Draft. Aimed at *Mathematics of Computation.*

FASTER SQUARE ROOTS IN ANNOYING FINITE FIELDS

DANIEL J. BERNSTEIN

ABSTRACT. Let q be an odd prime number. There are several methods known to compute square roots in $\mathbf{Z}/q$: the quadratic-extension methods of Legendre, Pocklington, Cipolla, Lehmer, et al., and the discrete-logarithm methods of Tonelli, Shanks, et al. The quadratic-extension methods use $(3+o(1))\lg q$ multiplications and, on average, $2+o(1)$ Jacobi-symbol computations mod q. The discrete-logarithm methods use only $(1+o(1))\lg q$ multiplications, after an easy precomputation of one element of $\mathbf{Z}/q$, if $\operatorname{ord}_2(q-1) \in o(\sqrt{\lg q})$. This paper presents an algorithm that uses only $(1+o(1))\lg q$ multiplications, after an easy precomputation of $(\lg q)^{O(1)}$ elements of $\mathbf{Z}/q$, if $\operatorname{ord}_2(q-1) \in o(\sqrt{\lg q}\lg\lg q)$. For example, the new algorithm can compute square roots in $\mathbf{Z}/q$ for $q = 2^{224} - 2^{96} + 1$ using 364 multiplications in $\mathbf{Z}/q$ and 1024 precomputed elements of $\mathbf{Z}/q$. The same technique speeds up the Silver-Pohlig-Hellman algorithm for computing discrete logarithms in any cyclic group of smooth order.

1. INTRODUCTION

This paper considers the problem of computing square roots in a finite field F of odd cardinality q.

If $q \in 3+4\mathbf{Z}$, and if u is a square in F, then $u^{(q+1)/4}$ is a square root of u. Indeed, if $u = x^2$ then $u = x^{q+1} = u^{(q+1)/2} = (u^{(q+1)/4})^2$. One can compute $u^{(q+1)/4}$ with $(1+o(1))\lg q$ multiplications in F by Brauer's exponentiation algorithm in [14].

One cannot expect such a simple algorithm to work for $q \in 1+4\mathbf{Z}$: there are square roots of -1 in F, but they are not powers of -1. Can we compute square roots in F as quickly as we can exponentiate?

For most prime powers q, the answer is yes, if we are given more information about F: namely, an element g of order 2^n in F, where $n = \operatorname{ord}_2(q-1)$. Assume that u is nonzero; then the power $u^{(q-1)/2^n}$ is a power of g. Usually n is small, so one can quickly find a discrete logarithm of $u^{(q-1)/2^n}$ base g, and then a square root of u.

There are, however, some annoying prime powers q for which $\operatorname{ord}_2(q-1)$ is large. For example, the NIST P-224 elliptic curve in [4], standardized by the United States government for cryptographic applications, is the curve $y^2 = x^3 - 3x + c_6$ over $\mathbf{Z}/q$, where c_6 is a particular constant and q is the prime $2^{224} - 2^{96} + 1$. The first step of compressed P-224 point multiplication, as discussed in [12], is to compute one of the possibilities for y, given x; in other words, to compute a square root modulo q. The best discrete-logarithm methods in the literature use thousands of multiplications for this prime.

Date: 20011123.
1991 *Mathematics Subject Classification.* Primary 11Y16.
The author was supported by the National Science Foundation under grant CCR–9983950.

An alternative is Cipolla's algorithm in [17], which writes u as the norm of an element of a random quadratic field extension of F. Then the $(q+1)/2$ power of that element is a square root of u. One can compute this power with $(3+o(1))\lg q$ multiplications in F. There are other quadratic-extension methods that run at similar speed. These methods still take more than 700 multiplications for $q = 2^{224} - 2^{96} + 1$.

This paper presents an accelerated discrete-logarithm square-root algorithm that remains efficient for substantially larger $\mathrm{ord}_2(q-1)$, given more information about F. The new algorithm takes only 364 multiplications for $q = 2^{224} - 2^{96} + 1$, for example, with the help of a table of 1024 elements of F. One can further reduce the number of multiplications at the expense of a larger table: for example, 304 multiplications using a table of 3072 elements, or 258 multiplications using a table of 32768 elements.

Section 2 reviews quadratic-extension square-root methods. Section 3 reviews discrete-logarithm square-root methods and presents the new algorithm. Section 4 focuses on prime q: it discusses arithmetic in the usual representation of $\mathbf{Z}/q$, and analyzes the time taken by all these methods.

Additional techniques for non-prime q, such as computing a square root of u for $q = p^3$ by computing a square root of u^{1+p+p^2}, are not discussed in this paper.

2. Square roots via quadratic extensions

Legendre's method finds a square root of u by computing $(r-x)^{(q-1)/2}$ in the ring $F[x]/(x^2-u)$. Here r is any element of F such that $r^2 - u$ is not a square. Pocklington's method is a variant that finds a square root of $-u$. Cipolla's method finds a square root of u by computing $(r-x)^{(q+1)/2}$ in the ring $F[x]/(x^2-(r^2-u))$.

Choosing r. If $u = 0$ then $r^2 - u$ is a square. One does not use these methods to find a square root of 0. So assume $u \neq 0$.

There are exactly $(q-3)/2$ choices of r for which $r^2 - u$ is a nonzero square. (Each choice corresponds to two pairs (r, s) with $(r-s)(r+s) = u$ and $s \neq 0$. Each pair corresponds to a unique nonzero $r - s$ that is not a square root of u.) There are also exactly 2 choices of r for which $r^2 = u$. Hence there are exactly $(q-1)/2$ choices of r for which $r^2 - u$ is not a square.

One can repeatedly choose a uniform random r until $r^2 - u$ is not a square, and then use a quadratic-extension method described below to find a square root of u. This takes one exponentiation in a quadratic extension and, on average, approximately two Jacobi-symbol tests.

Or one can skip the Jacobi-symbol tests: choose a uniform random r, use a quadratic-extension method described below, and try again if the method does not find a square root of u. This takes, on average, approximately two exponentiations in quadratic extensions. Exception: Pocklington's method succeeds for more values of r if $q \in 1 + 8\mathbf{Z}$; it is tried approximately $2^n/(2^n - 2)$ times on average, where $n = \mathrm{ord}_2(q-1)$.

Instead of choosing r randomly, one can try elements of F in a fixed order. When q is prime, for example, one can try $r = 1$, then $r = 2$, etc. It is believed that a small r exists for every u. Another strategy for choosing r, for the sake of faster exponentiations, is discussed in section 4.

Legendre's method. The most popular method to find the roots of a polynomial $g \in F[x]$ is to compute $\gcd\left\{g, (x-r)^{(q-1)/2} - 1\right\}$ for various $r \in F$. This gcd is, as illustrated in the following theorem, the product of $x - s$ for all roots s of g such that $s - r$ is a nonzero square in F.

This method has been discovered many times and is known by many different names. According to [7], it was introduced by Legendre in [24].

If $g(r)$ is not a square, and g splits completely as $\prod(s - x)$, then the gcd is guaranteed to be a nontrivial factor of g. The suggestion to do Jacobi-symbol tests in this context, at least for $\deg g = 2$, was made by Berlekamp in [8], [9, chapter 6], and [10, section 7].

Theorem 2.1. *Let F be a finite field of odd cardinality q. Let a_0, a_1, r, u be elements of F such that u is a square in F, $r^2 - u$ is not a square in F, and $a_0 + a_1 x$ is the $(q-1)/2$ power of $r - x$ in the ring $F[x]/(x^2 - u)$. Then $a_0 = 0$ and $(1/a_1)^2 = u$.*

Proof. Select a square root v of u. The product $(r - v)(r + v) = r^2 - v^2 = r^2 - u$ is not a square, so $(r-v)^{(q-1)/2}(r+v)^{(q-1)/2} = -1$.

There is a ring morphism $\varphi : F[x]/(x^2-u) \to F \times F$ that maps x to $(v, -v)$. Now $(a_0+a_1v, a_0-a_1v) = \varphi(a_0+bx) = \varphi(r-x)^{(q-1)/2} = ((r-v)^{(q-1)/2}, (r+v)^{(q-1)/2}) = (\pm 1, \mp 1)$ so $2a_0 = (\pm 1) + (\mp 1) = 0$ and $2a_1 v = (\pm 1) - (\mp 1) = \pm 2$. □

Pocklington's method. In the context of Theorem 2.1, assume that $q \in 1 + 4\mathbf{Z}$, and write $b_0 + b_1 x$ for the $(q-1)/4$ power of $r - x$ in $F[x]/(x^2 - u)$. Then $(b_0/b_1)^2 = -u$, since $b_0^2 + b_1^2 u + 2b_0 b_1 x = (b_0 + b_1 x)^2 = a_0 + a_1 x = a_1 x$. We thus obtain a square root of $-u$.

More generally, fix $k \geq 1$, and assume that $q \in 1 + 2^{k+1}\mathbf{Z}$. If $(r-v)^{(q-1)/2^k} = -(r+v)^{(q-1)/2^k}$, where v is a square root of u, then $(b_0/b_1)^2 = -u$, where $b_0 + b_1 x$ is the $(q-1)/2^{k+1}$ power of $r - x$ in $F[x]/(x^2 - u)$. This happens with probability $1/2^k$ if r is a uniform random nonzero element of F.

One may quickly check this condition for all $k \in \{1, 2, \ldots, n-1\}$, where $n = \operatorname{ord}_2(q-1)$, by first computing the $(q-1)/2^n$ power of $r - x$, then successively squaring through the $(q-1)/2$ power. If $(b_0 + b_1 x)^2 \in Fx$ then $(b_0/b_1)^2 = -u$. This happens with probability $1/2 + 1/4 + \cdots + 1/2^{n-1} = 1 - 1/2^{n-1}$ if r is a uniform random nonzero element of F.

This variant of Legendre's method is often credited to Peralta, who published it in [30]. However, it was actually introduced by Pocklington in [34], according to [7]. The connection between [30] and Legendre's method was pointed out by Bach in [6], along with the suggestion to use Jacobi-symbol tests in this context. I still have to check what [22] and [23] say about this.

Cipolla's method. The following square-root method was introduced by Cipolla in [17], according to [7]. I still have to read [25], [29], and [43].

Theorem 2.2. *Let F be a finite field of odd cardinality q. Let a_0, a_1, r, u be elements of F such that u is a square in F, $r^2 - u$ is not a square in F, and $a_0 + a_1 x$ is the $(q+1)/2$ power of $r - x$ in the ring $F[x]/(x^2 - (r^2 - u))$. Then $a_1 = 0$ and $a_0^2 = u$.*

Proof. $x^q = x(x^2)^{(q-1)/2} = x(r^2 - u)^{(q-1)/2} = -x$ so $(r - x)^q = r + x$. Hence $a_0^2 + a_1^2(r^2 - u) + 2a_0 a_1 x = (a_0 + a_1 x)^2 = (r - x)^{q+1} = (r + x)(r - x) = r^2 - x^2 = u$. If $a_1 \neq 0$ then $a_0 = 0$, so $r^2 - u = u/a_1^2$, so $r^2 - u$ is a square, contradiction. Thus $a_1 = 0$ and $a_0^2 = u$. □

3. Square roots via discrete logarithms

The following square-root method is often credited to Shanks in [39]. According to [7], the basic idea was published by Tonelli in [41]. Some crucial improvements were introduced by Shanks, as discussed below. I still have to read [42], [39], [5], [20], and [21].

Theorem 3.1. *Let F be a finite field of odd cardinality q. Write $n = \operatorname{ord}_2(q-1)$ and $m = (q-1)/2^n$. Let g, r, u, v be elements of F such that $g = r^m$, r is not a square, u is nonzero, and $v = u^{(m-1)/2}$. Then there is an integer e in $\{0, 1, \ldots, 2^n - 1\}$ such that $uv^2 = g^e$. If u is a square then e is even and $(uvg^{-e/2})^2 = u$.*

Proof. $g^{2^n} = r^{2^n m} = r^{q-1} = 1$, but $g^{2^{n-1}} = r^{2^{n-1}m} = r^{(q-1)/2} = -1$, so g has order 2^n; and $uv^2 = u^m$ is a 2^nth root of 1, so it is a power of g, say g^e. If u is a square then $g^{2^{n-1}e} = u^{(q-1)/2} = 1$ so e is even; and $(uvg^{-e/2})^2 = u^2v^2g^{-e} = uu^mg^{-e} = u$. □

Precomputation. One can quickly find a non-square r by trying uniform random elements of F. This takes $2 + o(1)$ Jacobi-symbol tests on average, as in section 2. One can then compute $g = r^m$.

The same g works for every u in Theorem 3.1, so the cost of finding r and g can be ignored if there are many square roots to be computed in F.

Exponentiation. The first step of handling u is to compute $v = u^{(m-1)/2}$. For most prime powers q, namely those where n is small, the computation of v is the bulk of the square-root computation.

There is another exponentiation at the end of the square-root computation: $g^{-e/2}$. Tonelli and Shanks do this exponentiation bit by bit. One can save time, for large n, by using Brauer's algorithm. One can save even more time by precomputing some powers of g, with Yao's method in [44] or Pippenger's much more powerful method in [31]. Beware that the most important ideas in Pippenger's method were republished by, and are often incorrectly credited to, Brickell, Gordon, McCurley, and Wilson in [15].

I use the following special case of Pippenger's method. Select a positive integer w. Write $e/2$ in the form $2^{n-w}d_\ell + 2^{n-2w}d_{\ell-1} + \cdots + 2^{n-\ell w}d_1 + d_0$ where $\ell = \lceil n/w \rceil - 1$ and where each d_k is in the obvious range. Then

$$g^{-e/2} = g^{-2^{n-w}d_\ell} g^{-2^{n-2w}d_{\ell-1}} \cdots g^{-2^{n-\ell w}d_1} g^{-d_0}.$$

This can be computed with just ℓ multiplications of precomputed powers of g.

Discrete logarithm. The tricky part of the square-root computation, for the occasional prime powers q where n is large, is the computation of e given g^e.

Tonelli and Shanks compute e bit by bit. Say $e = e_0 + 2e_1 + 4e_2 + \cdots$ with each $e_k \in \{0, 1\}$. The strategy is to assume $e_0 = 0$, compute e_1, compute e_2, etc. Given g^e, and given $e_0, \ldots, e_{k-1}$, Tonelli and Shanks compute e_k from

$$(g^e g^{-(e \bmod 2^k)})^{2^{n-1-k}} = g^{2^{n-1}e_k} = \begin{cases} 1 & \text{if } e_k = 0, \\ -1 & \text{if } e_k = 1. \end{cases}$$

Tonelli's algorithm, as reported in [7, exercise 7.3], computes $g^{e2^{n-1-k}} = u^{m2^{n-1-k}}$ starting from u at each step, and computes $g^{-(e \bmod 2^k)2^{n-1-k}} = r^{-m(e \bmod 2^k)2^{n-1-k}}$ starting from r at each step. Shanks instead keeps track of $g^e g^{-(e \bmod 2^k)}$ as k increases.

The time-consuming part of this computation is all the squarings: computing something to the power 2^{n-2}, something to the power 2^{n-3}, something to the power 2^{n-4}, etc. Overall there are about $n^2/2$ squarings.

Shanks observes that if $e_k = 0$ then the next "something" is the same as the current one, so all the computations for the next k have already been done. This reduces the number of squarings to $n^2/4$ for an average u; see [27] and [28] for a precise analysis. Users concerned about the distribution of u can compute $\sqrt{u}$ as $\sqrt{uz^2}/z$ where z is a uniform random nonzero element of F.

Accelerated discrete logarithm. I compute e several bits at a time, reusing Pippenger's precomputed table of powers of g.

Say $e = 2^{n-w}e_\ell + 2^{n-2w}e_{\ell-1} + \cdots + 2^{n-\ell w}e_1 + e_0$, where $\ell = \lceil n/w \rceil - 1$ as above. Compute and record the powers $(g^e)^{2^w}, (g^e)^{2^{2w}}, \ldots, (g^e)^{2^{\ell w}}$ by successive squarings. Then determine successively $e_0, e_1, e_2, \ldots, e_\ell$ from

$$\begin{aligned}
g^{2^{\ell w}e_0} &= (g^e)^{2^{\ell w}} \\
g^{2^{n-w}e_1} &= (g^e)^{2^{(\ell-1)w}} \; g^{-2^{(\ell-1)w}e_0} \\
g^{2^{n-w}e_2} &= (g^e)^{2^{(\ell-2)w}} \; g^{-2^{(\ell-2)w}e_0} \; g^{-2^{n-2w}e_1} \\
g^{2^{n-w}e_3} &= (g^e)^{2^{(\ell-3)w}} \; g^{-2^{(\ell-3)w}e_0} \; g^{-2^{n-3w}e_1} \; g^{-2^{n-2w}e_2} \\
&\;\vdots \\
g^{2^{n-w}e_\ell} &= (g^e) \qquad g^{-e_0} \qquad g^{-2^{n-\ell w}e_1} \; g^{-2^{n-(\ell-1)w}e_2} \cdots g^{-2^{n-2w}e_{\ell-1}}.
\end{aligned}$$

The right side of each equation is a product of a recorded power of g^e and some precomputed powers of g. I look up the left side in a small precomputed hash table to determine e_k. One can instead find e_k by binary search, if the relevant powers in Pippenger's table are sorted.

The number of multiplications in this accelerated algorithm is still quadratic in n, but it is inverse quadratic in w. Slight further improvements are possible, as in Brauer's algorithm and Pippenger's algorithm.

An example. The prime $q = 2^{224} - 2^{96} + 1$ has $q - 1 = 2^{96}(2^{128} - 1)$, so $m = 2^{128} - 1$ and $n = 96$. A Jacobi-symbol test reveals that $r = 11$ is not a square in $\mathbf{Z}/q$, so take $g = 11^{2^{128}-1}$.

Choose $w = 6$. Build a table of the powers $g^{-i}, g^{-2^6 i}, g^{-2^{12}i}, \ldots, g^{-2^{90}i}$ for all $i \in \{0, 1, \ldots, 63\}$.

Given a square u in $\mathbf{Z}/q$, compute $v = u^{(m-1)/2} = u^{2^{127}-1}$. This can be done with 126 squarings and just 10 more multiplications, thanks to the shape of m. The multiplications produce exponents $2^2 - 1, 2^3 - 1, 2^6 - 1, 2^{12} - 1, 2^{24} - 1, 2^{48} - 1, 2^{96} - 1, 2^{120} - 1, 2^{126} - 1, 2^{127} - 1$.

Compute uv and uv^2 with 2 more multiplications. Now $uv^2 = u^m$ is a power of g, say g^e, with $e = e_0 + 2^6 e_1 + 2^{12} e_2 + \cdots + 2^{90} e_{15}$. Compute $(g^e)^{2^6}, (g^e)^{2^{12}}, \ldots, (g^e)^{2^{90}}$ with 90 squarings.

The power $(g^e)^{2^{90}}$ is the same as $g^{2^{90}e_0}$. Look it up in a precomputed table of 64th roots of 1 to determine e_0.

Next compute $(g^e)^{2^{84}} g^{-2^{84}e_0}$. This is the same as $g^{2^{90}e_1}$. Look it up in the same table to determine e_1.

Subsequent steps involve $(g^e)^{2^{78}} g^{-2^{78}e_0} g^{-2^{84}e_1}$, $(g^e)^{2^{72}} g^{-2^{72}e_0} g^{-2^{78}e_1} g^{-2^{84}e_2}$, and so on through $(g^e) g^{-e_0} g^{-2^6 e_1} \ldots g^{-2^{84}e_{14}}$. Overall there are 120 multiplications by precomputed powers of g.

Write $e/2$ as $d_0 + 2^6 d_1 + \cdots + 2^{90} d_{15}$, compute $g^{-e/2} = g^{-d_0} g^{-2^6 d_1} \ldots g^{-2^{90} d_{15}}$, and multiply by uv to obtain a square root of u. There are 16 more multiplications here.

Total work: 364 multiplications, of which 216 are squarings. Increasing w to 8, and precomputing 3072 powers instead of 1024 powers, would reduce the cost to 304 multiplications, of which 214 are squarings.

In contrast, the Tonelli-Shanks algorithm needs about 2600 multiplications for an average u. The quadratic-extension methods need more than 700 multiplications, plus at least one division or Jacobi-symbol test.

Generalizations. Adleman, Manders, and Miller in [5] replaced 2^n with 3^n, 5^n, etc. to compute cube roots, fifth roots, etc.

Adleman, Manders, and Miller suggested computing a sixth root as a square root of a cube root. Lindhurst in [27, chapter 3] pointed out that it can be faster to compute a sixth root directly. Lindhurst's algorithm for 2^wth roots, like my algorithm, determines the exponent w bits at a time, although it still uses bit-by-bit exponentiation.

Pohlig and Hellman in [35], and independently Silver, replaced 2^n with any smooth number to compute discrete logarithms in arbitrary cyclic groups of smooth order. Precomputation is useful in this generality.

4. Square roots in prime fields

Assume that q is prime. How quickly can we compute square roots in the field $\mathbf{Z}/q$? This section analyzes the speed of the quadratic-extension methods presented in section 2 and the discrete-logarithm methods presented in section 3.

Representation. Elements of the field $\mathbf{Z}/q$ are conventionally represented as nonnegative integers smaller than q. Nonnegative integers are, in turn, represented as strings of bits in radix 2.

One can save time, as illustrated in [12], by allowing negative integers, integers slightly larger than q, "carry-save" bits, etc. The rest of this section also applies to these modified representations.

One can save much more time by using a discrete-logarithm representation of nonzero elements of $\mathbf{Z}/q$. But this representation is not useful for most applications: other common operations, such as addition, become extremely difficult.

Integer multiplication. See [11] for a survey of multiplication techniques, and [13] for a detailed analysis of the complexity of multiplication on current computers.

Given two integers a, b, each with $(1+o(1)) \lg q$ bits, one can compute ab in time $(3+o(1))M(q)$ under a realistic machine model:

- time $(1+o(1))M(q)$ transforming a,
- time $(1+o(1))M(q)$ transforming b,
- time $o(1)M(q)$ combining the transformed inputs, and
- time $(1+o(1))M(q)$ reversing the transformation.

Here $M(q) = \mu \lg q \lg \lg q \lg \lg \lg q$, where μ is a constant that depends on the model.

Squaring is faster: one can compute a^2 in time $(2+o(1))M(q)$, because there are only two transformations. Similar speedups apply to $a^2 + b^2$ and other quadratic forms.

One can compute $a + b$ in time $o(1)M(q)$. The same comment applies to other "easy" operations, such as subtraction and comparison. This time is asymptotically negligible in the context of this paper.

Integer division. Given an integer a with at most $(2 + o(1)) \lg q$ bits, one can compute $a \bmod q$ as follows: multiply the top half of a by an approximate reciprocal of q to obtain an approximate quotient b; compute $a - bq$; subtract a few multiples of q if necessary.

One can compute the approximate reciprocal of q in time $O(1)M(q)$. See, e.g., [11]. This time is asymptotically negligible in the context of this paper, because it is done only once. The computation of $a \bmod q$ then takes time $(6 + o(1))M(q)$.

Similarly, one can save the transform of q, and the transform of the approximate reciprocal of q. The computation of $a \bmod q$ then takes time $(4 + o(1))M(q)$.

One can further reduce the time by using half-sized transforms to compute $a-bq$. See [38, page 216]. The computation of $a \bmod q$ then takes time $(3 + o(1))M(q)$.

Reduction mod q takes less time for "sparse" primes q. For example, division by $2^{224} - 2^{96} + 1$ is easy. This makes a big difference in the total time for arithmetic mod q.

Field multiplication. Given two elements a, b of $\mathbf{Z}/q$, one can compute ab in time $(6+o(1))M(q)$: first $(3+o(1))M(q)$ for integer multiplication, then $(3+o(1))M(q)$ for reduction mod q, as discussed above.

The time drops to $(5+o(1))M(q)$ if $a = b$, or if the transform of b can be reused from a previous multiplication.

Field division. Schönhage's algorithm in [37] computes the continued fraction for a/q in time $O(1)M(q) \lg \lg q$.

Consequently one can compute reciprocals in $\mathbf{Z}/q$, and Jacobi symbols modulo q, in time $O(1)M(q) \lg \lg q$.

Squaring in quadratic extensions. The methods described in section 2 spend most of their time squaring elements of $(\mathbf{Z}/q)[x]/(x^2 - d)$.

The obvious representation of $f_0 + f_1 x$ in $(\mathbf{Z}/q)[x]/(x^2 - d)$ is as the vector (f_0, f_1). The square $(f_0 + f_1 x)^2$ is represented by $(f_0^2 + f_1^2 d, 2 f_0 f_1)$, which can be computed from (f_0, f_1) with four multiplications, of which two are squarings and one is multiplication by d. The total time is $(22 + o(1))M(q)$, if I've counted correctly.

One can rewrite $2 f_0 f_1$ as $(f_0 + f_1)^2 - f_0^2 - f_1^2$. More importantly, one can save the transforms of f_0, f_1, and d. One can also add the transforms of the integers representing f_0^2 and $f_1^2 d$, transform the result, and reduce mod q, rather than transforming and reducing each piece separately. The total time with these improvements is $(15 + o(1))M(q)$. I think one can do better by using a larger transform for $f_1^2 d$, but I haven't analyzed the details yet.

One can save more time if d has substantially fewer than $\lg q$ bits. This is unlikely in Legendre's method and Pocklington's method, if the input is random. However, in Cipolla's method, d is $r^2 - u$, and one has control over r. I suggest looking for r very close to the square root of u in $\mathbf{R}$, so that $r^2 - u$ has only about half as many bits as u. I conjecture that there are $\delta, \epsilon \in o(1)$ such that, out of the integers r with $\left|r^2 - u\right| \le q^{1/2+\delta}$, the fraction for which $r^2 - u$ is square is within ϵ of $1/2$, for all primes q and all integers u between 1 and $q - 1$.

Eric Bach has pointed out to me that one can instead represent $f_0 + f_1x$ as the vector $(f_0, f_1, f_1^2 d)$. Then $(f_0 + f_1x)^2$ is represented by $(f_0^2 + f_1^2 d, 2f_0f_1, 4f_0^2 f_1^2 d)$, which can be computed from $(f_0, f_1, f_1^2 d)$ with only three multiplications, of which one is a squaring. The total time here is $(16 + o(1))M(q)$, if I've counted correctly: in other words, slightly worse asymptotics than the obvious representation, despite the smaller number of multiplications. This representation might nevertheless be useful in practice.

Square roots via quadratic extensions. The main computation in Legendre's method is a $(q - 1)/2$ power in $(\mathbf{Z}/q)[x]/(x^2 - u)$. This takes $(4 + o(1)) \lg q$ or $(3 + o(1)) \lg q$ multiplications in $\mathbf{Z}/q$, depending on the choice of representation of $(\mathbf{Z}/q)[x]/(x^2 - u)$. The first representation takes total time $(15 + o(1))M(q) \lg q$.

Legendre's method also involves a reciprocal, $1/a_1$ in Theorem 2.1, and $2 + o(1)$ Jacobi-symbol computations on average. These computations take average time $O(1)M(q) \lg\lg q$, which is asymptotically negligible but quite noticeable for small values of q.

Pocklington's method involves essentially the same computations. For primes q where $\mathrm{ord}_2(q - 1)$ is large, Pocklington's method has the advantage of succeeding for almost all choices of r. If $\mathrm{ord}_2(q - 1) \geq \lg\lg q$, for example, then Pocklington's method succeeds with probability at least $1 - 2/\lg q$. One can and should skip the Jacobi-symbol tests in this case.

Cipolla's method has the advantage of faster computations if $r^2 - u$ is small, as discussed above. Cipolla's method also avoids divisions.

Square roots via discrete logarithms. Write $n = \mathrm{ord}_2(q-1)$ and $m = (q-1)/2^n$ as in section 3.

There are two main pieces in the new algorithm: an $(m - 1)/2$ power in F and approximately $n^2/2w^2$ more multiplications in F. The precomputation involves $2 + o(1)$ Jacobi-symbol computations on average, an m power in F, and about $2^w n/w$ more multiplications in F, to produce a table of about $2^w n/w$ elements of F.

There are several ways to see how much the new algorithm improves on the Tonelli-Shanks algorithm. For example, fix $\lambda \geq 0$, and consider $n \in (\lambda + o(1))\sqrt{\lg q}$. The Tonelli-Shanks algorithm uses $(1 + \lambda^2/4 + o(1)) \lg q$ multiplications on average, mostly squarings, for a total time of $(5 + 5\lambda^2/4 + o(1))M(q) \lg q$. The new algorithm, with $w \approx (1/3) \lg\lg q$, uses $(1 + o(1)) \lg q$ multiplications, mostly squarings, for a total time of $(5 + o(1))M(q) \lg q$.

One can instead consider the range of n for which square-root time is within a factor $1+o(1)$ of exponentiation time. The Tonelli-Shanks method uses $(1+o(1)) \lg q$ multiplications if $n \in o(1)\sqrt{\lg q}$. The new algorithm, again with $w \approx (1/3) \lg\lg q$, uses $(1 + o(1)) \lg q$ multiplications if $n \in o(1)\sqrt{\lg q} \lg\lg q$. The precomputation in both cases uses $(1 + o(1)) \lg q$ multiplications; the additional precomputation for the new algorithm is asymptotically less expensive than the precomputation of g.

One can also consider the range of n for which the discrete-logarithm methods, including precomputation time, are faster than the quadratic-extension methods. The Tonelli-Shanks algorithm, including precomputation, takes average time at most $(15 + o(1))M(q) \lg q$ if n is at most $(2 + o(1))\sqrt{\lg q}$. The new algorithm, including precomputation, takes average time at most $(15 + o(1))M(q) \lg q$ if n is at most $(\sqrt{1/2} + o(1))\sqrt{\lg q} \lg\lg q$; here $w \approx (1/2) \lg\lg q - \lg\lg\lg q$.

Finally, one can consider the range of n for which the discrete-logarithm methods, after a polynomial-time precomputation, are faster than the quadratic-extension methods. The Tonelli-Shanks algorithm takes average time $(15 + o(1))M(q) \lg q$ if $n \in (\sqrt{8} + o(1))\sqrt{\lg q}$. For any constant $\lambda > 0$, the new algorithm takes time $(15 + o(1)M(q) \lg q$ if $n \in (\lambda + o(1))\sqrt{\lg q} \lg\lg q$; here $w \approx (1/2)\lambda \lg\lg q$, and the exponent in the precomputation time grows with λ.

References

[1] —, *17th annual symposium on foundations of computer science*, IEEE Computer Society, Long Beach, California, 1976. MR 56 #1766.

[2] —, *18th annual symposium on foundations of computer science*, IEEE Computer Society, Long Beach, California, 1977. MR 57 #18173.

[3] —, *Proceedings of the 17th annual ACM symposium on theory of computing*, Association for Computing Machinery, New York, 1985.

[4] —, *Digital signature standard (DSS)*, Federal Information Processing Standard 186-2, National Institute of Standards and Technology, Washington, 2000. Available from `http://csrc.nist.gov/publications/fips/`.

[5] Leonard Adleman, Kenneth Manders, Gary Miller, *On taking roots in finite fields*, in [2] (1977), 175–178. MR 58 #19339.

[6] Eric Bach, *A note on square roots in finite fields*, IEEE Transactions on Information Theory **36** (1990), 1494–1498.

[7] Eric Bach, Jeffrey Shallit, *Algorithmic number theory, volume 1: efficient algorithms*, MIT Press, Cambridge, Massachusetts, 1996. ISBN 0–262–02405–5. Available from `http://www.math.uwaterloo.ca/~shallit/ant.html`.

[8] Elwyn R. Berlekamp, *Factoring polynomials over finite fields*, Bell System Technical Journal **46** (1967), 1853–1859. MR 36 #2314.

[9] Elwyn R. Berlekamp, *Algebraic coding theory*, McGraw-Hill, New York, 1968. MR 38 #6873.

[10] Elwyn R. Berlekamp, *Factoring polynomials over large finite fields*, Mathematics of Computation **24** (1970), 713–735. MR 43 #1948.

[11] Daniel J. Bernstein, *Multidigit multiplication for mathematicians*, to appear, Advances in Applied Mathematics.

[12] Daniel J. Bernstein, *Fast point multiplication on the NIST P-224 elliptic curve*, draft.

[13] Daniel J. Bernstein, *Faster multiplication of integers*, draft.

[14] Alfred Brauer, *On addition chains*, Bulletin of the American Mathematical Society **45** (1939), 736–739. MR 1,40a.

[15] Ernest F. Brickell, Daniel M. Gordon, Kevin S. McCurley, David B. Wilson, *Fast exponentiation with precomputation (extended abstract)*, in [36] (1993), 200–207; newer version in [16].

[16] Ernest F. Brickell, Daniel M. Gordon, Kevin S. McCurley, David B. Wilson, *Fast exponentiation with precomputation: algorithms and lower bounds* (1995); draft in [15]. Available from `http://research.microsoft.com/~dbwilson/bgmw/`.

[17] Michele Cipolla, *Un metodo per la risoluzione della congruenza di secondo grado*, Rendiconto dell'Accademia delle Scienze Fisiche e Matematiche Napoli **9** (1903), 154–163.

[18] Rajiv Gupta, Kenneth S. Williams (editors), *Number theory*, American Mathematical Society, Providence, 1999. ISBN 0–8218–0964–4. MR 99k:11005.

[19] Frederick Hoffman, Roy B. Levow, R. S. D. Thomas (editors), *Proceedings of the third Southeastern conference on combinatorics, graph theory, and computing*, Florida Atlantic University, Boca Raton, 1972. MR 48 #8247.

[20] Ming-Deh A. Huang, *Riemann hypothesis and finding roots over finite fields*, in [3] (1985), 121–130.

[21] Ming-Deh A. Huang, *Generalized Riemann hypothesis and factoring polynomials over finite fields*, Journal of Algorithms **12** (1991), 464–481. MR 92j:68057.

[22] Toshiya Itoh, *Efficient probabilistic algorithm for solving quadratic equations over finite fields*, Electronics Letters **23** (1987), 869–870.

[23] Toshiya Itoh, *An efficient probabilistic algorithm for solving quadratic equation over finite fields*, Electronics and Communications in Japan **72** (1989), 88–96. MR 91i:11190.

[24] Adrien-Marie Legendre, *Recherches d'analyse indéterminée*, Histoire de L'Académie Royale des Sciences (1785), 465–559.

[25] Derrick H. Lehmer, *Computer technology applied to the theory of numbers*, in [26] (1969), 117–151. MR 40 #84.

[26] William J. Leveque (editor), *Studies in number theory*, MAA Studies in Mathematics 6, Prentice-Hall, Englewood Cliffs, New Jersey, 1969. MR 39 #1388.

[27] Scott Lindhurst, *Computing roots in finite fields and groups, with a jaunt through sums of digits*, Ph.D. thesis, University of Wisconsin at Madison, 1997. Available from `http://members.aol.com/SokobanMac/scott/papers/papers.html`.

[28] Scott Lindhurst, *An analysis of Shanks's algorithm for computing square roots in finite fields*, in [18] (1999), 231–242. MR 2000b:11140. Available from `http://members.aol.com/SokobanMac/scott/papers/papers.html`.

[29] Maurice Mignotte, *Calcul des racines d-ièmes dans un corps fini*, Comptes Rendus des Séances de l'Académie des Sciences **290** (1980), A205–A206. MR 81b:12022.

[30] René C. Peralta, *A simple and fast probabilistic algorithm for computing square roots modulo a prime number*, IEEE Transactions on Information Theory **32** (1986), 846–847. MR 87m:11125.

[31] Nicholas Pippenger, *On the evaluation of powers and related problems (preliminary version)*, in [1] (1976), 258–263; newer version split into [32] and [33]. MR 58 #3682.

[32] Nicholas Pippenger, *The minimum number of edges in graphs with prescribed paths*, Mathematical Systems Theory **12** (1979), 325–346; draft in [31]. MR 81e:05079.

[33] Nicholas Pippenger, *On the evaluation of powers and monomials*, SIAM Journal on Computing **9** (1980), 230–250; draft in [31]. MR 82c:10064.

[34] Henry C. Pocklington, *The direct solution of the quadratic and cubic binomial congruences with prime moduli*, Proceedings of the Cambridge Philosophical Society **19** (1917), 57–59.

[35] Stephen C. Pohlig, Martin E. Hellman, *An improved algorithm for computing logarithms over GF(p) and its cryptographic significance*, IEEE Transactions on Information Theory **24** (1978), 106–110. MR 58 #4617.

[36] Rainer A. Rueppel (editor), *Advances in cryptology: EUROCRYPT '92*, Lecture Notes in Computer Science 658, Springer, Berlin, 1993. ISBN 3–540–56413–6. MR 94e:94002.

[37] Arnold Schönhage, *Schnelle Berechnung von Kettenbruchentwicklugen*, Acta Informatica **1** (1971), 139–144.

[38] Arnold Schönhage, Andreas F. W. Grotefeld, Ekkehart Vetter, *Fast algorithms: a multitape Turing machine implementation*, Bibliographisches Institut, Mannheim, 1994. ISBN 3–411–16891–9. MR 96c:68043.

[39] Daniel Shanks, *Five number-theoretic algorithms*, in [40] (1973), 51–70. MR 51 #8072.

[40] R. S. D. Thomas, Hugh C. Williams (editors), *Proceedings of the second Manitoba conference on numerical mathematics*, Congressus Numerantium 7, Utilitas Mathematica, Winnipeg, Manitoba, 1973. MR 50 #3517.

[41] Alberto Tonelli, *Bemerkung über die Auflösung quadratischer Congruenzen*, Göttinger Nachrichten (1891), 344–346.

[42] Hugh C. Williams, *Some algorithms for solving* $x^q \equiv N \pmod p$, in [19] (1972), 451–462. MR 51 #320.

[43] Kenneth S. Williams, Kenneth Hardy, *A refinement of H. C. Williams' qth root algorithm*, Mathematics of Computation **61** (1993), 475–483. MR 93k:11003.

[44] Andrew C. Yao, *On the evaluation of powers*, SIAM Journal on Computing **5** (1976), 100–103. MR 52 #16128.

Department of Mathematics, Statistics, and Computer Science (M/C 249), The University of Illinois at Chicago, Chicago, IL 60607–7045

E-mail address: `djb@cr.yp.to`