J. Cryptology (1993) 6: 3–13

Journal of Cryptology

Arithmetic Operations in $GF(2^m)$

G. B. Agnew
Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

T. Beth
Universitat Karlsruhe, Postfach 6980, D-7500 Karlsruhe 1, Federal Republic of Germany

R. C. Mullin and S. A. Vanstone
Department of Combinatorics and Optimization and Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

Communicated by Ernest F. Brickell

Received 26 January 1988 and revised 25 October 1991

Abstract. This article is concerned with various arithmetic operations in $GF(2^m)$. In particular we discuss techniques for computing multiplicative inverses and doing exponentiation. The method used for exponentiation is highly suited to parallel computation. All methods achieve much of their efficiency from exploiting a normal basis representation in the field.

Key words. Public key cryptography, Normal basis, Discrete exponentiation.

1. Introduction

In this article we are concerned with arithmetic operations in the finite field $GF(2^m)$. In particular we discuss the computation of multiplicative inverses and exponentiation.

We can think of the elements in $GF(2^m)$ as being m-tuples which form an m-dimensional vector space over $GF(2)$. If

$$\beta, \beta^2, \beta^4, \ldots, \beta^{2^{m-1}}$$

is a basis for this space, then we call it a normal basis and we call β a generator of the normal basis. It is well known [5] that $GF(2^m)$ contains a normal basis for every $m \geq 1$. It is of interest to point out that recently it was shown [4] that a normal basis exists in $GF(2^m)$ with the additional property that a generator of the normal basis is also a generator for the entire multiplicative cyclic group of the field.

For $a \in GF(2^m)$ let $(a_0, a_1, \ldots, a_{m-1})$ be the coordinate vector of a relative to the ordered normal basis N generated by β. It follows that a^2 then has coordinate vector $(a_m - 1, a_0, a_1, \ldots, a_m - 2)$, so squaring is simply a cyclic shift of the vector representation of a. In a hardware implementation squaring an element takes one clock cycle and so is negligible. For the remainder of this article we assume that squaring an element is "free".[1] In Section 2 we address the problem of computing inverses in $GF(2^m)$ and in Section 3 we discuss methods for speeding up exponentiation in such fields.

2. Computing Inverses

Let α be any nonzero element of $GF(2^m)$. Suppose we want to compute α^{-1}. We observe that

$$\begin{aligned}\alpha^{-1} &= \alpha^{2^m-2} \\ &= \alpha^{\sum_{i=1}^{m-1} 2^i} \\ &= \prod_{i=1}^{m-1} \alpha^{2^i}.\end{aligned}$$

This can be computed in $m - 2$ multiplications. We now describe several techniques for improving the situation.

The most efficient technique in terms of minimizing the number of multiplications to compute an inverse was proposed by Itoh *et al.* [3]. Since this reference is not easily accessible, we describe the technique and give a proof of its operation count below.

Consider $GF(2^m)$ and the factorizations

$$2^{m-1} - 1 = \begin{cases}(2^{(m-1)/2} - 1)(2^{(m-1)/2} + 1), & m \text{ odd}, \\ 2^{m-2} + (2^{(m-2)/2} - 1)(2^{(m-2)/2} + 1), & m \text{ even}.\end{cases}$$

For m odd, we require one multiplication to compute $\alpha^{2^{m-1}-1}$ assuming $\alpha^{2^{(m-1)/2}-1}$ has been evaluated. For m even, we require two multiplications to calculate $\alpha^{2^{m-1}-1}$ assuming $\alpha^{2^{(m-1)/2}-1}$ has been evaluated. Hence, to compute α^{-1} in $GF(2^m)$ we show that using this procedure recursively the number of multiplications required is

$$nb(m-1) + \omega(m-1) - 2, \qquad (*)$$

where $nb(x)$ is defined as the minumum number of bits required to represent the integer x and $\omega(m-1)$ is the Hamming weight of the binary representation of $m - 1$. We note that there are precisely $\omega(m-1)$ values of t in the recursion where α^{2^t-1} must be evaluated and t is odd. Before giving the proof we consider an example.

[1] In our comparison of techniques based on the number of clock cycles required, the overhead for squaring will be taken into account.

2.1. *Example* 1

Consider $GF(2^{593})$.

$$\begin{aligned}\alpha^{-1} &= \alpha^{2^{593}-2}\\ &= \alpha^{2(2^{592}-1)}\\ &= \alpha^{2(2^{37}-1)(2^{37}+1)(2^{74}+1)(2^{148}+1)(2^{296}+1)},\\ \alpha^{2^{37}-1} &= \alpha^{2(2^9-1)(2^9+1)(2^{18}+1)+1},\\ \alpha^{2^9-1} &= \alpha^{2(2+1)(2^2+1)(2^4+1)+1}.\end{aligned}$$

2.1.1. *Hardware Implementation.* To implement this function we regroup the above equation using the following steps:

Step 1. Calculate

$$\gamma = \alpha^{2(2^{37}+1)(2^{74}+1)(2^{148}+1)(2^{296}+1)}.$$

Using the fact that squaring is a cyclic shift, γ can be calculated with four multiplications and 556 cyclic shift operations.

Step 2. Store γ. Compute

$$\beta = \gamma^{2(2^9+1)(2^{18}+1)}.$$

This requires two multiplications and 28 cyclic shift operations.

Step 3. Store β. Compute

$$\delta = \beta^{2(2+1)(2^2+1)(2^4+1)}.$$

This requires three multiplications and eight cyclic shift operations.

Step 4. Now compute $\delta\beta\gamma$, which requires a further two multiplications.

In total, this inverse requires 11 multiplications and the storage of two intermediate results (of 593 bits). If we use an optimal normal basis multiplier [1], multiplications can be performed in m clock cycles. Thus the total number of clock cycles to compute an inverse in $GF(2^{593})$ using this method is 7115. We note that $nb(592) = 10$ and $\omega(592) = 3$.

It is a simple matter to prove (*) by induction on m. The result is clearly true for $m = 1$. Suppose it is true for all $k < m$. We consider two cases.

If m is even, then

$$2^m - 1 = (2^{m/2} - 1)(2^{m/2} + 1)$$

and $\alpha^{2^{m/2}-1}$ can be evaluated in $nb(m/2) + \omega(m/2) - 2$ multiplications. Since $nb(m/2) = (nbm) - 1$ and $\omega(m/2) = \omega(m)$ for even m, the result follows by induction.

If m is odd, then we write

$$2^m - 1 = (2^{m-1} + (2^{(m+1)/2} - 1)(2^{(m-1)/2} + 1)).$$

Now, $\alpha^{2^{(m-1)/2}-1}$ can be evaluated, by the induction hypothesis, in $nb((m-1)/2) + \omega((m-1)/2) - 2$ multiplications. Since $nb((m-1)/2) = (nbm) - 1$ and $\omega((m-1)/2) = \omega(m-1) = \omega(m) - 1$ the result follows.

2.2. *Alternative Methods*

Clearly the above method, although efficient in terms of the number of multiplications required to do an inverse, can be costly in terms of a hardware implementation. In some applications, it is more expedient to tradeoff time for implementation complexity.[2] This was the case in the arithmetic processor build for $GF(2^{593})$ [6]. The following algorithm, while requiring more multiplies, is far less complex to implement in hardware.

Suppose that $m - 1 = gh$. Then

$$\begin{aligned} 2^{m-1} - 1 &= 2^{gh} - 1 \\ &= (2^g - 1)\left(\sum_{i=0}^{h-1} 2^{gi}\right). \end{aligned}$$

Now,

$$\begin{aligned} \alpha^{-1} &= \alpha^{2^m-2} \\ &= \alpha^{2(2^{m-1}-1)} \\ &= \gamma^{(2^g-1)(\sum_{i=0}^{h-1} 2^{gi})}, \end{aligned}$$

where $\gamma = \alpha^2$. Then $\beta = \gamma^{(2^g-1)}$ can be calculated in $g - 1$ multiplications and $\beta^{\sum_{i=0}^{h-1} 2^{gi}}$ can be computed in $h - 1$ multiplications. Hence, α^{-1} can be computed in $g + h - 2$ multiplications. It is clear that the number of multiplications using this approach is minimized when g and h are about $\sqrt{m-1}$. We compare this approach to the previous method.

2.2.1. *Example* 2. Consider $GF(2^{593})$.

$$\begin{aligned} \alpha^{-1} &= \alpha^{2^{593}-2} \\ &= \alpha^{2(2^{16\times 37}-1)} \\ &= \alpha^{2(2^{16\times 36}+2^{16\times 35}+\cdots+2^{16}+1)(2^{16}-1)} \end{aligned}$$

and

$$\alpha^{(2^{16}-1)} = \alpha^{2^{15}+2^{14}+2^{13}+\cdots+2+1}.$$

2.2.2. *Hardware Implementation.* To implement this algorithm for computing inverses we proceed as follows:

Step 1. Calculate

$$\beta = \alpha^{(2^{16}-1)}.$$

This requires 15 multiplications and 15 cyclic shift operations.

Step 2. Compute

$$\beta^{(1+2^{16}+2^{32}+\cdots+2^{16\times 36})}.$$

The result is obtained with one additional cyclic shift. This requires 36 multiplications and 10,657 cyclic shift operations.

---

[2] In our example we consider any extra storage requirements for intermediate results to increase the hardware complexity of the device.

It should be noted that while this method requires 51 multiplications (a total of 40,900 clock cycles), no additional storage of intermediate results is required. This is extremely important where a compact hardware implementation is required.

As a very minor modification of this procedure we observe that if $m-1$ does not factor in a reasonably "nice" way (i.e., one of g or h is small compared with $\sqrt{m-1}$), then we look at the factorization of $m-x$ for some positive integer $x \geq 2$. If $m-x=gh$, then we have

$$\begin{aligned} 2^{m-1}-1 &= 2^{m-1}2^{-(x-1)}2^{(x-1)}-1 \\ &= 2^{m-x}2^{x-1}-2^{x-1}+2^{x-1}-1 \\ &= 2^{x-1}(2^{m-x}-1)+2^{x-1}-1. \end{aligned}$$

The number of multiplications to compute α^{-1} is $g+h-1$.

2.2.3. *Example* 3. Consider $GF(2^{1279})$ and observe that

$$\begin{aligned} 2^{1278}-1 &= (2^{1272}-1)2^6+2^6-1 \\ &= (2^{24\times 53}-1)2^6+(2^6-1) \end{aligned}$$

can be evaluated as above in 63 multiplications.

We conclude this section with the following observations. Suppose that we want to compute α^{-a}. This can be done by first computing $\beta=\alpha^a$ and then applying the techniques of this section to compute β^{-1}. The number of multiplications to compute β is $\omega(a)-1$ and then $nb(m-1)+\omega(m-1)-2$ more multiplications for β^{-1} or

$$nb(m-1)+\omega(m-1)+\omega(a)-3$$

in total. If we compute α^{-a} directly we will require $\omega(2^m-1-a)-1$ multiplications. Hence, if

$$\omega(2^m-1-a) < nb(m-1)+\omega(m-1) = \omega(a)-2$$

or

$$\omega(a) > \tfrac{1}{2}(m-nb(m-1)-\omega(m-1)),$$

then it is better to evaluate α^{-a} directly.

2.2.4. *Example* 4. In $GF(2^m)$ for $1 \leq m \leq 8$ it is always faster to compute α^{-a} for any a, $1 < a \leq 2^m-2$, directly.

We note that for some values of m and for certain applications (e.g., where hardware complexity and speed are a consideration), a combination of the preceding methods is preferable.

3. Discrete Exponentiation

Suppose that we want to compute $\alpha^e \in GF(2^m)$ where

$$e=\sum_{i=0}^{m-1} a_i 2^i, \qquad a_i \in \{0, 1\},$$

then

$$\alpha^e = \prod_{i=0}^{m-1} \alpha^{a_i 2^i}$$

and this requires $A = (\sum_{i=0}^{m-1} a_i) - 1$ multiplications. On average for randomly choose e, A will be about $m/2$ and so we require $m/2$ multiplications to do the exponentiation. We now examine ways of doing better.

Select a positive integer k and rewrite the exponent e as

$$e = \sum_{i=0}^{\lceil m/k \rceil - 1} b_i 2^{ki},$$

where $b_i = \sum_{j=0}^{k-1} a_{j+ki} 2^j$. Of course, each b_i can be represented by a binary k-tuple over Z_2 which we represent by $\bar{b}_i$. We now rewrite e in the form

$$e = \sum_{\overline{w} \in Z_2^k \setminus \{0\}} \left(\sum_{i=0}^{\lceil m/k \rceil - 1} C_{i,w} 2^{ki} \right) w, \qquad C_{i,w} \in \{0, 1\}.$$

3.1.1. *Example* 5. If $e = 2^{10} + 2^8 + 2^7 + 2^6 + 2^4 + 2^3 + 2^1 + 1$ and $k = 2$, then

$$e = (1)2^{10} + (1)2^8 + (1 + 2)2^6 + (1)2^4 + (2)2^2 + (1 + 2)2^0$$

or

$$e = (2^{10} + 2^8 + 2^4)(1 + (0)2) + 2^2(0(1) + 2) + (2^6 + 2^0)(1 + 2).$$

If we let $\lambda(w) = \sum_{i=0}^{\lceil m/k \rceil - 1} C_{i,w} 2^{ki}$, then

$$\begin{aligned} \alpha^e &= \alpha^{\sum_{\overline{w}} \lambda(w) w} \\ &= \prod_{\overline{w}} (\alpha^{\lambda(w)})^w. \end{aligned}$$

On average $\lambda(w)$ will have $m/k2^k$ nonzero terms in it and, hence, will require $m/k2^k - 1$ multiplications to evaluate. Since w is represented by a binary k-tuple, w will have on average $k/2$ nonzero terms and require $k/2 - 1$ multiplications to evaluate β^w. Therefore, to evaluate $\alpha^{\lambda(w)w}$ we need $t = (m/k2^k + k/2 - 2)$ multiplications. Finally, to compute α^e we need t multiplications for each $\overline{w} \in Z_2^k \setminus \{0\}$ and then $2^k - 2$ multiplications to multiply the results together. In total we require

$$\begin{aligned} M(k) &= (2^k - 1)\left\{\frac{m}{k2^k} + \frac{k}{2} - 2\right\} + 2^k - 2 \\ &= (2^k - 1)\left\{\frac{m}{k2^k} + \frac{k}{2} - 1\right\} - 1 \end{aligned}$$

multiplications.

If we use $2^k - 1$ processors in parallel to evaluate each $\alpha^{\lambda(w)w}$ simultaneously, then the number of multiplications is on average

$$T(k) = \frac{m}{k2^k} + \frac{k}{2} + 2^k - 4.$$

3.1.2. *Example* 6. For $m = 2^{10}$ and various values of k we compute $M(k)$ and $T(k)$:

k	$M(k)$	$T(k)$
6	293	—
5	244	37
4	254	30
3	315	48

$M(k)$ is minimized by $k = 5$ and $T(k)$ by $k = 4$.

3.1.3. *Example* 7. For $m = 2^{16}$ and various values of k we compute $M(k)$ and $T(k)$:

k	$M(k)$	$T(k)$
11	15,165	2,052
10	10,638	1,031
9	9,055	527
8	8,924	288
7	9,605	201
6	10,877	234

$M(k)$ is minimized by $k = 8$ and $T(k)$ by $k = 7$.

A more extensive tabulation of the functions $M(k)$ and $T(k)$ is given in the Appendix. It appears at least for small values of m that $M(k)$ and $T(k)$ are minimized for k about $\log_2 \sqrt{m}$. In the next section we analyze these functions to determine the asymptotic behaviour of the minimums.

We note that a worst-case analysis will give a number of multiplications on the order of (m/k). Since for most cryptographic applications we are dealing with random exponents, an average-case analysis seems appropriate.

4. Asymptotic Behaviour of $T(k)$ and $M(k)$

In this section we investigate the asymptotic behaviour of

$$T(k) = \frac{m}{k2^k} + \frac{k}{2} + 2^k - 4$$

and

$$M(k) = (2^k - 1)\left(\frac{m}{k2^k} + \frac{k}{2} - 1\right) - 1.$$

We observe that

$$T'(k) = -m\left(\frac{2^k k \ln 2 + 2^k}{k^2 2^{2k}}\right) + 2^k \ln 2 + \tfrac{1}{2}$$

and

$$M'(k) = 2^k \ln 2\left(\frac{m}{k2^k} + \frac{k}{2} - 1\right) + (2^k - 1)\left(\frac{1}{2} - \frac{m(1 + k \ln 2)}{k^2 2^k}\right).$$

The following lemmas will be useful. Since their proofs are straightforward, they are omitted.

Lemma 4.1.

$$T'(k) = 0$$

if and only if

$$\frac{m}{k2^{2k}} = 1 + \left(\frac{(k/2)(1/2^k) - 1}{1 + k \ln 2}\right).$$

Lemma 4.2.

$$M'(k) = 0$$

if and only if

$$\frac{m}{k^3 2^k} = \frac{\ln 2}{2}\left(\frac{1 + (1/(k \ln 2)) - (2/k) - (1/(k2^k \ln 2))}{1 - (\ln 2/2^k) - (1/k2^k)}\right).$$

It follows now from Lemmas 4.1 and 4.2 that for m sufficiently large

$$T'(k) = 0 \qquad \text{implies} \quad m \approx k2^{2k}$$

and

$$M'(k) = 0 \qquad \text{implies} \quad m \approx \frac{\ln 2}{2} k^3 2^k.$$

From these relations we want to determine the asymptotic value of k in terms of m. This is facilitated by the following result.

Lemma 4.3. *Let $m = k^a 2^{bk} c$ where $a, b, c \in \mathfrak{R}$ and are positive. Then, for sufficiently large k,*

$$k - \frac{1}{b}(\log_2 m - a \log_2 \log_2 m + \log_2 c - a \log_2 b) = O\left(\frac{\log_2 k}{k}\right).$$

Proof.

$$\begin{aligned}\log_2 m - a \log_2 \log_2 m &= a \log_2 k + bk + \log_2 c \\ &\quad - a \log_2 b - a \log_2\left(k + \left(\frac{a}{b}\right)\log_2 k + \left(\frac{1}{b}\right)\log_2 c\right) \\ &= bk + \log_2 c - a \log_2 b \\ &\quad - a \log_2\left(\frac{k + (a/b)\log_2 k + (1/b)\log_2 c}{k}\right)\end{aligned}$$

or

$$\frac{1}{b}(\log_2 m - a \log_2 \log_2 m) = k + \left(\frac{1}{b}\right)\log_2 c - \left(\frac{a}{b}\right)\log_2 b - \left(\frac{a}{b}\right)\log_2\left(1 + \left(\frac{a}{b}\right)\left(\frac{\log_2 k}{k}\right) + \left(\frac{1}{b}\left(\frac{\log_2 c}{k}\right)\right)\right).$$

Now, for sufficiently large k,

$$\log_2\left(1 + \left(\frac{a}{b}\right)\left(\frac{log_2\, k}{k}\right) + \left(\frac{1}{b}\right)\left(\frac{\log_2 c}{k}\right)\right) \approx \frac{1}{\ln 2}\left(\left(\frac{a}{b}\right)\left(\frac{\log_2 k}{k}\right) + \left(\frac{1}{b}\right)\left(\frac{\log_2 c}{k}\right)\right).$$

(Recall: $\log_2(1 + \varepsilon) \approx \varepsilon/\ln 2$ for $-1 < \varepsilon < 1$.) □

A simple rearrangement proves the stated result. We are now in a positive to determine the asymptotic behaviour of the minimums of $T(k)$ and $M(k)$.

Theorem 4.4. *For m sufficiently large:*

(a) *If* $k^* = \frac{1}{2}(\log_2 m - \log_2 \log_2 m) - \frac{1}{2}$, *then* $T(k^*)$ *will be in the neighbourhood of the minimum of* $T(k)$.

(b) *If* $k^* = (\log_2 m - 3 \log_2 \log_2 m) + \log_2 \ln 2$, *then* $M(k^*)$ *will be in the neighbourhood of the minimum of* $M(k)$.

Proof. To prove (a), we see from Lemma 4.1 that points in the neighbourhood of the critical point satisfy $n = k2^{2k}$ when n is sufficiently large. Applying Lemma 4.3 with $a = 1$, $b = 2$, and $c = 1$ gives the stated value of k^*. We observe that, for sufficiently large values of m,

$$T(1) > T(k^*) < T(m),$$

which implies that $T(k^*)$ is in the neighbourhood of the minimum of the $T(k)$. The proof of (b) follows similarly. □

There are many situations in cryptographic schemes where exponentiation is always done using the same exponent. For instance, in the Diffie–Hellman scheme [2] a user's private key is used repeatedly for exponentiation. In this situation it may be advantageous to use techniques analogous to the methods introduced in Section 2 for computing inverses. Suppose e is the exponent under consideration and for some nonnegative integer x we have

$$e - x = \prod_{i=1}^{t} n_i,$$

where the n_i are positive integers greater than 1 and not necessarily primes. Then α^e can be computed in

$$\left\{\sum_{i=1}^{t} \omega(n_i) - 1\right\} + \omega(x) - 1$$

multiplications.

4.1.1. *Example* 8. If $e = 45$, then $e = 9 \times 5$ and α^e can be evaluated in $\omega(9) + \omega(5) - 2$ or two multiplications.

4.1.2. *Example* 9. If $e = 237$, then we can take $x = 16$ and write $e = 221 + 16 = 13(17) + 16$ which will require four multiplications to evaluate α^e.

Appendix

Table 1 lists the values of k which minimize $M(k)$ and $T(k)$ for various values of n where n is a power of 2. Table 2 is similar for values of n in increment of 100.

Table 1. Values of k which minimize $M(k)$ and $T(k)$.

	k for min		Min value	
n	$M(k)$	$T(k)$	$M(k)$	$T(k)$
64	3	3	21	8
128	3	3	39	10
256	4	3	74	16
512	4	4	134	22
1024	5	3	243	30
2048	5	5	442	43
4096	6	5	797	56
8192	6	5	1469	81

Table 2. Values of k which minimize $M(k)$ and $T(k)$.

	k for min		Min value	
n	$M(k)$	$T(k)$	$M(k)$	$T(k)$
100	3	3	31	9
200	3	3	60	13
300	4	3	84	18
400	4	4	107	20
500	4	4	131	21
600	4	4	154	23
700	4	4	178	24
800	5	4	200	26
900	5	4	219	28
1000	5	4	239	29
1100	5	4	258	31
1200	5	4	278	32
1300	5	4	297	34
1400	5	4	316	35
1500	5	4	336	37
1600	5	4	355	39
1700	5	4	374	40
1800	5	5	394	41
1900	5	5	413	42
2000	5	5	433	43

References

[1] G. Agnew, R. Mullin, and S. Vanstone, An implementation for a fast public key cryptosystem, *J. Cryptology*, **3**(2), 63–79.

[2] W. Diffie and M. Hellman, New directions in cryptography, *IEEE Trans. Inform. Theory*, **22**(6) (1976), 644–654.

[3] T. Itoh, O. Teechai, and S. Tsujii, A fast algorithm for computing multiplicative inverses in $GF(2^l)$ using normal bases, *J. Soc. Electron. Comm. (Japan)*, **44** (1986), 31–36.

[4] H. W. Lenstra, Jr., and R. J. Schoof, Primitive normal bases for finite fields, *Math. Comp.*, **48** (1987), 217–232.

[5] O. Ore, On a special class of polynomials, *Trans. Amer. Math. Soc.*, **35** (1933), 559–584.

[6] T. Rosati, A high speed data encryption processor for public key cryptography, *Proceeding of the IEEE Custom Integrated Circuits Conference*, San Diego, May 1989, pp. 12.3.1–12.3.5.