Fast Exponentiation in $GF(2^n)$

G.B. Agnew *R.C. Mullin* *S.A. Vanstone*

University of Waterloo
Waterloo, Ontario, Canada

1. Introduction

In this article we will be concerned with arithmetic operations in the finite field $GF(2^n)$. In particular, we examine methods of exploiting parallelism to improve the speed of exponentiation.

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

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

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 ([1]) that $GF(2^n)$ contains a normal basis for every $n \geq 1$. For $a \in GF(2^n)$ let $(a_0, a_1, \ldots, a_{n-1})$ 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 $(a_{n-1}, a_0, a_1, \ldots, a_{n-2})$, 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 will assume that squaring an element is "free".

2. Discrete exponentiation

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

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

Then

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

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

C.G. Guenther (Ed.): Advances in Cryptology - EUROCRYPT '88, LNCS 330, pp. 251-255, 1988.

Select a positive integer k and rewrite the exponent e as

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

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 $\mathbb{Z}_2$ which we represent by $\bar{b}_i$. We now rewrite e in the form

$$e = \sum_{\bar{w} \in \mathbb{Z}_2^k \setminus \{0\}} \left(\sum_{i=0}^{\lceil \frac{n}{h} \rceil - 1} C_{i,w} \, 2^{ki} \right) w, \quad C_{i,w} \in \{0,1\}$$

Example 1. 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 $\lambda(w) = \sum_{i=0}^{\lceil \frac{n}{k} \rceil - 1} C_{i,w} \, 2^{ki}$ then

$$\alpha^e = \alpha^{\sum_{\bar{w}} \lambda(w) w}$$
$$= \Pi (\alpha^{\lambda(w)})^w$$

On average $\lambda(w)$ will have $\frac{n}{k2^k}$ nonzero terms in it and, hence, will require $\frac{n}{k2^k}-1$ multiplications to evaluate. Since w is represented by a binary k-tuple, w will have on average $\frac{k}{2}$ non-zero terms and require $\frac{k}{2}-1$ multiplications to evaluate β^w. Therefore, to evaluate $\alpha^{\lambda(w)w}$ we need $t = \left(\frac{n}{k2^k}+\frac{k}{2}-2 \right)$ multiplications. Finally, to compute α^e we need t multiplications for each $\bar{w} \in \mathbb{Z}_2^k \setminus \{0\}$ and then 2^k-2 multiplications to multiply the results together. In total we require

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

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

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

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

Example 2. For $n = 2^{10}$ 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 $k = 4$.

Example 3. For $n = 2^{16}$ and various values of k we compute $M(k)$ and $T(k)$.

k	$M(k)$	$T(k)$
11	15165	2052
10	10638	1031
9	9055	527
8	8924	288
7	9605	201
6	10877	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 n that $M(k)$ and $T(k)$ are minimized for k about $\log_2 \sqrt{n}$.

Summary

In this paper, we have examined techiques for exponentiating in $GF(2^n)$. These techniques take advantage of parallelism in exponentiation and use processor/time tradeoffs to greatly improve the speed. A more complete study of this problem and other techniques for exploiting parallelism in operation in $GF(2^n)$ is presented in [2].

References

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

[2] G.B. Agnew, R.C. Mullin, S.A. Vanstone, Arithmetic Operations in $GF(2^n)$, *Submitted to the Journal of Cryptology*

Appendix

Table 1 below 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 below is similar for values of n in increment of 100.

	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 1

	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

Table 2