HIROSHIMA MATH. J.
34 (2004), 201–210

Practical fast algorithm for finite field arithmetics using group rings

Makoto MATSUMOTO and Shigehiro TAGAMI

(Received October 27, 2003)
(Revised January 16, 2004)

ABSTRACT. This paper studies a fast algorithm for finite field arithmetics, by representing a finite field as a residue of a group ring of a finite cyclic group, where the frobenius (q-th power) operation is efficiently computable. When the characteristic of the field is greater than 2, our algorithm is often much faster than a standard method (NTL) in computing inverse and power. For example, ours is roughly 23.6 times faster in computing power in $\mathbf{F}_{8191^{136}}$ than NTL. The implementation contains a new scheme for computing powers, which is applicable for any group if the q-th power operation is negligibly fast.

1. Introduction

Let p be a prime number, q a power of p, and $\mathbf{F}_q$ the finite field of order q. Our aim is a *practical* fast algorithm for arithmetic operations such as addition, multiplication, power, and inversion in $\mathbf{F}_{q^d}$, assuming that we have those for $\mathbf{F}_q$.

A most naive method to represent $\mathbf{F}_{q^d}$ is to use a polynomial basis. Choose a generator $\alpha \in \mathbf{F}_{q^d}$ as a ring over $\mathbf{F}_q$, then

$$\mathbf{F}_{q^d} = \mathbf{F}_q[\alpha] \cong \mathbf{F}_q[t]/\varphi_\alpha(t)$$

for the minimal polynomial $\varphi_\alpha(t)$ of α over $\mathbf{F}_q$. Thus, addition and multiplication are done in the polynomial ring $\mathbf{F}_q[t]$ and then the residue modulo $\varphi_\alpha(t)$ is taken. The inverse is given by Euclidean algorithm.

There is a more sophisticated method called a normal basis, which is a linear basis of $\mathbf{F}_{q^d}$ over $\mathbf{F}_q$ in the form of $\{\alpha^{q^j} \mid 0 \le j \le d-1\}$. An advantage is that the q-th power operation (often called the *frobenius* operation)

$$F_q : \mathbf{F}_{q^d} \to \mathbf{F}_{q^d}, \qquad x \mapsto x^q$$

can be computed by a cyclic shift of the coefficients with respect to the normal basis, which enables efficient exponentiation and inversion (for such basics on finite field algorithms, refer to [8]).

Date: Oct 27, 2003.
1991 *Mathematics Subject Classification*. Primary 11T71, 11T21; Secondary 94A60.
Key words and phrases. Finite field, algorithm.

Sophisticated algorithms attain low order of computational complexity using a normal basis, see for example [5]. However, when implemented, they are often slower than the naive polynomial basis method, because the sophisticated algorithms are complicated and require (practically too) large d to be advantageous over the naive method.

As evidence, the naive polynomial basis method is adopted in NTL (Number Theory Library), which is one of the fastest libraries at present for algebraic operations including finite field arithmetics. NTL is designed through the actual time comparisons between many sophisticated and naive methods, and adopts the naive methods with deliberate optimizations. It is freely available to the scientific community from http://www.shoup.net/ntl.

S. Gao et al. [5] introduced a new method, which has both the simplicity of polynomial basis and the efficiency of the frobenius operation in normal basis. The idea is to represent $\mathbf{F}_{q^d}$ as a subring of a group ring, where arithmetics are more efficient. The purpose of this paper is to realize this idea in a software, and compare it with NTL. We introduce a data structure which enables an efficient implementation of the frobenius operation. We develop algorithms for power and inversion, using the quick frobenius operation effectively. As a result, our method is much faster than NTL, if the characteristic of the field is greater than 2.

2. Cyclic group rings

The following is a modified version of [5], where $\mathbf{F}_{q^d}$ is constructed as a residue of a group ring. Take a generator $\alpha \in \mathbf{F}_{q^d}$ as a ring over $\mathbf{F}_q$, and let m be the multiplicative order of α (hence m is coprime to p). Then we have a surjective ring homomorphism

$$\mathrm{Pr} : \mathbf{F}_q[t]/(t^m - 1) \to \mathbf{F}_{q^d}, \qquad t \mapsto \alpha. \tag{1}$$

The left hand side is a group ring over the cyclic group, which we shall call a *cyclic group ring*, or CGR. (The term *cyclotomic ring* used in [10] has a different meaning in number theory.) Using its polynomial basis t^i $(i = 0, 1, \ldots, m-1)$, the q-th power frobenius operation is computed by

$$F_q : \sum_{i=0}^{m-1} a_i t^i \mapsto \left(\sum_{i=0}^{m-1} a_i t^i \right)^q = \sum_{i=0}^{m-1} a_i t^{[iq \pmod m]} = \sum_{i=0}^{m-1} a_{[iq^{-1} \pmod m]} t^i,$$

where q^{-1} is taken modulo m. Thus, q-th power operation can be realized by a permutation of the coefficients. We represent elements of $\mathbf{F}_{q^d}$ by any of its inverse images in the cyclic group ring. Addition, multiplication and power operations can be computed in the cyclic group ring. The inverse can be com-

puted as a special case of power, using [7] (see Theorem 3.3). To check the equality of two elements in $\mathbf{F}_{q^d}$, we do not need to compute the projection Pr. Let $\psi(t) := (t^m - 1)/\varphi_\alpha(t)$. Since $t^m - 1$ has no multiple factor, φ_α and ψ are coprime. Thus, two elements $x, y \in \mathbf{F}_q[t]/(t^m - 1)$ have the same image in $\mathbf{F}_{q^d}$ if and only if $\psi(t)(x - y) = 0$ in $\mathbf{F}_q[t]/(t^m - 1)$. In the implementations in this paper, we treat only the cases where $m = d + 1$ and consequently $\psi(t) = t - 1$.

To find such a pair (m, d), the following lemma is useful.

LEMMA 2.1. *Let q be a power of a prime p, d a positive integer, and m be a positive integer coprime to q. If d equals the order of q in the multiplicative group $(\mathbf{Z}/m)^\times$ of $\mathbf{Z}/m$, then there exists $\alpha \in \mathbf{F}_{q^d}$ such that* (1) *is surjective.*

Conversely, if m is the minimum positive integer such that for some $\alpha \in \mathbf{F}_{q^d}$ (1) *is surjective, then d is the order of q in $(\mathbf{Z}/m)^\times$.*

PROOF. Let β be a primitive m-th root of unity in the algebraic closure of $\mathbf{F}_q$. The condition is equivalent to that d is the degree of the minimum polynomial of β, which is the order of the action of the frobenius operation on β by Galois theory, in other words, the order of q in $(\mathbf{Z}/m)^\times$. □

The computational complexity in the cyclic group ring is a function of m, and thus m far larger than d is not practical. For example, if $(q^d - 1)/(q - 1)$ is a prime, m must coincide with it and is practically too large. In the other extreme, if m is a prime and if q is a generator of $(\mathbf{Z}/m)^\times$, then $d = m - 1$ and m is optimally small.

REMARK 2.2. *The probability that q generates $(\mathbf{Z}/m)^\times$ is explicitly known under the generalized Riemann hypothesis, if m is a prime* [3]. *For example, assume $q = p$, $p \not\equiv 1 \mod 4$ and let $M_q(x)$ denote the number of primes m not exceeding x for which q is a generator of $(\mathbf{Z}/m)^\times$. Then we have*

$$M_q(x) = C\frac{x}{\log_e x} + O\left(\frac{x \log_e \log_e x}{\log_e^2 x}\right),$$

where $C = 0.39\cdots = \prod_{p:\text{prime}}\left(1 - \frac{1}{p(p-1)}\right)$. For general q, a similar but a little complicated asymptotic formula is given there. These show that the optimal case $d = m - 1$ occurs rather often.

REMARK 2.3. *When $\mathbf{F}_{q^{d'}}$ requires a large m to be represented using* (1), *it is often the case that some of its finite extensions $\mathbf{F}_{q^d}$ can be realized for a much smaller m. Then we may compute $\mathbf{F}_{q^{d'}}$ inside $\mathbf{F}_{q^d}$. See the table of such d', d in* [6], *which treats a hardware implementation for the case $p = q = 2$. For example, if $q = 2$ and $d' = 127$, then $2^{127} - 1$ is a Mersenne prime and hence this is the required m, but for $d = 508 = 4 \cdot 127$, the required m is* 509.

3. Software implementation

There are a number of studies on implementation of arithmetics using CGR (cyclic group ring, often called *redundant representations*), e.g. [4] [5] [2] [6] [10]. They treat mostly only the case of $p = q = 2$ and hardware implementations, and pay little attention to software implementations.

One purpose of this paper is to give a software implementation of CGR and to compare its speed with NTL. Unexpectedly, it turns out that in the case of $p = q = 2$, CGR is much slower than NTL, whereas for $p = q > 2$, CGR is faster.

A problem in a software implementation lies in the frobenius operation. It involves a permutation of coefficients, whose time-complexity is $O(1)$ in hardware but is $O(m)$ in a naive software implementation. To solve this, we shall introduce a data structure which enables a frobenius operation just by one time multiplication in $\mathbf{Z}/m$.

An element x of a CGR

$$\mathbf{F}_q[t]/(t^m - 1)$$

is realized as an array of elements of $\mathbf{F}_q$ of length m, with two integer variables named skip s and origin i with values in $\mathbf{Z}/m$. The skip s should be coprime with m.

Assume that the array consists of $a[0], a[1], \ldots, a[m-1]$, the skip is s, and the origin is i. Then the tuple (a, s, i) represents the polynomial

$$\sum_{j=0}^{m-1} a[i + sj \pmod m] t^j \in \mathbf{F}_q[t]/(t^m - 1).$$

In this representation, multiplication by t is computed as the change of the origin

$$i \leftarrow i - s \pmod m,$$

and thus multiplication of two arbitrary elements

$$f(t)\left(\sum_{j=0}^{m-1} a[i + sj \pmod m] t^j\right)$$

can be done as usual, by adding $a[i + sj \pmod m](f(t) \cdot t^j)$ for $j = 0, \ldots, m-1$.

Frobenius operation F_q is realized as the change of the skip

$$s \leftarrow s \times q^{-1} \pmod m.$$

We may precompute $q^{-1} \in \mathbf{Z}/m$ for efficiency.

REMARK 3.1.

1. *We may dispense with the origin, if we do not need to make multiplication by t^j fast.*
2. *Since a frobenius operation is much faster than a multiplication in this representation, from now on, we shall neglect the time consumed in frobenius operations in the evaluation of the time complexity.*

POWER. Using the q-th power frobenius operation, the following q-ary method [9, P. 464] gives an algorithm for n-th power with $\log_q n$ times multiplications, with precomputation of a table of size q.

We assume that n is q-adically expanded:

$$n = \sum_{j=0}^{k} c_j q^j \qquad (c_j \in \{0, 1, \ldots, q-1\}, c_k \neq 0). \tag{2}$$

Let y be a variable, set to x^{c_k}. Then, for $j = k-1$ to 0, we iterate the following:

$$y \leftarrow y^q; \qquad y \leftarrow y x^{c_j}.$$

If we precompute x^c for all $0 < c \leq q-1$, this gives an algorithm with $\lfloor \log_q(n-1) \rfloor$ times multiplications. An experiment with $q = 8191$ shows that keeping such a large table is time-consuming. If we precompute x^{2^s} for $s = 1, 2, \ldots, \lfloor \log_2(q-1) \rfloor$, and use them to compute x^{c_j}, then the required number of multiplications is in average

$$\lfloor \log_q(n-1) \rfloor \times \lfloor \log_2(q-1) \rfloor / 2 \sim 0.5 \log_2 n$$

and the size of the precomputation table is $\lfloor \log_2(q-1) \rfloor$.

An improvement is as follows. Let b be a fixed positive integer. We precompute $x, x^2, \ldots, x^{2^b-1}$. Put

$$h := \lfloor \log_2(q-1) \rfloor + 1, \qquad h_b := \lceil h/b \rceil.$$

Let $y \in \mathbf{F}_{q^d}$ be a variable, first set to 1. Let us consider the 2-adic expansion of each c_j $(j = 0, 1, \ldots, k)$ appearing in (2). Let $0 \leq d_j \leq 2^b - 1$ be the integer represented by the consecutive b bits of c_j starting from the (bh_b)-th bit and ending at the $(bh_b - b + 1)$-st bit (a sequence of 0 is supplemented at the left in the 2-adic expansion of c_j). We execute

$$\text{if } d_j \neq 0 \text{ then } y \leftarrow y \times (x^{d_j})^{q^j}, \text{ for } j = 0, 1, \ldots, k-1. \tag{3}$$

This is computable using k times multiplications, since x^{d_j} is pre-computed.

Then, execute

$$y \leftarrow y^{2^b},$$

redefine d_j $(j = 0, \ldots, k)$ to be the integer represented by the consecutive b bits of c_j starting from the $b(h_b - 1)$-th bit and ending at the $(b(h_b - 1) - b + 1)$-st bit, and execute (3) again.

Iterate this process with d_j being the integer given by the bs-th bit to the $(bs - b + 1)$-st bit of c_j $(j = 0, \ldots, k)$ for $s = h_b, h_b - 1, \ldots, 1$, then we obtain $y = x^n$. (The above describes the first two steps: $s = h_b, h_b - 1$.)

THEOREM 3.2. *The above algorithm computes the power x^n using*

$$((1 - 2^{-b})\lfloor \log_q n \rfloor + b)h_b \sim \frac{1 - 2^{-b}}{b} \log_2 n + \log_2 q$$

times multiplications in average, using a precomputation table of size $2^b - 1$.

In this theorem, we neglect the number of frobenius operations, and we need $2^b - 1$ times multiplications for the precomputation table.

PROOF. The coefficient $1 - 2^{-b}$ is the probability that $d_j \neq 0$. To compute (3) we use $(1 - 2^{-b})\lfloor \log_q n \rfloor$ times multiplications, and it is iterated h_b times. We iterate $y \leftarrow y^{2^b}$ for h_b times, and each iteration requires b times multiplications. □

This order is comparable to the asymptotically optimal method given in [1]. The optimal value of b depends on other parameters. In the following implementations, we selected the optimum value of b through experimentation. Sometimes $b = 5$ is optimal (see §4). The consumed time for computing a power using an optimal b (including the time for constructing the precomputation table) is roughly $1/3$ of that using $b = 1$.

INVERSE. To compute the inverse, we may use the following method by Itoh and Tsujii [7, Theorem 3] (they describe the case $q = 2^m$, but the method is applicable to general q).

For $x \in (\mathbf{F}_{q^d})^\times$ and an integer s, let us define

$$N_s(x) := \prod_{j=0}^{s-1} x^{q^j}.$$

Thus, N_d is the norm function and $N_d(x) \in \mathbf{F}_q^\times$. Its inverse can be computed in $\mathbf{F}_q$. Then we have

$$x^{-1} = (N_{d-1}(x))^q (N_d(x))^{-1}. \tag{4}$$

We can compute $N_{d-1}(x)$ using $1.5\lfloor \log_2(d-1) \rfloor$ times multiplications in average, by reading the 2-adic expansion of $d - 1$ from the top to the bottom and by using

Table 1. Consumed time for four operations (each iterated 10000 times) by CGR and NTL for $\mathbf{F}_{2^{130}}$.

Operation	CGR (sec.)	NTL (sec.)
frobenius	0.008	0.011
multiplication	2.417	0.054
inverse	19.448	0.048
power	277.641	2.798

$$N_{2s}(x) = N_s(x) \cdot (N_s(x)^{q^s}), \qquad N_{2s+1}(x) = (N_s(x) \cdot (N_s(x)^{q^s}))^q \cdot x.$$

This scheme computes the inverse of x by $1.5\lfloor \log_2(d-1) \rfloor + 1$ times multiplications in $\mathbf{F}_{q^d}$ and one time inversion in $\mathbf{F}_q$. With a little more care, we obtain the following. Let $H_w(n)$ denote the number of 1's in the 2-adic expansion of a positive integer n (i.e. Hamming weight).

THEOREM 3.3 [7]. *An inversion in $\mathbf{F}_{q^d}$ using (4) requires $\lfloor \log_2(d-1) \rfloor + H_w(d-1) - 1$ times multiplications in $\mathbf{F}_{q^d}$ if $q = 2$, and in addition one time multiplication in $\mathbf{F}_{q^d}$ and one time inversion in $\mathbf{F}_q$ if $q > 2$.*

This theorem is applicable not only to CGR but also to any $\mathbf{F}_{q^d}$, if time consumed by frobenius operation is negligible.

4. Comparison of speed

We implemented the above algorithm in C-programming language, and compared its speed with NTL. We implemented only the cases $p = q$ and $d = m - 1$.

4.1. $p = q = 2$. In this case, NTL is much faster than ours. Table 1 lists the consumed time (in seconds) for frobenius operation, multiplication, inversion, and power, each iterated for 10000 times, for $p = q = 2$ and $d = 130$. CGR means our algorithm.

This superiority of NTL seems to come from the following. A unit object which NTL treats is a polynomial of degree up to 32, represented in a 32-bit word. NTL has a very fast multiplication of two such polynomials (resulting in a 64-bit word), well optimized for pipelined 32-bit machines. While, the unit object for CGR is 1-bit.

We tested a modified version of CGR where all the 32-bits in one word are used, but it is not much faster than the original, since the permutation of the coefficients becomes more difficult and time-consuming.

We also tested smaller m's, and similar superiority of NTL was observed.

Table 2. Consumed time (sec.) for four operations (each iterated 10000 times) by CGR and NTL for $\mathbf{F}_{p^{18}}$, $p = 3, 13, 127, 8191, 32713$.

	$p=3$		$p=13$		$p=127$	
Operation	CGR	NTL	CGR	NTL	CGR	NTL
frobenius	<0.01	3.596	<0.01	8.280	<0.01	20.623
multiplication	0.119	2.056	0.148	2.172	0.186	2.298
inverse	0.815	4.817	0.860	7.540	1.041	8.434
power	2.304	59.090	3.109	142.759	6.218	266.819

	$p=8191$		$p=32713$	
Operation	CGR	NTL	CGR	NTL
frobenius	<0.01	40.821	<0.01	41.858
multiplication	0.126	2.176	0.237	2.292
inverse	0.977	8.804	1.173	8.918
power	10.149	527.647	12.871	665.900

Table 3. Consumed time (sec.) for frobenius operation, multiplication, inverse (each iterated 10000 times), and power (one time), using CGR and NTL for $\mathbf{F}_{8191^d}$, $d = 46, 82, 102, 136$.

	$d=46$		$d=82$		$d=102$		$d=136$	
Operation	CGR	NTL	CGR	NTL	CGR	NTL	CGR	NTL
frobenius	<0.01	171.174	<0.01	352.017	<0.01	374.146	<0.01	720.518
multiplication	0.763	10.004	1.652	20.302	2.350	21.641	3.379	41.832
inverse	6.730	43.869	17.701	112.583	30.244	159.630	53.812	262.646
power*	0.023	0.845	0.084	2.448	0.148	3.565	0.356	8.400

(* Power is iterated only one time.)

4.2. $p = q > 2$. In this case, our method is much faster than NTL. Table 2 lists the consumed time for frobenius operation, multiplication, inversion, and power by CGR and NTL (each iterated 10000 times), for fixed value of $d = 18$ with five selected values $p = 3, 13, 127, 8191, 32713$, each of which generates $(\mathbf{Z}/19)^{\times}$. For optimal power-computation using Theorem 3.3, the value $b = 3, 4, 4, 5, 5$ (respectively for $p = 3, 13, 127, 8191, 32713$) is selected through experimentation. For each power-computation, the precomputation table of size 2^b is constructed.

Table 3 lists a similar table, where $p = 8191$ is fixed and d is varied among $46, 82, 102, 136$. First three kinds of operations are iterated 10000 times, whereas power is executed only one time, since it is time-consuming. The value $b = 5$ is chosen, which is optimal except for $d = 136$. In each power-computation, the precomputation table of size 2^b is constructed.

Table 4. Consumed time (sec.) for four operations (each iterated 10000 times) by CGR for $\mathbf{F}_{8191^{136}}$ and NTL for $\mathbf{F}_{2^{1768}}$.

Operation	CGR	NTL
frobenius	0.012	0.146
multiplication	3.961	1.520
inverse	53.812	12.752
power	2121.4	744.4

These results show that arithmetics in CGR is much faster than NTL for $p = q > 2$. For example, in the case of $\mathbf{F}_{8191^{136}}$, multiplication is 12 times, inversion is 5 times, power is 23 times faster, respectively.

In NTL, arithmetic operations over $\mathbf{F}_2$ are very fast. Table 4 shows the comparison of the speed of CGR for $\mathbf{F}_{8191^{136}}$ and NTL for $\mathbf{F}_{2^{1768}}$ (note that $8191^{136} = 2^{1767.98\cdots}$). Here, for power-computation in CGR, the optimal value $b = 7$ is selected through experimentation. The precomputation table of size 2^b is constructed in each power-computation. Among 2121.4 seconds for the power-computation, precomputation consumes 792.4 seconds. In this comparison, NTL is faster than CGR by a factor of $3 \sim 4$.

Although not listed, the consumed time for power-computation when $b = 5$ is 2187.1 seconds, not much more than 2121.4 seconds for $b = 7$.

5. Conclusion

We give an efficient software implementation of arithmetic operations in a finite field $\mathbf{F}_{q^d}$, using the cyclic group ring over $\mathbf{F}_q$. From the viewpoint of the efficiency, the method is practical when q generates the multiplicative group $(\mathbf{Z}/(d+1))^{\times}$.

In these implementations, the q-th power frobenius operation is very efficient. We introduced two algorithms for general $\mathbf{F}_{q^d}$, which is practical if the frobenius operation is efficient. One is an algorithm for power (Theorem 3.2), and the other is a version of Itoh-Tsujii inversion for $q \neq 2$ (Theorem 3.3).

For $q = 2$, the naive method adopted in NTL is faster than ours, but for $q > 2$ ours are faster by a factor of 5 to 23.

These would be useful in a cryptographic system, where we need a variety of efficient discrete log problems (not only those based on $\mathbf{F}_{2^d}$).

Acknowledgment

The authors are thankful to P. Hellekalek for informing us of NTL and other fast algorithms for finite field arithmetics.

References

[1] E. F. Brickell, D. M. Gordon, K. S. McCurley and D. B. Wilson, Fast exponentiation with precomputation (Extended Abstract). in: Proceedings of Eurocrypt '92, Lecture Notes in Computer Science **658**, 200–207, (1993).
[2] G. Drolet, A new representation of elements of finite fields GF(2^m) yielding small complexity arithmetic circuits. IEEE Transaction on Computers, **47**, 938–946, (1998).
[3] C. Hooley, On Artin's conjecture. J. Reine Angew. Math. **225**, 209–220, (1967).
[4] S. Gao, J. von zur Gathen and D. Panario, Gauss periods and fast exponentiation in finite fields, Lecture Notes in Computer Science **911**, 311–322, (1995).
[5] S. Gao, J. von zur Gathen, D. Panario and V. Shoup, Algorithms for exponentiation in finite fields. J. Symbolic Computation **29**, 879–889, (2000).
[6] W. Geiselmann and H. Lukhaub, Redundant representation of finite fields. in: Proc. Public Key Cryptography, Lecture Notes in Computer Science **1992**, 339–352, (2001).
[7] T. Itoh and S. Tsujii, A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. Information and Computation **78**, 171–177, (1988).
[8] D. Jungnickel, Finite Fields, Structure and Arithmetics. B.I. and F.A. Brockhaus AG, Mannheim, 1993.
[9] D. E. Knuth, The Art of Computer Programming. Vol. 2. Seminumerical Algorithms, 3rd Ed. Addison-Wesley, Reading, Mass., 1997.
[10] H. Wu, M. A. Hasan, I. F. Blake and S. Gao, Finite field multiplier using redundant representation. IEEE Transaction on Computers, **51**, 1306–1316, (2002).

Makoto Matsumoto
Department of Mathematics
Graduate School of Science
Hiroshima University
Hiroshima 739-8526 *JAPAN*
E-mail address: m-mat@math.sci.hiroshima-u.ac.jp

Shigehiro Tagami
Department of Mathematics
Graduate School of Science
Hiroshima University
Hiroshima 739-8526 *JAPAN*