, & \text{if } \delta \ge (w-1)\rho. \end{cases}$$

We leave it to the reader to verify that in the case that $\rho = \zeta$, this minimum occurs at $\delta = (3\tau^2/2)^{1/3}$ or $(3\tau^2/2)^{1/3}/\sqrt{2}$ and is equal to $(32\tau^2/9)^{1/3}$. Since $2^c < N_{c,\rho} < 2^{c+1}$, we can replace the quantity $2^c$ in (3.16) with $N_{c,\rho}$. Thus we cannot do better than (3.12) on the set of numbers $N_{c,\zeta}$. This completes our argument in support of Conjecture 3.11.

The secondary constant in (3.12) is, of course, bounded by the secondary constant in the running time proposed in Conjecture 3.3. Figure 3.17 below gives the values of these constants for small values of $w$. All entries are rounded off to the nearest thousandth. We note that in the case that $w = 2$, our upper and lower bounds equal $(32/9)^{1/3}$ and that both constants approach $(64/9)^{1/3}$ as $w \to \infty$.

| | $\lambda$ | $(32\tau^2/9)^{1/3}$ | $(32\theta/9)^{1/3}$ |
|---|---|---|---|
| $w = 2$ | 1.526 | 1.526 | 1.526 |
| $w = 3$ | 1.729 | 1.730 | 1.747 |
| $w = 4$ | 1.781 | 1.796 | 1.810 |
| $w = 5$ | 1.805 | 1.828 | 1.839 |
| $w = 6$ | 1.819 | 1.847 | 1.857 |
| $w = 7$ | 1.828 | 1.860 | 1.868 |
| $w = 8$ | 1.835 | 1.869 | 1.876 |
| $w = 9$ | 1.840 | 1.876 | 1.882 |
| $w = 10$ | 1.843 | 1.881 | 1.887 |

Figure 3.17

Though Conjectures 3.3 and 3.11 give important information about Algorithm 2.1 and are easily juxtaposed with analogous conjectures for the SNFS and the general NFS, they are deceptive. On the one hand the asymptotic improvement over the general NFS that they report suggests, overly optimistically, that for a given integer $N$, Algorithm 2.1 should be the method of choice. However, for a given weight $w$, this improvement is only realized if $N$ is sufficiently large. Indeed, in §4 we observe that Algorithm 2.1 gives an improvement over standard NFS methods only if $w$ is at most on the same order of $d$, that is on the order of $(\log N/\log\log N)^{1/3}$.

Conjectures 3.3 and 3.11 are also overly pessimistic. As is clear from our analysis, for a given input $N$, the running time of Algorithm 2.1 is not so much determined by the weight of $N$ but by the the largest gap in the sequence (2.2). As the next result shows, this gap is significantly greater on average than the value $e/(w-1)$ used to obtain Conjecture 3.3, especially when $w$ is small.

**Proposition 3.18.** *Let $w$ and $e$ be integers greater than or equal to 2. Let $S = S(e, w)$ be the set of sequences of length $w-2$ of non-negative integers less than $e$. For $s \in S$, let $g(s)$ be the largest difference between consecutive terms of the sequence obtained by ordering the elements of $s$, together with the numbers 0 and $e$, from smallest to largest. Then the average value of $g(s)$ as $s$ ranges over $S$ is*

$$\frac{1}{e^{w-2}} \sum_{k=1}^{w-2} \frac{k^{w-2-k}(e!)\, G(e, k+2)}{(e-k)!}, \tag{3.19}$$

*where*

$$G(e,k) = \frac{\sum\limits_{g=\lceil \frac{e}{(k-1)} \rceil}^{e} g \sum\limits_{j=1}^{\lfloor \frac{e}{g} \rfloor} (-1)^{j+1} \binom{k-1}{j} \sum\limits_{l=1}^{\lfloor \frac{e}{jg} \rfloor} (-1)^{l+1} \binom{k-2}{l-1} \binom{e-1-(j+l-1)g}{k-(j+2)}}{\binom{e}{k-2}}. \tag{3.20}$$

*Proof.* For a given $w$, let $S_0 = S_0(e, w)$ be the set of increasing sequences of length $w-2$ of distinct non-negative integers less than $e$. We first calculate the average value of $g(s)$ as $s$ varies over $S_0$. We accomplish this by counting, for a given value $g$, the number of $s$ such that $g(s) = g$. The first step is to determine for a given $g$, how many ways there are to partition $e-g$ into $w-2$ parts, subject to the condition that each summand is at least 1 and at most $g$, and then to multiply by $w-1$ to account for the fact that the maximum gap $g$ can occur at any one of the $w-1$ parts of $e$. Standard, elementary, combinatorial techniques yield the answer

$$(w-1) \sum_{j=1}^{\lfloor e/g \rfloor} (-1)^{j+1} \binom{w-2}{j-1} \binom{e-1-jg}{w-3}.$$

Of course, this tally is too large since those partitions of $e$ containing more than one $g$ have been counted more than once. We thus need to subtract off the number of ways to partition $e-2g$ into $w-3$ parts, multiplied by $\binom{w-1}{2}$, add back in the number of ways to partition $e-3g$ into $w-4$ parts, multiplied by $\binom{w-1}{3}$, and so on. Multiplying the resulting alternating sum by $g$, summing over all the possible values of $g$, and dividing by $|S_0|$ yields the value $G(e,w)$ given in (3.20).

Since $G(e,w)$ is also the average value of $g(s)$ as $s$ ranges over the set of sequences of length $w-2$ of distinct non-negative integers less than $e$, it only remains to handle those sequences in $S(e,w)$ with repeated terms. We classify these according to how many distinct terms each such sequence contains. In particular, we observe that for $k=1,\ldots,w-2$, the number of sequences in $S(e,w)$ containing $k$ distinct terms is $k^{w-2-k}|S_0(e,k)|$. The average value of $g(s)$ for these sequences is $G(e,k+2)$. We find, therefore, that the average value of $g(s)$ over $S(e,w)$ is

$$\frac{\sum_{k=1}^{w-2} k^{w-2-k}|S_0(e,k)|G(e,k+2)}{|S(e,w)|}.$$

This expression is the same as (3.19), and the proof of the proposition is complete.

Proposition 3.18 gives us an idea of what to expect when running Algorithm 2.1 on a number $N$ of weight $w$. Let $c_w$ be the largest exponent appearing in the weight $w$ representation of $N$. Then the quantity in (3.19) represents, roughly speaking, the size of the gap we can expect in sequence (2.2) when Algorithm 2.1 is run with $e$ chosen so that the least non-negative residue of $c_w$ mod $e$ is negligible. It is easy enough to compare this value with the worst-case value of $e/(w-1)$. We can make a comparison, however, that does not depend on $e$ by noting that (3.19) is asymptoticaly linear in $e$. That is, as $e\to\infty$ with $w$ fixed, the ratio of (3.19) to $e$ approaches a constant which we call $\mu(w)$ . In Figure 3.21 below, we give for small $w$, the value of (3.19) divided by $e$ when $e=200$ and when $e=1000$, the value of $\mu(w)$, and for comparison, the value of $1/(w-1)$. All entries are rounded to the nearest one thousandth. The data reveals clearly that one can expect in practice to fare much better than the running times given in Conjectures 3.3 and 3.11. Though we opt not to provide an analogue of Proposition 3.18 that addresses the possibility that $c_w$ has a large non-negative residue mod $e$, the implications of the proposition apply to Algorithm 2.1 in general. Indeed, in the case that $e$ is unconstrained, the values in the three middle columns of the following chart can only go down.

|  | $e = 200$ | $e = 1000$ | $\mu(w)$ | $1/(w-1)$ |
|---|---|---|---|---|
| $w = 3$ | .753 | .751 | .750 | .500 |
| $w = 4$ | .604 | .610 | .611 | .333 |
| $w = 5$ | .508 | .518 | .521 | .250 |
| $w = 6$ | .439 | .453 | .457 | .200 |
| $w = 7$ | .386 | .404 | .408 | .167 |
| $w = 8$ | .340 | .364 | .370 | .143 |
| $w = 9$ | .301 | .332 | .340 | .125 |
| $w = 10$ | .268 | .305 | .314 | .111 |

Figure 3.21

Going one step further, we can replace $\theta$ with $2 - \mu(w)$ in (3.7) and modify the statement of Conjecture 3.3 accordingly. In particular, quantity (3.5) with $\theta$ replaced by $2 - \mu(w)$ represents a kind of average running time for Algorithm 2.1 to succeed upon input of an integer of weight $w$ and subject to the restriction that $c_w$ is close to a multiple of $e$. We leave a precise formulation of this statement to the reader.

We conclude this section by noting that the algorithms sketched in §2 which use $A$ to factor $N$ or compute discrete logarithms mod $N$, have running times dominated by the linear algebra computation appearing in those methods. Because of the sparseness of $A$, the method in [16] runs in time $B^{2+o(1)}$. Since $M$ and $B$ are taken to be equal in our analysis of Algorithm 2.1 and since Algorithm 2.1 runs in time $M^{2+o(1)}$, we find that the results described in this section apply also to the factoring and discrete logarithm algorithms which incorporate Algorithm 2.1.

## 4. Examples.

In practice, when confronted with a particular $N$ which one wants to factor or modulo which, one wants to compute logarithms, asymptotic results are of little interest. One simply wants to choose the best method for the number at hand. In this section, we compare the performance of Algorithm 2.1 with that of other variations of the number field sieve (NFS).

The chief factor in determing how fast the sieving stage of these methods runs for a given input is the size of the numbers being tested for smoothness. The algorithm to choose is the one with the smallest smoothness candidates. In the standard version of the NFS for factoring, these numbers are bounded by

$$2(d+1)M^{d+1}N^{2/(d+1)}, \tag{4.1}$$

where $M$ as usual is the bound on the size of the coefficients of the elements considered for smoothness and $d$ is the degree of the polynomial $f$. In the case of discrete logarithms modulo a prime, the best approach, due to Joux and Lercier, is one in which two polynomials are produced having a shared root mod $N$ ([6]). One then works in the two extensions of $\mathbb{Q}$ obtained by adjoining the roots of these polynomials. The smoothness candidates in this method are bounded by

$$\Big(\frac{d^2}{4}+\frac{3d}{2}+2\Big)M^{d+1}N^{2/(d+2)}, \tag{4.2}$$

where $d$ is the sum of the degrees of the two extensions being used.

Comparsion of these bounds with the corresponding bound in Algorithm 2.1 already provides some information. Indeed, if for the sake of simplicity, we use the bound of $M^{d+1}N^{\theta/d}$, where $\theta = (2w-3)/(w-1)$, for Algorithm 2.1 and compare it with $M^{d+1}N^{2/(d+2)}$, we see immediately that Algorithm 2.1 should begin to outperform other NFS methods, whether for factoring or discrete logarithms, when $\theta/d \le 2/(d+2)$, or equivalently when

$$w \le \frac{d+6}{4}.$$

Since $d$ grows very slowly with respect to $N$, this inequality represents a severe restriction on $w$ and suggests that Algorithm 2.1 is only useful for numbers of extremely small weight. However, we have already observed that weight is not a great predictor of performance for this method. Indeed, each case must be consider individually. This is precisely what we do in this section with the examples from [7]. The first three are for computing logarithms in prime fields. The remaining four are for computing logarithms in fields of degree two and are accompanied by a brief discussion of this case in general.

For each example in the prime case, we compare the bounds given in (4.1) and (4.2) with the size of the largest smoothness candidates tested when Algorithm 2.1 is used. This quantity is equal to

$$2dM^{d+1}2^{2e-\gamma},$$

where for a given $e$, we let $\gamma$ be the largest gap in sequence (2.2) and for a given $d$, we choose $e$ between $e/(d+1)$ and $e/(d-1)$ so as to minimize the exponent $2e-\gamma$. For fixed $d$, the three bounds are easily ranked, since the $M^{d+1}$ term can be ignored. For different $d$, however, we encounter a problem, since the bounds now depend on the value of $M$. We overcome this difficulty by minimizing for a given $d$, the value of max$\{M, B\}$ subject to the constraint, already seen in §3, that

$$\frac{(12/\pi^2)M^2\psi(z,B)}{z} \geq \pi(B) + |T| + k, \tag{4.3}$$

where $T$ and $k$ are as defined Algorithm 2.1 and $z$ is a bound on the size of the smoothness candidates. We use the resulting minima for our rankings. Our choice of target function max$\{M, B\}$ reflects our desire to take into account the linear algebra step that dominates the NFS once the sieving is complete. Indeed, at least asymptotically, this is the right function to minimize since the sieve in the NFS runs in time $M^{2+o(1)}$ and the linear algebra runs in time $B^{2+o(1)}$. Our choice of target also reflects our desire for simplicity, since we can now take $M = B$. Recall that this equality holds when Algorithm 2.1 is optimized on its own, as well as when the sieving stage of any of the other versions of the NFS is optimized. We further simplify constraint (4.3) by replacing $\pi(B) + |T| + k$ with the estimate $(d+1)B$ and by using (3.8) to approximate $\psi(z,B)/z$. We are left then with the problem of computing, for a given $d$ and a given expression for the bound $z$, the smallest $M$ such that

$$Mu^{-u} \geq \frac{\pi^2(d+1)}{12},$$

where $u = (\log z)/\log M$. By comparing the values of $M$ produced, as we vary $d$ and the NFS version used, we obtain an optimal value of $M$ for $N$ itself.

We include this information in our examples below, albeit with some hesitation as these numbers are very rough and are generally higher than those found by means of extrapolation techniques such as those used in [8]. We also use this data, in those cases that Algorithm 2.1 is the best choice for $N$, to estimate the size of a general prime on which the NFS variation of Joux and Lercier runs in time approximately equal to that required by Algorithm 2.1 to compute logarithms mod $N$. More specifically, we determine the bit-length of the smallest general prime having the property that when we estimate how many smoothness candidates must be tested, using bound (4.2) and the same estimate for $\psi(z, B)$ as before, we obtain in the best case the optimal value of $M$ associated to $N$. This bit-length then measures in some sense the NFS-security of $N$.

Though we have checked, for each example that follows, all feasible values of $d$, we only include data for those $d$ for which we estimate the algorithms run fastest. In each case, the interval from $\lfloor(3\log N/2\log\log N)^{1/3}\rfloor$ to $\lceil(3\log N/\log\log N)^{1/3}\rceil$ is contained in the range of $d$ values presented. All values given in the charts are bit-lengths. Only powers of 2 were considered when determing the optimal values of $M$. In those cases that we give the NFS-security of a prime, we round to the nearest 10.

**Example 4.4.** We consider the weight 7 prime

$$2^{3202} - 2^{3121} + 2^{3038} + 2^{2947} - 2^{2865} + 2^{2690} + 1.$$

In this case, we find that the method of Joux and Lercier beats Algorithm 2.1. The "optimal $M$" values given below are for the former method. We note that there are degrees for which the polynomial obtained using Algorithm 2.1 is superior to the one given by the method of Joux and Lercier. Indeed, when the bit-length of $e$ is close to 90 or 180, the polynomial produced by Algorithm 2.1 has very small coefficients. The degree in these cases, however, is far too large to be practical.

| | $2d2^{2e-\gamma}$ | $2(d+1)N^{2/(d+1)}$ | $(\frac{d^2}{4} + \frac{3d}{2} + 2)N^{2/(d+2)}$ | optimal $M$ |
|---|---|---|---|---|
| $d = 5$ | 975 | 1071 | 919 | 78 |
| $d = 6$ | 819 | 919 | 805 | 77 |
| $d = 7$ | 751 | 805 | 717 | 77 |
| $d = 8$ | 661 | 716 | 646 | 77 |
| $d = 9$ | 588 | 645 | 588 | 78 |
| $d = 10$ | 547 | 587 | 540 | 79 |

**Example 4.5.** Our second example, also of weight 7, is the prime

$$2^{8376} - 2^{8333} + 2^{8288} - 2^{7991} + 2^{7947} + 2^{7604} + 1.$$

In this case, we find that Algorithm 2.1 is the winner. Moreover, the value of 104 bits in the "optimal $M$" column corresponds to the optimal value of $M$ obtained when using the method of Joux and Lercier on a prime of 7470 bits. Thus the NFS-security of this prime is only about 7470 bits.

| | $2d2^{2e-\gamma}$ | $2(d+1)N^{2/(d+1)}$ | $(\frac{d^2}{4}+\frac{3d}{2}+2)N^{2/(d+2)}$ | optimal $M$ |
|---|---|---|---|---|
| $d=9$ | 1428 | 1680 | 1529 | 110 |
| $d=10$ | 1252 | 1528 | 1402 | 108 |
| $d=11$ | 1157 | 1401 | 1295 | 108 |
| $d=12$ | 1090 | 1294 | 1203 | 109 |
| $d=13$ | 1026 | 1202 | 1123 | 111 |

**Example 4.6.** Our final example for discrete logarithms in a prime field is the field of size

$$2^{15474} - 2^{14954} + 2^{14432} + 1.$$

Once again, we find that Algorithm 2.1 is the best method. Based on the optimal $M$ in the degree 14 case, we estimate the NFS-security of this prime to be approximately 13180 bits.

| | $2d2^{2e-\gamma}$ | $2(d+1)N^{2/(d+1)}$ | $(\frac{d^2}{4}+\frac{3d}{2}+2)N^{2/(d+2)}$ | optimal $M$ |
|---|---|---|---|---|
| $d=12$ | 1977 | 2386 | 2217 | 137 |
| $d=13$ | 1785 | 2216 | 2070 | 135 |
| $d=14$ | 1621 | 2069 | 1941 | 135 |
| $d=15$ | 1522 | 1940 | 1827 | 136 |
| $d=16$ | 1461 | 1826 | 1726 | 138 |

The remaining examples concern the computation of logarithms in fields of degree two over their prime fields. In the case of characteristic $p$, we denote the field of degree two by $\mathbb{F}_{p^2}$. Computation of logarithms in $\mathbb{F}_{p^2}$ can be accomplished using the NFS in much the same way as described above. The major difference stems from the fact that $\mathbb{F}_{p^2}$ cannot be represented as a quotient of $\mathbb{Z}$ but instead is represented as the quotient of an order of a quadratic extension $K$ of $\mathbb{Q}$. The number field employed in the NFS is then produced by adjoining to $K$ the root of a polynomial with integral coefficients, preferably small, which has a root mod $p$, also preferably small ([13]). In the case that $p$ has small weight, the first step of Algorithm 2.1 can be used to find such a polynomial. The bound on the smoothness candidates that arises when working with $\mathbb{F}_{p^2}$ involves various factors resulting from the replacement of the base field $\mathbb{Q}$ by the quadratic field $K$. However, if we ignore these terms, which are relatively small, and if we define $M$ suitably (see [13]), then we obtain as a bound on the size of the smoothness candidates, the value $4d^2M^{d+1}2^{2(2e-\gamma)}$, where for a given $d$, both $e$ and $\gamma$ are as they would be if $p$ were input into Algorithm 2.1. Similarly, for the standard algorithms, we use as a bound on the numbers tested for smoothness the quantity

$$4(d+1)^2M^{d+1}(p^2)^{2/(d+1)}$$

in the case that a polynomial is produced by means of the techniques used for factoring and

$$\left(\frac{d^2}{4}+\frac{3d}{2}+2\right)^2M^{d+1}(p^2)^{2/(d+2)}$$

in the case that the method of Joux and Lercier is used to produce two polynomials with a shared root mod $p$ and in turn, two extensions of $K$. Thus, for the examples that follow, we compute the same bit-lengths as we did before and simply double them. We again include the value of the optimal $M$ for each finite field considered and the accompanying estimate of the NFS-security. Because the weights of the primes in these examples are so low, Algorithm 2.1 registers significant gains.

**Example 4.7** Our first example of a degree two field is the one of characteristic

$$2^{520}+2^{363}-2^{360}-1.$$

In this case, the optimal $M$ when $d=6$ corresponds to what we expect for a degree two field of approximately 880 bits. Note that the best method in the degree 2 and 5 cases is that of Joux and Lercier.

| | $4d^22^{2(2e-\gamma)}$ | $4(d+1)^2(p^2)^{2/(d+1)}$ | $(\frac{d^2}{4}+\frac{3d}{2}+2)^2(p^2)^{2/(d+2)}$ | optimal $M$ |
|---|---|---|---|---|
| $d=2$ | 685 | 699 | 526 | 52 |
| $d=3$ | 386 | 527 | 423 | 47 |
| $d=4$ | 327 | 423 | 354 | 46 |
| $d=5$ | 317 | 354 | 306 | 48 |
| $d=6$ | 216 | 305 | 269 | 45 |
| $d=7$ | 180 | 269 | 241 | 46 |
| $d=8$ | 199 | 240 | 218 | 51 |

**Example 4.8** Our next example is the degree two field of characteristic

$$2^{1582} + 2^{1551} - 2^{1326} - 1.$$

The case that $d = 7$ yields an NFS-security of approximately 2250 bits, significantly lower than the 3162 bit size of the field.

| | $4d^2 2^{2(2e-\gamma)}$ | $4(d+1)^2(p^2)^{2/(d+1)}$ | $(\frac{d^2}{4} + \frac{3d}{2} + 2)^2(p^2)^{2/(d+2)}$ | optimal $M$ |
|---|---|---|---|---|
| $d = 5$ | 799 | 1062 | 912 | 73 |
| $d = 6$ | 594 | 912 | 800 | 67 |
| $d = 7$ | 520 | 800 | 713 | 66 |
| $d = 8$ | 521 | 712 | 643 | 70 |
| $d = 9$ | 521 | 642 | 586 | 73 |
| $d = 10$ | 491 | 585 | 539 | 76 |

**Example 4.9** In this example, we consider the prime

$$2^{4231} - 2^{3907} + 2^{3847} - 1.$$

We estimate that the NFS-security of the degree two field in this case is 5850 bits, again far below the size of the field.

| | $4d^2 2^{2(2e-\gamma)}$ | $4(d+1)^2(p^2)^{2/(d+1)}$ | $(\frac{d^2}{4} + \frac{3d}{2} + 2)^2(p^2)^{2/(d+2)}$ | optimal $M$ |
|---|---|---|---|---|
| $d = 8$ | 1477 | 1889 | 1703 | 107 |
| $d = 9$ | 1243 | 1702 | 1549 | 102 |
| $d = 10$ | 1055 | 1548 | 1422 | 99 |
| $d = 11$ | 901 | 1420 | 1314 | 97 |
| $d = 12$ | 830 | 1312 | 1221 | 98 |
| $d = 13$ | 778 | 1219 | 1141 | 100 |

**Example 4.10.** Our final example is the weight 3 prime

$$2^{7746} - 2^{6704} - 1.$$

Our estimate for the NFS-security for the field in this case is merely 9770 bits.

| | $4d^2 2^{2(2e-\gamma)}$ | $4(d+1)^2(p^2)^{2/(d+1)}$ | $(\frac{d^2}{4}+\frac{3d}{2}+2)^2(p^2)^{2/(d+2)}$ | optimal $M$ |
|---|---|---|---|---|
| $d=9$ | 2093 | 3107 | 2827 | 128 |
| $d=10$ | 2093 | 2826 | 2593 | 132 |
| $d=11$ | 2093 | 2592 | 2395 | 135 |
| $d=12$ | 1802 | 2393 | 2225 | 131 |
| $d=13$ | 1502 | 2223 | 2078 | 126 |
| $d=14$ | 1250 | 2076 | 1949 | 122 |
| $d=15$ | 1062 | 1947 | 1836 | 119 |
| $d=16$ | 1127 | 1833 | 1735 | 126 |

We conclude with a comment about a recent paper of Barreto and Naehrig ([1]) in which the authors provide a method to construct pairing-friendly elliptic curves. These curves are defined over a prime field $\mathbb{F}_p$ and depend for their security on the intractibility of the discrete logarithm problem in $\mathbb{F}_{p^{12}}$. The primes proposed for the cardinality of $\mathbb{F}_p$ are of the form $36n^4+36n^3+24n^2+6n+1$ for some integer $n$. Without discussing the evident difficulty of implementing the NFS for degree 12 fields, we observe that the special form of $p$ may reduce the difficulty of computing logarithms in $\mathbb{F}_{p^{12}}$. In particular, assume that $\mathbb{F}_{p^{12}}$ is represented as a residue field of a field $K$ of degree 12 over $\mathbb{Q}$ and that we want to work in an extension of $K$ obtained by adjoining a root of a polynomial with small coefficients and a small root mod $p$. Then the polynomial $f(x)=36x^4+36x^3+24x^2+6x+1$ is an obvious choice for this polynomial so long as $p$ is small enough that the small degree of $f$ is not prohibitive. More interesting, however, from the point of view of this paper, is the fact that for any size $p$, if $n$ is a power of a small number like 2, then Algorithm 2.1 should produce a polynomial that is preferable to those produced by other methods.

**References.**

[1] S.L.M. Barreto, M. Naehrig, *Pairing-friendly elliptic curves of prime order*, Selected Areas in Cryptography – SAC 2005, Lecture Notes in Computer Science, Springer-Verlag, to appear

[2] J.P. Buhler, H.W. Lenstra, Jr., C. Pomerance, *Factoring integers with the number field sieve*, in [9], pp. 50–94

[3] E.R. Canfield, P. Erdös, C. Pomerance, *On a problem of Oppenhiem concerning "factorisatio numerorum"*, J. Number Theory 17 (1983), pp. 1–28

[4] A. Commeine, I. Semaev, *An algorithm to solve the discrete logarithm problem with the number field sieve*, preprint

[5] M. Filaseta, *A further generalization of an irreducibility theorem of A. Cohn*, Canad. J. Math. 34 (1982), pp. 1390–1395

[6] A. Joux, R. Lercier, *Improvements on the general number field sieve for discrete logarithms in prime fields*, Math. Comp. 72 (2003), pp. 953–967

[7] N. Koblitz, A. Menezes, *Pairing-based cryptography at high security levels*, Cryptography and Coding: 10th IMA International Conference, Lecture Notes in Computer Science, 3796 (2005), pp. 13-36

[8] A.K. Lenstra, *Unbelievable security: matching AES security using public key systems*, in Advances in Cryptology - ASIACRYPT 2001, LNCS 2248, Springer-Verlag (2001), pp. 67–86

[9] A.K. Lenstra, H.W. Lenstra, Jr., (eds.),*The development of the number field sieve*, LNM 1554, Springer-Verlag, 1993

[10] A.K. Lenstra, H.W. Lenstra, Jr., M.S. Manasse, J.M. Pollard, *The number field sieve*, in [9], pp. 11–42

[11] O. Schirokauer, *Discrete logs and local units*, in Theory and Applications of numbers without large prime factors, R.C. Vaughan ed., Philos. Trans. Roy. Soc. London, Ser. A, 345, Royal Society, London (1993), pp. 409–424

[12] O. Schirokauer, *The special function field sieve*, SIAM J. Discrete Math. 16 (2002), pp. 81–98

[13] O. Schirokauer, *The impact of the number field sieve on the discrete logarithm problem*, Proceedings of a conference at the Mathematical Sciences Research Institute, Algo-

rithmic Number Theory: Lattices, Number Fields, Curves, and Cryptography, Cambridge University Press, to appear

[14] O. Schirokauer, *Virtual logarithms*, J. Algorithms 57 (2005), pp. 140-147

[15] J. Solinas, *Generalized Mersenne numbers*, Technical Report CORR 99-39, University of Waterloo, 1999

[16] P. Stevenhagen, *The number field sieve*, Proceedings of a conference at the Mathematical Sciences Research Institute, Algorithmic Number Theory: Lattices, Number Fields, Curves, and Cryptography, Cambridge University Press, to appear

[17] D.H. Weidemann, *Solving sparse linear equations over finite fields*, IEEE Trans. Inform. Theory 32 (1986), pp. 54–62