A Note on Square Roots in Finite Fields

Eric Bach

Computer Sciences Technical Report #795

October 1988

A Note on Square Roots in Finite Fields

Eric Bach

Computer Sciences Department
University of Wisconsin–Madison
Madison, WI 53706

October 4, 1988

Abstract. A simple method is presented whereby the quadratic character in a finite field of odd order q can be computed in $O(\log q)^2$ steps. It is also shown how sequences generated deterministically from a random seed can be used reliably in a recent randomized algorithm of Peralta for computing square roots in finite fields.

Submitted to IEEE Transactions on Information Theory.

This research was sponsored by the National Science Foundation.

1. Introduction.

In [10], Peralta has published two interesting randomized algorithms for computing square roots modulo an odd prime p. In fact, his algorithms work in any finite field of odd characteristic, and in connection with this I would like to discuss two questions:

1. What is the best way to decide if an element in a finite field is a square?
2. How should one choose successive pseudo-random numbers for use in these algorithms?

The first question arises because several randomized algorithms for computing square roots in a finite field of order q fail when a randomly constructed field element is a square. If failure can be predicted in less than $O(\log q)^3$ steps – the time needed to run the algorithm – then another trial can be started with very little loss of time. One can test if a nonzero element t is a square by computing $t^{(q-1)/2}$, but this requires an $O(\log q)^3$ computation. Section 2 gives another test that requires $O(\log q)^2$ steps, which generalizes the Jacobi symbol algorithm to any finite field. The algorithm is implicit in the reciprocity law for polynomials over finite fields [4], but it does not seem to have been analyzed before.

The second question has already been studied in connection with some algorithms for computing square roots mod p. In [1] I showed that these algorithms are very reliable when run on sequences derived in a simple fashion from a randomly chosen seed. The results therein do not apply to Peralta's second algorithm, nor do they apply to non-prime finite fields. It therefore seemed interesting to ask if these results could be extended to Peralta's algorithms. Section 3 reviews the algorithms, and section 4 answers this question in the affirmative.

2. A quadratic character algorithm.

In the sequel p denotes an odd prime, and $q = p^n$, a prime power. $\mathbb{F}_q$ or K denotes a finite field of q elements, K^* its group of nonzero elements, $\bar{K}$ its algebraic closure, and $K[X]$ the polynomial ring in one indeterminate. I assume that $\mathbb{F}_q$ is implemented as $\mathbb{F}_p(\alpha)$, where α satisfies a monic polynomial equation of degree n over $\mathbb{F}_p$. Then in $\mathbb{F}_q$, addition and subtraction take $O(\log q)$ steps, and multiplication and division take $O(\log q)^2$ steps (assuming that classical algorithms are used for integer and polynomial arithmetic). χ denotes the usual quadratic character on K; the following definition generalizes the Jacobi symbol to polynomials over finite fields.

Definition 2.1. Let K be a finite field of odd characteristic, and let f and g be polynomials in $K[X]$, with g monic and of positive degree. If g is irreducible, then

$$\left(\frac{f}{g}\right) = \begin{cases} 1, & \text{if } f \text{ and } g \text{ are relatively prime and } f \text{ is a square mod } g; \\ -1, & \text{if } f \text{ and } g \text{ are relatively prime and } f \text{ is not a square mod } g; \\ 0, & \text{otherwise.} \end{cases}$$

If $g = g_1 \cdots g_r$ with each g_i monic and irreducible, then $(\frac{f}{g}) = \prod_{i=1}^{r} (\frac{f}{g_i})$.

The above definition implies that if $\alpha \in K$, $(\frac{\alpha}{g}) = \chi(\alpha)^{\deg g}$. The analog of quadratic reciprocity is the following.

Theorem 2.2 [4]. Let $f, g \in K[X]$, with f, g monic and $\deg f, \deg g > 0$. Then

$$(\frac{f}{g}) = (-1)^{\deg f \deg g \frac{(|K|-1)}{2}} (\frac{g}{f}).$$

Algorithm 2.3. Input: $f, g \in K[X]$, with g monic and $\deg g > 0$; output: $(\frac{f}{g})$.

$f \leftarrow f \bmod g$.
$\alpha \leftarrow$ leading coefficient of f.
If $\deg f = 0$ then return $\chi(\alpha)^{\deg g}$.
$f \leftarrow f/\alpha$.
Return $\pm\chi(\alpha)^{\deg g}(\frac{g}{f})$ (taking '$-$' iff $(q-1)/2$, $\deg f$, and $\deg g$ are all odd).

Theorem 2.4. If f and g in $K[X]$ have degree at most d, then algorithm 2.3 computes $(\frac{f}{g})$ using $O(d^2)$ field operations and $O(d)$ evaluations of the quadratic character in K. Consequently, the quadratic character in $\mathbb{F}_q$ can be evaluated with $O(\log q)^2$ bit operations.

Proof. The correctness of the algorithm follows from the reciprocity law. To analyze its running time, note that it computes a polynomial remainder sequence

$$u_0 = q_0 u_1 + \alpha_0 u_2$$

$$u_1 = q_1 u_2 + \alpha_1 u_3$$

$$\cdots$$

$$u_{k-1} = q_{k-1} u_k + \alpha_{k-1}$$

where $u_0 = f$, $u_1 = g$, u_i is monic for $i \geq 1$, and $\deg u_i > \deg u_{i+1}$ for $i = 1, \ldots, k-1$. Such remainder sequences have length $O(d)$ and can be computed with $O(d^2)$ field operations [9]; this proves the first assertion.

To prove the second, note that if $\mathbb{F}_q = \mathbb{F}_p(\alpha)$ where α has a monic defining polynomial g, then $\mathbb{F}_q \cong \mathbb{F}_p[X]/g(X)$, so $\chi(f(\alpha)) = (\frac{f}{g})$. In $\mathbb{F}_p$, χ can be evaluated in $O(\log p)^2$ steps by the Jacobi symbol algorithm [5], and arithmetic operations take $O(\log p)^2$ steps. So if $q = p^n$, the total number of bit operations used is at most a constant times

$$n^2(\log p)^2 + n(\log p)^2 = O(n \log p)^2 = O(\log q)^2. \qquad \blacksquare$$

Remarks.

1. The subresultant algorithm can be used to compute a remainder sequence that differs from $u_0, \ldots, u_k$ only by constant factors [9], from which it is easy to recover the u_i's and the α_i's. Then

$$(\frac{f}{g}) = \chi\Big(\prod_{\substack{0 \le i < k \\ \deg u_i + 1 \text{ odd}}} \alpha_i\Big)(-1)^{\frac{q-1}{2}\sum_{i=1}^{k-1} \deg u_i \deg u_{i+1}},$$

from which the quadratic character on $\mathbb{F}_q$ can be quickly reduced in parallel to arithmetic in $\mathbb{F}_p$ and one evaluation of χ. This gives another proof of Fich and Tompa's result [6] that for fields of small characteristic, the quadratic character is in NC.

2. Even for prime fields, it seems to be unknown if one can decide if an element of $\mathbb{F}_q$ is a d-th power in $O(\log q)^2$ steps, when $d \neq 2$, and $d \mid q-1$.

3. The question of this section becomes moot if K has characteristic 2, for then every element is a square.

3. Two Square Root Algorithms.

Below, I review Peralta's results, generalized to any finite field.

Algorithm 3.1. Input: a, a nonzero square in K; output: a square root of a.

Choose $x \in K$ at random.
If $x^2 = a$, output x.
Otherwise, in $A = K[T]/(T^2 - a)$, compute $(T + x)^{(q-1)/2} = uT + v$.
If $v = 0$, output u^{-1}.

Theorem 3.2 [10]. Algorithm 3.1 returns a square root of a unless $\chi(x^2 - a) = 1$. It takes $O(\log q)^3$ steps, and fails with probability $1/2 - 3/(2q)$.

Algorithm 3.3. Input: a, a nonzero square in K; output: a square root of a.

Let $q - 1 = 2^e d$, where d is odd.
Choose $x \in K$ at random.
If $x = 0$ or $x^2 = -a$, fail.
Otherwise, in $A = K[T]/(T^2 + a)$, compute $(T + x)^d = uT + v$.
If $uv = 0$, fail.
Otherwise, find the least i such that $(uT + v)^{2^i} = (rT + s)^2 = wT$.
Return s/r.

Theorem 3.4 [10]. Let b denote a square root of $-a$, and let $m = 2^{e-1}$; assume $q \equiv 1 \pmod 4$. Algorithm 3.3 returns a square root of a unless $x = \pm b$ or for some nonzero y, $(x+b)/(x-b) = y^m$. It takes $O(\log q)^3$ steps, and fails with probability $1/m + 1/q$.

Ordinarily one simply repeatedly tries a randomized algorithm until it works. However, by the results of section 2, it is faster to test random values of x until one is found with $\chi(x^2 - a) \neq 1$ than to repeatedly run algorithm 3.1 (this strategy was suggested by Berlekamp [2]).

If m is large, algorithm 3.3 is much more reliable than algorithm 3.1. Unfortunately, there seems to be no better way to tell if a choice of x will work than to try it. However, any x for which $\chi(x^2 + a) = -1$ will succeed in algorithm 3.3, for then $(x + b)/(x - b)$ is not a square, and surely not an m-th power. One might also use algorithm 2.3 here to quickly find such an x.

Remarks.

1. Peralta's first method is a simplified version of the Berlekamp-Rabin algorithm for factoring $T^2 - a$. For, if $(T + x)^{(q-1)/2} \equiv uT \bmod T^2 - a$, then Rabin's algorithm [11] would compute $\gcd((T+x)^{(q-1)/2} - 1, T^2 - a) = T - u^{-1}$. Since $K[T] \cong K[T+x]$, replacing $T+x$ by T shows that $\gcd(T^{(q-1)/2} - 1, (T - x)^2 - a) = T - (x + u^{-1})$, from which Berlekamp's algorithm [2] would return u^{-1} as a square root of a.
2. The failure criterion for algorithm 3.1 is nearly identical to that of the Cippola-Lehmer algorithm [8], which computes $\sqrt{a}$ in $O(\log q)^3$ steps when a random $x \in K$ satisfies $\chi(x^2 - 4a) \neq 1$. Also, given $x \in K$ for which $\chi(x) = -1$, the Tonelli-Shanks algorithm [13] will compute a square root in K in $O(\log q)^4$ steps. Both of these methods could profit from algorithm 2.3.
3. All algorithms discussed in this section for computing square roots in K take at least $O(\log q)^3$ steps. It is unknown if this can be reduced to $O(\log q)^2$, although when K has characteristic 2, one can compute square roots by inverting the matrix for the Frobenius map $x \to x^2$ [3]. This method takes $O(\log q)^2$ steps once the inverse matrix is computed.

4. Deterministic Sequences for Square Root Computation.

The algorithms of the last section will in general require a sequence of random inputs from K; this section discusses simple methods to generate them from a random seed $x \in K$. The results show that if fixed constants $c_1, \ldots, c_k$ are properly chosen, then trials using $x + c_1, \ldots, x + c_k$ will simulate independence.

Definition 4.1. A *line* in $\mathbb{F}_q$ is a set of the form $\{\gamma + \delta t : t \in \mathbb{F}_p\}$, where $\gamma, \delta \in \mathbb{F}_q$ and $\delta \neq 0$.

In the results below I will assume that $c_1, \ldots, c_k$ are distinct elements of K, chosen from a set containing no lines (this is the interpretation of "properly chosen"). Such sets are easy to find. For example (viewing $\mathbb{F}_q$ as a vector space over $\mathbb{F}_p$), $\{(x_1, ..., x_n) : \text{all } x_i \neq 0\}$ contains no lines. Also, any set of size less than p cannot contain a line.

Lemma 4.2. Let $m \mid q-1$, with $m > 1$, and let $0 < e_i < m$ for $i = 1, \ldots, l$. Then

$$f(X) = \prod_{i=1}^{l}((X + c_i + b)(X + c_i - b)^{m-1})^{e_i}$$

is not an m-th power in $\bar{K}[X]$.

Proof. The zeroes of f are $\alpha_i = b - c_i$ and $\beta_i = -b - c_i$. Certainly the α_i's are distinct, as are the β_i's. Hence no three of them can be equal. Localizing at $X - \alpha_i$ shows that if f is an m-th power, then $\{\alpha_1, \ldots, \alpha_l\} = \{\beta_1, \ldots, \beta_l\}$, that is, $\{c_1 - b, \ldots, c_l - b\} = \{c_1 + b, \ldots, c_l + b\}$. Thus $C = \{c_1, \ldots, c_l\}$ is invariant under translation by $2b$, so C must be a disjoint union of sets of the form $\{\gamma + 2b \cdot t : t \in \mathbb{F}_p\}$. This contradicts the hypothesis that C contains no lines. ■

Lemma 4.3. Let $m \mid q-1$, with $m > 1$, and let $b \neq 0$. If N denotes the number of $(x, y_1, \ldots, y_k)$ in K^{m+1} satisfying

$$(x + c_i + b)(x + c_i - b)^{m-1} = y_i^m, \quad i = 1, \ldots k,$$

then $N \leq q + 2km^k\sqrt{q}$.

Proof. Let χ denote a character of order m on K^*. Weil showed that if $f \in K[X]$, with d distinct roots in $\bar{K}$, but not an m-th power, then

$$\left| \sum_{x \in K} \chi(f(x)) \right| \leq (d-1)\sqrt{q}.$$

[12, Theorem 2C]. By the orthogonality relation for characters,

$$N = \sum_{x \in K} \prod_{i=1}^{k} \sum_{0 \leq e_i < m} \chi((x + c_i + b)(x + c_i - b)^{m-1})^{e_i}$$

$$= \sum_{0 \leq e_1, \ldots, e_k < m} \sum_{x \in K} \chi((x + c_1 + b)(x + c_1 - b)^{m-1})^{e_1} \cdots \chi((x + c_k + b)(x + c_k - b)^{m-1})^{e_k}.$$

Group the terms in the first sum according to how many e_i's are nonzero and use Weil's estimate and lemma 4.2 to show that N is at most

$$q + \sum_{i=1}^{k}(2i-1)(m-1)^k \binom{k}{i}\sqrt{q} \quad = \quad q + (m^{k-1}[2k(m-1) - m] + 1)\sqrt{q} \quad \leq \quad q + 2km^k\sqrt{q}. \ ■$$

Theorem 4.4. Choose $x \in K$ at random. The probability that algorithm 3.1 fails on $x + c_i$, $i = 1, \ldots, k$, is at most $1/2^k + 2k/\sqrt{q}$. If $k = \lceil \frac{1}{2} \log_2 q \rceil$, this is $O(\log q/\sqrt{q})$.

Proof. If all trials fail, there are nonzero $y_1, ..., y_k \in K$ for which $(x + c_i)^2 - a^2 = y_i^2$, $i = 1, \ldots, k$. Take $m = 2$ and $b = \sqrt{a}$ in lemma 4.3, and divide by 2^k to get the number of x's and by q to get the probability of failure. The second result follows by substitution. ■

Theorem 4.5. Choose $x \in K$ at random. The probability that algorithm 3.3 fails on $x + c_i$, $i = 1, \ldots, k$, is at most $1/m^k + 2k/\sqrt{q} + 2k/q$. If $k = \lceil \frac{1}{2} \log_m q \rceil$, this is $O(\log q/(\log m \sqrt{q}))$.

Proof. Choose b with $b^2 = -a$. If all trials fail either $x = -c_i \pm b$ for some i, or there are nonzero $y_1, \ldots, y_k \in K$ such that $(x + c_i + b)/(x + c_i - b) = y_i^m$, $i = 1, \ldots, k$. The rest of the proof is similar to that of theorem 4.4. ■

Remarks.

1. By lemma 4.2, the equations $(x + c_i + b)/(x + c_i - b) = y_i^m$, $i = 1, \ldots, k$, define an algebraic curve, whose genus g can be shown to be $k(m-1)m^{k-1} - m^k + 1$ by the Hurwitz formula [7]. Lemma 4.3 can be replaced by the Weil bound $N \leq q + 2g\sqrt{q}$, which will sharpen the above theorems. The improvement is very slight unless $m = 2$, in which case the dominant term in theorem 4.4 can be reduced by a factor of 2.
2. If $m \geq \sqrt{q}$, then theorem 4.5 becomes irrelevant as theorem 3.4 is sharper.
3. Theorem 4.4 evidently applies to the Cippola-Lehmer algorithm. If one merely wants to find an element in K that is not an m-th power (i.e. for use in the Tonelli-Shanks algorithm), then a bound similar to theorem 4.5 holds, provided only that $c_1, \ldots, c_k$ are distinct [14].

Acknowledgement. Thanks to Gilles Brassard for posing the question. Thanks also to Victor Shoup for suggesting the use of character sums.

References.

1. E. Bach, Realistic analysis of some randomized algorithms, Proceedings of the 19th ACM Symposium on Theory of Computing (1987).
2. E.R. Berlekamp, Factoring Polynomials over large finite fields, Mathematics of Computation 24, pp. 713-735 (1970).
3. E.R. Berlekamp, H. Rumsey, and G. Solomon, On the solution of algebraic equations over finite fields, Information and Control 10, pp. 553-564 (1967).
4. L. Carlitz, The arithmetic of polynomials in a Galois field, American Journal of Mathematics 54, pp. 39-50 (1932).
5. G. Collins and R. Loos, The Jacobi symbol algorithm, SIGSAM Bulletin 16:1, pp. 12-16 (1982).

6. F.E. Fich and M. Tompa, The parallel complexity of exponentiating polynomials over finite fields, Journal of the ACM (1988).
7. R. Hartshorne, Algebraic Geometry, Springer (1977).
8. D.H. Lehmer, Computer technology applied to the theory of numbers, in Studies in Number Theory (ed. W.J. LeVeque), MAA Studies in Mathematics 6 (1969).
9. R. Loos, Generalized polynomial remainder sequences, in Computer Algebra: Symbolic and Algebraic Computation (second edition), edited by B. Buchberger, G.E. Collins, and R. Loos, Springer-Verlag (1983).
10. R. Peralta, A simple and fast probabilistic algorithm for computing square roots modulo a prime number, IEEE Transactions on Information Theory IT-32, No. 6, pp. 846-847 (1986).
11. M.O. Rabin, Probabilistic algorithms in finite fields, SIAM Journal on Computing 9, pp. 273-280.
12. W.M. Schmidt, Equations over Finite Fields: An Elementary Approach, Springer (1976).
13. D. Shanks, Five number-theoretic algorithms, Proceedings of the Second Manitoba Conference on Numerical Mathematics, pp. 51-70 (1972).
14. V. Shoup, New algorithms for finding irreducible polynomials over finite fields, Proceedings of the 29th IEEE Symposium on Foundations of Computer Science (1988).