Performance comparison of linear sieve and cubic sieve algorithms for discrete logarithms over prime fields

Abhijit Das and C. E. Veni Madhavan

Department of Computer Science and Automation
Indian Institute of Science, Bangalore 560 012, India
abhij,cevm@csa.iisc.ernet.in

Abstract. It is of interest in cryptographic applications to obtain practical performance improvements for the discrete logarithm problem over prime fields $\mathbb{F}_p$ with p of size ≤ 500 bits. The linear sieve and the cubic sieve methods described in Coppersmith, Odlyzko and Schroeppel's paper [3] are two practical algorithms for computing discrete logarithms over prime fields. The cubic sieve algorithm is asymptotically faster than the linear sieve algorithm.
We discuss an efficient implementation of the cubic sieve algorithm incorporating two heuristic principles. We demonstrate through empirical performance measures that for a special class of primes the cubic sieve method runs about two to three times faster than the linear sieve method *even* in cases of *small* prime fields of size about 150 bits.

1 Introduction

Computation of discrete logarithms over a finite field $\mathbb{F}_q$ is a difficult problem. No algorithms are known that solve the problem in time polynomially bounded by the size of the field (i.e. $\log q$). The index calculus algorithm [3, 7, 9–11] is currently the best known algorithm for this purpose and has a sub-exponential expected running time given by

$$L\langle q, \gamma, c\rangle = \exp\left((c + o(1))(\log q)^{\gamma}(\log\log q)^{1-\gamma}\right)$$

for some constant c and for some real number $0 < \gamma < 1$. For practical applications, one typically uses prime fields or fields of characteristic 2. In this paper, we focus on prime fields only.

Let $\mathbb{F}_p$ be a prime field of cardinality p. For an element $a \in \mathbb{F}_p$, we denote by $\overline{a}$ the representative of a in the set $\{0, 1, \ldots, p-1\}$. Let g be a primitive element of $\mathbb{F}_p$ (i.e. a generator of the cyclic multiplicative group $\mathbb{F}_p^*$). Given an element $a \in \mathbb{F}_p^*$, there exists a unique integer $0 \leq x \leq p-2$ such that $a = g^x$ in $\mathbb{F}_p$. This integer x is called the *discrete logarithm* or *index* of a in $\mathbb{F}_p$ with respect to g and is denoted by $\mathrm{ind}_g(a)$. The determination of x from the knowledge of p, g and a is referred to as the *discrete logarithm problem*. In general, one need not assume g to be a primitive element and is supposed to compute x from a and

g, if such an x exists (i.e. if a belongs to the cyclic subgroup of $\mathbb{F}_p^*$ generated by g). In this paper, we always assume for simplicity that g is a primitive element of $\mathbb{F}_p$.

In what follows we denote by $L(p, c)$ any quantity that satisfies

$$L(p,c) = L\langle p, 1/2, c\rangle = \exp\left((c + o(1))\sqrt{\ln p \ln\ln p}\right),$$

where c is a positive constant and $\ln x$ is the natural logarithm of x.[1] When p is understood from the context, we write $L[c]$ for $L(p, c)$. In particular, $L[1]$ is denoted simply by L.

The naïve index calculus algorithm [10, Section 6.6.2] for the computation of discrete logarithms over prime fields and the adaptations of this algorithm take time $L[c]$ for c between 1.5 and 2 and are not useful in practice for prime fields $\mathbb{F}_p$ with $p > 2^{100}$. Coppersmith, Odlyzko and Schroeppel [3] proposed three variants of the index calculus method that run in time $L[1]$ and are practical for $p \leq 2^{250}$. A subsequent paper [7] by LaMacchia and Odlyzko reports implementation of two of these three variants, namely the linear sieve method and the Gaussian integer method. They were able to compute discrete logarithms in $\mathbb{F}_p$ with p of about 200 bits.

The paper [3] also describes a cubic sieve algorithm due to Reyneri for the computation of discrete logarithms over prime fields. The cubic sieve algorithm has a heuristic running time of $L[\sqrt{2\alpha}]$ for some $\frac{1}{3} \leq \alpha < \frac{1}{2}$ and is, therefore, asymptotically faster than the linear sieve algorithm (and the other $L[1]$ algorithms described in [3]). However, the authors of [3] conjectured that the theoretical asymptotics do not appear to take over for p in the range of practical interest (a few hundred bits). A second problem associated with the cubic sieve algorithm is that it requires a solution of a certain Diophantine equation. It's not known how to find a solution of this Diophantine equation in the general case. For certain special primes p a solution arises naturally, for example, when p is *close* to a whole cube.

Recently, a new variant of the index calculus method based on general number field sieves (NFS) has been proposed [6] and has a conjectured heuristic run time of

$$L\langle p, 1/3, c\rangle = \exp\left((c + o(1))(\log p)^{\frac{1}{3}}(\log\log p)^{\frac{2}{3}}\right).$$

Weber et. al. [12, 14, 15] have implemented and proved the practicality of this method. Currently the NFS-based methods are known to be the fastest algorithms for solving the discrete logarithm problem over prime fields.

In this paper, we report efficient implementations of the linear sieve and the cubic sieve algorithms. To the best of our knowledge, ours is the first large-scale implementation of the cubic sieve algorithm. In our implementation, we employ ideas similar to those used in the quadratic sieve algorithm for integer factorization [1, 5, 13]. Our experiments seem to reveal that the equation collecting phase of the cubic sieve algorithm, whenever applicable, runs faster than that in the linear sieve algorithm.

[1] We denote $\log x = \log_{10} x$, $\ln x = \log_e x$ and $\lg x = \log_2 x$.

In the next two sections, we briefly describe the linear sieve and the cubic sieve algorithms. Performance of our implementation and comparison of the two algorithms for a randomly chosen prime field are presented in Sections 4 and 5. Our emphasis is not to set a record on the computation of discrete logarithms, but to point out that our heuristic principles really work in practical situations. We, therefore, experimented with a *small* prime (of length around 150 bits). Even for this field we get a performance gain between two and three. For larger prime fields, the performance improvement of the cubic sieve method over the linear sieve method is expected to get accentuated. We conclude the paper in Section 6.

2 The linear sieve algorithm

The first stage for the computation of discrete logarithms over a prime field $\mathbb{F}_p$ using the currently known subexponential methods involves calculation of discrete logarithms of elements of a given subset of $\mathbb{F}_p$, called the *factor base*. To this end, a set of linear congruences are solved modulo $p-1$. Each such congruence is obtained by checking the factorization of certain integers computed deterministically or randomly. For the linear sieve algorithm, the congruences are generated in the following way.

Let $H = \lfloor\sqrt{p}\rfloor + 1$ and $J = H^2 - p$. Then $J \le 2\sqrt{p}$. For *small* integers c_1, c_2, the right side of the following congruence (henceforth denoted as $T(c_1, c_2)$)

$$(H + c_1)(H + c_2) \equiv J + (c_1 + c_2)H + c_1c_2 \pmod{p} \tag{1}$$

is of the order of $\sqrt{p}$. If the integer $T(c_1, c_2)$ is smooth with respect to the first t primes $q_1, q_2, \ldots, q_t$, that is, if we have a factorization like $J+(c_1+c_2)H+c_1c_2 = \prod_{i=1}^{t} q_i^{\alpha_i}$, then we have a relation

$$\operatorname{ind}_g(H + c_1) + \operatorname{ind}_g(H + c_2) = \sum_{i=1}^{t} \alpha_i \operatorname{ind}_g(q_i). \tag{2}$$

For the linear sieve algorithm, the factor base comprises of primes less than $L[1/2]$ (so that by the prime number theorem $t \approx L[1/2]/\ln(L[1/2])$) and integers $H+c$ for $-M \le c \le M$. The bound M on c is chosen such that $2M \approx L[1/2+\epsilon]$ for some small positive real ϵ. Once we check the factorization of $T(c_1, c_2)$ for all values of c_1 and c_2 in the indicated range, we are expected to get $L[1/2+3\epsilon]$ relations like (2) involving the unknown indices of the factor base elements. If we further assume that the primitive element g is a small prime which itself is in the factor base, then we get a relation $\operatorname{ind}_g(g) = 1$. The resulting system with asymptotically more equations than unknowns is expected to be of full rank and is solved to compute the discrete logarithms of elements in the factor base.

In order to check for the smoothness of the integers $T(c_1, c_2) = J + (c_1 + c_2)H + c_1c_2$ for c_1, c_2 in the range $-M, \ldots, M$, sieving techniques are used. First one fixes a c_1 and initializes to zero an array $\mathfrak{A}$ indexed $-M, \ldots, M$. One then computes for each prime power q^h (q is a small prime in the factor base and h is a small positive exponent), a solution for c_2 of the congruence $(H+c_1)c_2+(J+c_1H) \equiv 0 \pmod{q^h}$. If the gcd $(H+c_1, q) = 1$, i.e. if $H+c_1$ is not a multiple of q, then the solution is given by $d \equiv -(J+c_1H)(H+c_1)^{-1} \pmod{q^h}$. The inverse in the last equation can be calculated by running the extended gcd algorithm on $H+c_1$ and q^h. Then for each value of c_2 ($-M \leq c_2 \leq M$) that is congruent to $d \pmod{q^h}$, $\lg q$ is added[2] to the corresponding array locations $\mathfrak{A}_{c_2}$. On the other hand, if $q^{h_1}||(H+c_1)$ with $h_1 > 0$, we compute $h_2 \geq 0$ such that $q^{h_2}||(J+c_1H)$. If $h_1 > h_2$, then for each value of c_2, the expression $T(c_1, c_2)$ is divisible by q^{h_2} and by no higher powers of q. So we add the quantity $h_2 \ln q$ to $\mathfrak{A}_{c_2}$ for all $-M \leq c_2 \leq M$. Finally, if $h_1 \leq h_2$, then we add $h_1 \ln q$ to $\mathfrak{A}_{c_2}$ for all $-M \leq c_2 \leq M$ and for $h > h_1$ solve the congruence as $d \equiv -\left(\frac{J+c_1H}{q^{h_1}}\right)\left(\frac{H+c_1}{q^{h_1}}\right)^{-1} \pmod{q^{h-h_1}}$.

Once the above procedure is carried out for each small prime q in the factor base and for each small exponent h,[3] we check for which values of c_2, the entry of $\mathfrak{A}$ at index c_2 is *sufficiently close* to the value $\lg(T(c_1, c_2))$. These are precisely the values of c_2 such that for the given c_1, the integer $T(c_1, c_2)$ factorizes smoothly over the small primes in the factor base.

In an actual implementation, one might choose to vary c_1 in the sequence $-M, -M+1, -M+2, \ldots$ and, for each c_1, consider only the values of c_2 in the range $c_1 \leq c_2 \leq M$. The criterion for 'sufficient closeness' of the array element $\mathfrak{A}_{c_2}$ and $\lg(T(c_1, c_2))$ goes like this. If $T(c_1, c_2)$ factorizes smoothly over the small primes in the factor base, then it should differ from $\mathfrak{A}_{c_2}$ by a small positive or negative value. On the other hand, if the former is not smooth, it would have a factor at least as small as q_{t+1}, and hence the difference between $\lg(T(c_1, c_2))$ and $\mathfrak{A}_{c_2}$ would not be too less than $\lg q_{t+1}$. In other words, this means that the values of the difference $\lg(T(c_1, c_2)) - \mathfrak{A}_{c_2}$ for smooth values of $T(c_1, c_2)$ are well-separated from those for non-smooth values and one might choose for the criterion a check whether the absolute value of the above difference is less than 1.

This completes the description of the equation collecting phase of the first stage of the linear sieve algorithm. This is followed by the solution of the linear system modulo $p-1$. The second stage of the algorithm involves computation of discrete logarithms of arbitrary elements of $\mathbb{F}_p^*$ using the database of logarithms of factor base elements. We do not deal with these steps in this paper, but refer the reader to [3, 7, 8] for details.

---

[2] More precisely, some approximate value of $\lg q$, say, for example, the integer $\lfloor 1000 \lg q \rfloor$.

[3] The exponent h can be chosen in the sequence $1, 2, 3, \ldots$ until one finds an h for which none of the integers between $-M$ and M is congruent to d.

3 The cubic sieve algorithm

Let us assume that we know a solution of the Diophantine equation

$$\begin{aligned} X^3 &\equiv Y^2 Z \pmod p \\ X^3 &\neq Y^2 Z \end{aligned} \tag{3}$$

with X, Y, Z of the order of p^α for some $\frac{1}{3} \leq \alpha < \frac{1}{2}$. Then we have the congruence

$$\begin{aligned} &(X+AY)(X+BY)(X+CY) \equiv \\ &\qquad Y^2\left[Z + (AB+AC+BC)X + (ABC)Y\right] \pmod p \end{aligned} \tag{4}$$

for all triples (A, B, C) with $A + B + C = 0$. If the bracketed expression on the right side of the above congruence, henceforth denoted as $R(A, B, C)$, is smooth with respect to the first t primes $q_1, q_2, \ldots, q_t$, that is, if we have a factorization $R(A, B, C) = \prod_{i=1}^{t} q_i^{\beta_i}$, then we have a relation like

$$\begin{aligned} &\mathrm{ind}_g(X+AY) + \mathrm{ind}_g(X+BY) + \mathrm{ind}_g(X+CY) \equiv \\ &\qquad 2\,\mathrm{ind}_g(Y) + \textstyle\sum_{i=1}^{t} \beta_i\,\mathrm{ind}_g(q_i) \pmod{p-1} \end{aligned} \tag{5}$$

If A, B, C are *small* integers, then $R(A, B, C)$ is of the order of p^α, since each of X, Y and Z is of the same order. This means that we are now checking integers smaller than $O(p^{\frac{1}{2}})$ for smoothness over first t primes. As a result, we are expected to get relations like (5) more *easily* than relations like (2) as in the linear sieve method.

This observation leads to the formulation of the cubic sieve algorithm as follows. The factor base comprises of primes less than $L[\sqrt{\alpha/2}]$ (so that $t \approx L[\sqrt{\alpha/2}]/\ln\left(L[\sqrt{\alpha/2}]\right)$), the integer Y (or Y^2) and the integers $X + AY$ for $0 \leq |A| \leq M$, where M is of the order of $L[\sqrt{\alpha/2}]$. The integer $R(A, B, C)$ is, therefore, of the order of $p^\alpha L[\sqrt{3\alpha/2}]$ and hence the probability that it is smooth over the first t primes selected as above, is about $L[-\sqrt{\alpha/2}]$. As we check the smoothness for $L[\sqrt{2\alpha}]$ triples (A, B, C) (with $A + B + C = 0$), we expect to obtain $L[\sqrt{\alpha/2}]$ relations like (5).

In order to check for the smoothness of $R(A, B, C) = Z + (AB + AC + BC)X + (ABC)Y$ over the first t primes, sieving techniques are employed. We maintain an array $\mathfrak{A}$ indexed $-M \ldots +M$ as in the linear sieve algorithm. At the beginning of each sieving step, we fix C, initialize the array $\mathfrak{A}$ to zero and let B vary. The relation $A + B + C = 0$ allows us to eliminate A from $R(A, B, C)$ as $R(A, B, C) = -B(B + C)(X + CY) + (Z - C^2X)$. For a fixed C, we try to solve the congruence

$$-B(B+C)(X+CY) + (Z - C^2X) \equiv 0 \pmod{q^h} \tag{6}$$

where q is a small prime in the factor base and h is a small positive exponent. This is a quadratic congruence in B. If $X + CY$ is invertible modulo q^h (i.e. modulo q), then the solution for B is given by

$$B \equiv -\frac{C}{2} + \sqrt{(X + CY)^{-1}(Z - C^2X) + \frac{C^2}{4}} \pmod{q^h} \tag{7}$$

where the square root is modulo q^h. If the expression inside the radical is a quadratic residue modulo q^h, then for each solution d of B in (7), $\lg q$ is added to those indices of $\mathfrak{A}$ which are congruent to d modulo q^h. On the other hand, if the expression under the radical is a quadratic non-residue modulo q^h, we have no solutions for B in (6). Finally, if $X + CY$ is non-invertible modulo q, we compute $h_1 > 0$ and $h_2 \geq 0$ such that $q^{h_1} || (X + CY)$ and $q^{h_2} || (Z - C^2X)$. If $h_1 > h_2$, then $R(A, B, C)$ is divisible by q^{h_2} and by no higher powers of q for each value of B (and for the fixed C). We add $h_2 \lg q$ to $\mathfrak{A}_i$ for each $-M \leq i \leq M$. On the other hand, if $h_1 \leq h_2$, we add $h_1 \lg q$ to $\mathfrak{A}_i$ for each $-M \leq i \leq M$ and try to solve the congruence $-B(B + C)\left(\frac{X+CY}{q^{h_1}}\right) + \left(\frac{Z-C^2X}{q^{h_1}}\right) \equiv 0 \pmod{q^{h-h_1}}$ for $h > h_1$. Since $\frac{X+CY}{q^{h_1}}$ is invertible modulo q^{h-h_1}, this congruence can be solved similar to (7).

Once the above procedure is carried out for each small prime q in the factor base and for each small exponent h, we check for which values of B, the entry of $\mathfrak{A}$ at index B is *sufficiently close* to the value $\lg(R(A, B, C))$. These are precisely the values of B for which $R(A, B, C)$ is smooth over the first t primes for the given C. The criterion of 'sufficient closeness' of $\mathfrak{A}_B$ and $\lg(R(A, B, C))$ is the same as described in connection with the linear sieve algorithm.

In order to avoid duplication of effort, we should examine the smoothness of $R(A, B, C)$ for $-M \leq A \leq B \leq C \leq M$. With this condition, it can be easily shown that C varies from 0 to M and for a fixed C, B varies from $-C/2$ to $\min(C, M - C)$. Though we do not use the value of A directly in the sieving procedure described above, it's useful[4] to note that for a fixed C, A varies from $\max(-2C, -M)$ to $-C/2$. In particular, A is always negative.

After sufficient number of relations are available, the resulting system is solved modulo $p - 1$ and the discrete logarithms of the factor base elements are stored for computation of individual discrete logarithms. We refer the reader to [3, 7, 8] for details on the solution of sparse linear systems and on the computation of individual discrete logarithms with the cubic sieve method.

Attractive as it looks, the cubic sieve method has several drawbacks which impair its usability in practical situations.

1. It is currently not known how to solve the congruence (3) for a general p. And even when it is solvable, how large can α be? For practical purposes α should be as close to $\frac{1}{3}$ as possible. No non-trivial results are known to the authors, that can classify primes p according as the smallest possible values of α they are associated with.

[4] for a reason that will be clear in Section 5

2. Because of the quadratic and cubic expressions in A, B and C as coefficients of X and Y in $R(A,B,C)$, the integers $R(A,B,C)$ tend to be as large as $p^{\frac{1}{2}}$ even when α is equal to $1/3$. If we compare this scenario with that for $T(c_1,c_2)$ (See Equation (1)), we see that the coefficient of H is a linear function of c_1 and c_2 and as such, the integers $T(c_1,c_2)$ are larger than $p^{\frac{1}{2}}$ by a small multiplicative factor. This shows that though the integers $R(A,B,C)$ are asymptotically smaller than the integers $T(c_1,c_2)$, the formers are, in practice, around 10^4–10^6 times smaller than latter ones, even when α assumes the most favorable value (namely, $1/3$). In other words, when one wants to use the cubic sieve algorithm, one should use values of t (i.e. the number of small primes in the factor base) much larger than the values prescribed by the asymptotic formula for t.
3. The second stage of the cubic sieve algorithm, i.e. the stage that involves computation of individual logarithms, is asymptotically as slow as the equation collection stage. For the linear sieve algorithm, on the other hand, individual logarithms can be computed much faster than the equation collecting phase.

In this paper, we address this second issue related to the cubic sieve algorithm. We report an efficient implementation of the cubic sieve algorithm for the case $\alpha = 1/3$, that runs faster than the linear sieve method for the same prime. Our experimentation tends to reveal that the cubic sieve algorithm, when applicable, outperforms the linear sieve method, even when the cardinality of the ground field is around 150 bits long.

4 An efficient implementation of the linear sieve method

Before we delve into the details of the comparison of the linear and cubic sieve methods, we describe an efficient implementation of the linear sieve algorithm. The tricks that help us speed up the equation collecting phase of the linear sieve method are very similar to those employed in the quadratic sieve algorithm for integer factorization (See [1, 5, 13] for details).

We first recall that at the beginning of each sieving step, we find a solution for c_2 modulo q^h in the congruence $T(c_1,c_2) \equiv 0 \pmod{q^h}$ for every small prime q in the factor base and for a set of small exponents h. The costliest operation that need be carried out for each such solution is the computation of a modular inverse (namely, that of $H + c_1$ modulo q^h). As described in [7] and as is evident from our experiments too, calculations of these inverses take more than half of the CPU time needed for the entire equation collecting stage. Any trick that reduces the number of computations of the inverses, speeds up the algorithm.

One way to achieve this is to solve the congruence every time only for $h = 1$ and ignore all higher powers of q. That is, for every q (and c_1), we check which of the integers $T(c_1,c_2)$ are divisible by q and then add $\lg q$ to the corresponding indices of the array $\mathfrak{A}$. If some $T(c_1,c_2)$ is divisible by a higher power of q, this strategy fails to add $\lg q$ the required number of times. As a result, this $T(c_1,c_2)$, even if smooth, may fail to pass the 'closeness criterion' described in Section 2. This is, however, not a serious problem, because we may increase the cut-off from a value smaller than $\lg q_t$ to a value $\zeta \lg q_t$ for some $\zeta \geq 1$. This means that some non-smooth $T(c_1, c_2)$ will pass through the selection criterion in addition to some smooth ones that could not, otherwise, be detected. This is reasonable, because the non-smooth ones can be later filtered out from the smooth ones and one might use even trial divisions to do so. For primes p of less than 200 bits, values of $\zeta \leq 2.5$ work quite well in practice [1, 13].

The reason why this strategy performs well in practice is as follows. If q is small, for example $q = 2$, we should add *only* 1 to $\mathfrak{A}_{c_2}$ for every power of 2 dividing $T(c_1, c_2)$. On the other hand, if q is much larger, say $q = 1299709$ (the 10^5th prime), then $\lg q \approx 20.31$ is *large*. But $T(c_1, c_2)$ would not be, in general, divisible by a *high* power of this q. The approximate calculation of logarithm of the smooth part of $T(c_1, c_2)$, therefore, leads to a situation where the probability that a smooth $T(c_1, c_2)$ is actually detected as smooth is quite high. A few relations would be still missed out even with the modified 'closeness criterion', but that is more than compensated by the speed-up gained by the method.

The above strategy helps us in a way other than by reducing the number of modular inverses. We note that for practical values of p, the small primes in the factor base are usually single-precision ones. As a result, the computation of d (See Section 2) can be carried out using single-precision operations only.

Throughout the rest of this section we compare the performance of the modified strategy with that of the original strategy for a value of p of length around 150 bits. This prime is chosen as a random one satisfying the conditions (i) $(p-1)/2$ is also a prime, and (ii) p is close to a whole cube. This second condition is necessary, because for these primes, the cubic sieve algorithm is also applicable, so that we can compare the performance of the two sieve algorithms for these primes. Our experiments are based on the Galois Field Library routines developed by the authors [4] and are carried out on a 200 MHz Pentium machine running Linux version 2.0.34 and having 64 Mb RAM. The GNU C Compiler version 2.7 is used.

Table 1. Performance of the linear sieve algorithm

$p = 1320245474656309183513988729373583242842871683$
$t = 7000,\ M = 30000$

Algorithm	ζ	No. of Relations ($\bar{\rho}$)	No. of Variables ($\bar{\nu}$)	$\bar{\rho}/\bar{\nu}$	CPU Time (seconds)
Exact	0.1	108637	67001	1.6214	225590
Approximate	1.0	108215	67001	1.6151	101712
	1.5	108624	67001	1.6212	101818
	2.0	108636	67001	1.6214	102253
	2.5	108637	67001	1.6214	102250

In Table 1 we compare the performance of the 'exact' version of the algorithm (where all relations are made available by choosing values of $h \geq 1$) with that of the 'approximate' version of the algorithm (in which powers $h > 1$ are neglected). The CPU times listed in the table do not include the time for filtering out the 'spurious' relations obtained in the approximate version. It is evident from the table that the performance gain obtained using the heuristic variant is more than 2. It's also clear that values of ζ between 1.5 and 2 suffice for fields of this size.

5 An efficient implementation of the cubic sieve method

For the cubic sieve method, we employ strategies similar to those described in the last section. That is, we solve the congruence $R(A, B, C) \equiv 0 \pmod{q}$ for each small prime q in the factor base and ignore higher powers of q that might divide $R(A, B, C)$. As before, we set the cut-off at $\zeta \lg q_t$ for some $\zeta \geq 1$. We are not going to elaborate the details of this strategy and the expected benefits once again in this section. We concentrate on an additional heuristic modification of the equation collecting phase instead.

We recall from Section 3 that we check the smoothness of $R(A, B, C)$ for $-M \leq A \leq B \leq C \leq M$. With this condition, C varies from 0 to M. We note that for each value of C, we have to execute the entire sieving operation once. For each such sieving operation (that is, for a fixed C), the sieving interval for B is (i.e. the admissible values of B are) $-C/2 \leq B \leq \min(C, M - C)$. Correspondingly $A = -(B+C)$ can vary from $\max(-2C, -M)$ to $-C/2$. It's easy to see that in this case total number of triples (A, B, C) for which the smoothness of $R(A, B, C)$ is examined is $\tau = \sum_{C=0}^{M} \Big(1 + \lfloor C/2 \rfloor + \min(C, M - C)\Big) \approx M^2/2$. The number of unknowns, that is, the size of the factor base, on the other hand, is $\nu \approx 2M + t$.

If we remove the restriction $A \geq -M$ and allow A to be as negative as $-\lambda M$ for some $1 < \lambda \leq 2$, then we are benefitted in the following way. As before, we allow C to vary from 0 to M keeping the number of sieving operations fixed. Since A can now assume values smaller than $-M$, the sieving interval increases to $-C/2 \leq B \leq \min(C, \lambda M - C)$. As a result, the total number of triples (A, B, C) becomes $\tau_\lambda = \sum_{C=0}^{M} \Big(1 + \lfloor C/2 \rfloor + \min(C, \lambda M - C)\Big) \approx \frac{M^2}{4}(4\lambda - \lambda^2 - 1)$, whereas the size of the factor base increases to $\nu_\lambda \approx (\lambda + 1)M + t$. (Note that with this notation the value $\lambda = 1$ corresponds to the original algorithm and $\tau = \tau_1$ and $\nu = \nu_1$.) The ratio τ_λ/ν_λ is approximately proportional to the number of smooth integers $R(A, B, C)$ generated by the algorithm divided by the number of unknowns. Therefore, λ should be set at a value for which this ratio is maximum. If one treats t and M as constants, then the maximum is attained at $\lambda^* = -U + \sqrt{U^2 + 4U + 1}$, where $U = \frac{M + t}{M} = 1 + \frac{t}{M}$. As we increase U from 1 to ∞ (or, equivalently the ratio t/M from 0 to ∞), the value of λ^* increases monotonically from $\sqrt{6}-1 \approx 1.4495$ to 2. In the following table (Table 2), we summarize the variation of τ_λ/ν_λ for some values of U. These values of U correspond from left to right to $t \ll M$, $t \approx M/2$, $t \approx M$ and $t \approx 2M$ respectively. The corresponding values of λ^* are respectively 1.4495, 1.5414, 1.6056 and 1.6904. It's clear from the table, that for practical ranges of values of U, the choice $\lambda = 1.5$ gives performance quite close to the optimal.

Table 2. Variation of τ_λ/ν_λ with λ

	τ_λ/ν_λ (approx)			
λ	$U=1$	$U=1.5$	$U=2$	$U=3$
1	0.2500 M	0.2000 M	0.1667 M	0.1250 M
1.5	0.2750 M	0.2292 M	0.1964 M	0.1527 M
2	0.2500 M	0.2143 M	0.1875 M	0.1500 M
λ^*	0.2753 M	0.2293 M	0.1972 M	0.1548 M

We note that this scheme keeps M and the range of variation of C constant and hence does not increase the number of sieving steps and, in particular, the number of modular inverses and square roots. It is, therefore, advisable to apply the trick (with, say, $\lambda = 1.5$) instead of increasing M. With that one is expected to get a speed-up of about 10 to 20% and obtain a larger database.

In what follows, we report about the performance of the cubic sieve algorithm for various values of the parameters ζ and λ. We also compare the performance of the cubic sieve algorithm with that of the linear sieve algorithm. We work in the prime field $\mathbb{F}_p$ with

$$p = 1320245474656309183513988729373583242842871683$$

as in the last section. For this prime, we have

$$X = \lfloor \sqrt[3]{p} \rfloor + 1 = 1097029305312372, Y = 1, Z = 31165$$

as a solution of (3).

To start with, we fix $\lambda = 1.5$ and examine the variation of the performance of the equation collecting stage with ζ. We did not implement the 'exact' version of this algorithm in which one tries to solve (6) for exponents $h > 1$ of q. Table 3 lists the experimental details for the 'approximate' algorithm. As in Table 1, the CPU times do not include the time for filtering out the spurious relations available by the more generous closeness criterion for the approximate algorithm. For the cubic sieve method, the values of ζ around 1.5 works quite well for our prime p.

In Table 4, we fix ζ at 1.5 and tabulate the variation of the performance of the cubic sieve algorithm for some values of λ. It's clear from the table that among the cases observed, the largest value of the ratio $\bar{\rho}/\bar{\nu}$ is obtained at $\lambda = 1.5$. (The theoretical maximum is attained at $\lambda \approx 1.6$) We also note that changing

Table 3. Performance of the cubic sieve algorithm for various values of ζ

$p = 132024547465630918351398872937358324284287168 3$

$t = 10000,\ M = 10000,\ \lambda = 1.5$

ζ	No. of Relations ($\bar{\rho}$)	No. of Variables ($\bar{\nu}$)	$\bar{\rho}/\bar{\nu}$	CPU Time (seconds)
1.0	54805	35001	1.5658	43508
1.5	54865	35001	1.5675	43336
2.0	54868	35001	1.5676	43492

the value of λ incurs variation in the running time by at most 1%. Thus our heuristic allows us to build a larger database at approximately no extra cost.

Table 4. Performance of the cubic sieve algorithm for various values of λ

$p = 132024547465630918351398872937358324284287168 3$

$t = 10000,\ M = 10000,\ \zeta = 1.5$

λ	No. of Relations ($\bar{\rho}$)	No. of Variables ($\bar{\nu}$)	$\bar{\rho}/\bar{\nu}$	CPU Time (seconds)
1.0	43434	30001	1.4478	43047
1.5	54865	35001	1.5675	43336
1.6	56147	36001	1.5596	43347
2.0	58234	40001	1.4558	43499

5.1 Performance comparison with linear sieve

The speed-up obtained by the cubic sieve method over the linear sieve method is about 2.5 for the field of size around 150 bits. For larger fields, this speed-up is expected to be more. It is, therefore, evident that the cubic sieve algorithm, at least for the case $\alpha = 1/3$, runs faster than the linear sieve counterpart for the practical range of sizes of prime fields.

6 Conclusion

In this paper, we have described various practical aspects for efficient implementation of the linear and the cubic sieve algorithms for the computation of discrete logarithms over finite fields. We have also compared the performances of these two algorithms and established the superiority of the latter method over the former for the cases when p is close to a whole cube. It, however, remains unsettled whether the cubic sieve algorithm performs equally well for a general prime p. More importantly, the applicability of the cubic sieve algorithm banks on the availability of a 'favorable' solution of a certain Diophantine equation.

Finding an algorithm for computing the solution of this Diophantine equation or even for certifying if a solution exists, continues to remain an open problem and stands in the way of the general acceptance of the cubic sieve algorithm. Last but not the least, we need performance comparison of the cubic sieve method with the number field sieve method.

References

1. Bressoud, D.M.: Factorization and Primality Testing, UTM, Springer-Verlag, 1989.
2. Cohen, H.: A course in computational algebraic number theory, GTM **138**, Springer-Verlag, 1993.
3. Coppersmith, D., Odlyzko, A.M., Schroeppel, R.: Discrete logarithms in $GF(p)$, Algorithmica **1** (1986), 1–15.
4. Das, A., Veni Madhavan, C.E.: Galois field library: Reference manual, Technical report No. IISc-CSA-98-05, Department of Computer Science and Automation, Indian Institute of Science, Feb 1998.
5. Gerver, J.: Factoring large numbers with a quadratic sieve, Math. Comp. **41** (1983), 287–294.
6. Gordon, D.M.: Discrete logarithms in $GF(p)$ using the number field sieve, SIAM Journal of Discrete Mathematics **6** (1993), 124–138.
7. LaMacchia, B.A., Odlyzko, A.M.: Computation of discrete logarithms in prime fields, Designs, Codes, and Cryptography **1** (1991), 46–62.
8. LaMacchia, B.A., Odlyzko, A.M.: Solving large sparse linear systems over finite fields, Advances in Cryptology – CRYPTO'90, A. J. Menezes and S. A. Vanstone (eds.), LNCS **537** (1991), Springer-Verlag, 109–133.
9. McCurley, K.S.: The discrete logarithm problem, Cryptology and Computational Number Theory, Proc. Symp. in Appl. Math. **42** (1990), 49–74.
10. Menezes, A.J., ed.: 'Applications of finite fields', Kluwer Academic Publishers, 1993.
11. Odlyzko, A.M.: Discrete logarithms and their cryptographic significance, Advances in Cryptology: Proceedings of Eurocrypt'84, LNCS **209** (1985), Springer-Verlag, 224–314.
12. Schirokauer, O., Weber, D., Denny, T.: Discrete logarithms: the effectiveness of the index calculus method, Proc. ANTS II, LNCS **1122** (1996), Springer-Verlag, 337–361.
13. Silverman, R.D.: The multiple polynomial quadratic sieve, Math. Comp. **48** (1987), 329–339.
14. Weber, D.: Computing discrete logarithms with the general number field sieve, Proc. ANTS II, LNCS **1122** (1996), Springer-Verlag, 99–114.
15. Weber, D., Denny, T.: The solution of McCurley's discrete log challenge, Crypto'98, LNCS **1462** (1998), Springer-Verlag, 458–471.