A New Algorithm to Compute Terms in Special Types of Characteristic Sequences

Kenneth J. Giuliani[1] and Guang Gong[2]

[1] Dept. of Mathematical and Computational Sciences
University of Toronto at Mississauga
Mississauga, ON, Canada,
kgiulian@utm.utoronto.ca

[2] Dept. of Electrical and Computer Engineering
University of Waterloo
Waterloo, ON, Canada, N2L 3G1
ggong@calliope.uwaterloo.ca

Abstract. This paper proposes a new algorithm, called the Diagonal Double-Add (DDA) algorithm, to compute the k-th term of special kinds of characteristic sequences. We show that this algorithm is faster than Fiduccia's algorithm, the current standard for computational of general sequences, for fourth- and fifth-order sequences.

Keywords: characteristic sequences, sequence computation, cryptography

1 Introduction

Linear feedback shift register (LFSR) sequences have found an important role in public-key cryptography. One of the first to use them was Niederreiter [12–14] who proposed several cryptosystems based on LFSR's. More recently, cryptosystems such as LUC [11, 17], GH [5], XTR [8], and a fifth-order system [2, 15, 3] have been based upon linear recurrence sequences. As cryptography places a high priority on efficiency of computation, algorithms to compute sequence terms became very important.

Several algorithms to compute sequences terms have been proposed such as by Miller and Spencer-Brown [10], Shortt [16], Shortt and Wilson [20], Gries and Levin [6], Urbanek [18], and Fiduccia [1]. Of these, Fiduccia's appears to be the most efficient.

In this paper, we propose a new algorithm, called the Diagonal-Double-Add (DDA) algorithm to compute sequce terms for special type of sequence which has found use in cryptography. These sequences serve as the basis for XTR [8] and a special fifth-order cryptosystem [15, 3]. We examine its computational cost and show that it is more efficient than Fiduccia's algorithm for fourth- and fifth-order sequences. and show that it is more

This paper is organized as follows. In Section 2, we give background on the special type of sequence we are working with. In Section 3, we introduce the

algorithm and analyze its computational cost in Section 4. We then present and analyze Fiduccia's algorithm in Section 5 and compare the two in Section 6.

2 Preliminaries

The sequences we will be looking at were first proposed for the XTR [8] cryptosystem. They were also employed in a related fifth-order cryptosystem [15, 3]. These sequences were later generalized for use in cryptography for any n [4].

Let $p \equiv 2 \pmod 3$ be a prime and $q = p^2$. We denote by $GF(q)$ the finite field of order q.

Let $f(x)$ be an irreducible polynomial of degree n over $GF(q)$ and α a root of $f(x)$ in $GF(q^n)$. Then the roots of $f(x)$ are $\alpha_i = \alpha^{q^i}$ for $i = 0, \ldots, n-1$. Note that these roots all have the same order in $GF(q^n)$.

Suppose α has order dividing both $(q^n-1)/(q-1) = q^{n-1} + q^{n-2} + \cdots q + 1$ and $(p^{2n}-1)/(p^n-1) = p^n + 1$. Then we may represent $f(x)$ as

$$f(x) = x^n - a_1 x^{n-1} + a_2 x^{n-2} - \cdots + (-1)^{n-1} a_{n-1} x + (-1)^n \tag{1}$$

Note that the constant term is $(-1)^n \alpha^{q^{n-1}+q^{n-2}+\cdots+q+1} = (-1)^n$ by the assumption on the order of α.

Since the order of α also divides $p^n + 1$, we have that $\alpha^{-1} = \alpha^{p^n}$. It then follows that

$$\begin{aligned} a_j &= \sum_{0 \le i_1 < i_2 < \ldots < i_j \le n-1} \alpha^{q^{i_1} + \cdots + q^{i_j}} \\ &= \sum_{0 \le i_1 < i_2 < \ldots < i_{n-j} \le n-1} \alpha^{-(q^{i_1} + \cdots + q^{i_{n-j}})} \\ &= \sum_{0 \le i_1 < i_2 < \ldots < i_{n-j} \le n-1} \alpha^{p^n (q^{i_1} + \cdots + q^{i_{n-j}})} \\ &= \left(\sum_{0 \le i_1 < i_2 < \ldots < i_{n-j} \le n-1} \alpha^{q^{i_1} + \cdots + q^{i_{n-j}}} \right)^{p^n} \\ &= a_{n-j}^{p^n} \end{aligned} \tag{2}$$

for all $j = 1, \ldots, n-1$. Since $a_j \in GF(p^2)$, we must have $a_j = a_{n-j}$ if n is even and $a_j = a_{n-j}^p$ if n is odd.

Consider the recurrence relation of order n over $GF(q)$

$$s_{k+n} = a_1 s_{k+n-1} - a_2 s_{k+n-2} + \cdots + (-1)^n a_{n-1} s_{k+1} + (-1)^{n+1} s_k. \tag{3}$$

The sequence $\{s_i\}$ of elements in $GF(q)$ obtained from (3) with fixed initial conditions

$$s_i = Tr(\alpha^i) = \alpha_0^i + \alpha_1^i + \cdots + \alpha_{n-1}^i \tag{4}$$

for $i = 0, \ldots, n-1$ is called the n-th order characteristic sequence over $GF(q)$ generated by α. The period Q of this sequence is equal to the order of α and defining $s_{-i} = s_{Q-i}$, we get that (4) holds for s_i for all $i \in \mathbb{Z}$. We also have that

$$\begin{aligned} s_{-i} &= \alpha^{-i} + \alpha^{-iq} + \cdots + \alpha^{iq^{n-1}} \\ &= \alpha^{p^n i} + \alpha^{p^n iq} + \cdots + \alpha^{p^n q^{n-1}} \\ &= (\alpha^i + \alpha^{iq} + \cdots + \alpha^{q^{n-1}})^{p^n} \\ &= s_i^{p^n} \end{aligned}$$

Thus, we have $s_{-i} = s_i$ if n is even and $s_{-i} = s_i^p$ if n is odd.

For any integer k, let

$$f_k(x) = x^n - a_{1,k}x^{n-1} + \cdots + (-1)^n a_{n-1,k}x + (-1)^{n+1}$$

be the polynomial whose roots are α_i^k for all $i = 0, \ldots, n-1$. Using a similar argument as in (2), we see that for all $i = 1, \ldots, n-1$, $a_{i,k} = a_{n-i,k}$ if n is even and $a_{i,k} = a_{n-i,k}^p$ if n is odd. We also have an analogue to (3), namely

$$s_{kn+l} = a_{1,k}s_{k(n-1)+l} - \cdots + (-1)^n a_{n-1,k}s_{k+l} + (-1)^{n+1}s_l \tag{5}$$

The s_k and $a_{i,k}$ terms are related by the Newton Formula (see [9]).

Theorem 1 (Newton's Formula). *For a characteristic sequence $\{s_i\}_{i\in\mathbb{Z}}$ and any integers k and i,*

$$s_{i,k} = a_{1,k}s_{(i-1)k} - \cdots + (-1)^i a_{i-1,k}s_k + (-1)^{i+1} i a_{i,k} \tag{6}$$

and hence

$$a_{i,k} = i^{-1}((-1)^{i+1}s_{ik} + (-1)^i a_{1,k}s_{(i-1)k} + \ldots + a_{i-1,k}s_k) \tag{7}$$

The key use for this theorem is that given the sequence terms $s_0, s_k, s_{2k}, \ldots, s_{ik}$, we can efficiently calculate the Newton coefficients $a_{1,k}, a_{2,k}, \ldots, a_{i,k}$ and vice versa.

3 The Diagonal Double-Add Algorithm

We now introduce a new algorithm called the Diagonal Double-Add Algorithm to calculate the k-th term of a characteristic sequence. The following exposition is for odd n. The case for even n is analogous. Let $v = \frac{n-1}{2}$. For each integer j, define the $(n-1)/2 \times n$ array $\hat{S}_j$ as

$$\hat{S}_j = \begin{bmatrix} s_{-v} & s_{-v+1} & \cdots & s_0 & \cdots & s_v \\ s_{j-v} & s_{j-v+1} & \cdots & s_j & \cdots & s_{j+v} \\ s_{2j-v} & s_{2j-v+1} & \cdots & s_{2j} & \cdots & s_{2j+v} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ s_{vj-v} & s_{vj-v+1} & \cdots & s_{vj} & \cdots & s_{vj+v} \end{bmatrix} \tag{8}$$

The (m, l)-entry, that is the entry in the m-th row and l-th column, is s_{jm+l} where m is indexed from 0 to v and l is indexed from $-v$ to v.

Our goal is to calculate either $\hat{S}_{2j}$ or $\hat{S}_{2j+1}$ from $\hat{S}_j$. This will enable us to piece together a double-and-add type of algorithm to compute $\hat{S}_k$ given an integer k. The element s_k can then be read off from $\hat{S}_k$.

To calculate $\hat{S}_{2j}$ from $\hat{S}_j$, we simply add rows to the bottom of $\hat{S}_j$. Given the Newton coefficients $a_{1,j}, \ldots, a_{n-1,j}$, we can use variants of (5)

$$s_{jm+l} = a_{1,j}s_{j(m-1)+l} - a_{2,j}s_{j(m-2)+l} + \cdots + (-1)^{n-1}s_{j(m-n)+l} \tag{9}$$

to progressively compute the sequence terms s_{jm+l} for $m = v+1, \ldots, 2v$. We then keep the rows of even index to form $\hat{S}_{2j}$. We note here that since either $s_{-i} = s_i$ if n is even and $s_{-i} = s_i^p$ if n is odd, all the terms needed to compute these new terms are known since they are in $\hat{S}_j$. Note also that the Newton coefficients can be computed from the elements in $\hat{S}_j$.

To calculate $\hat{S}_{2j+1}$, we need to compute terms of the form $s_{(2j+1)u+w} = s_{2ju+u+w}$ for $u = 0, \ldots, v$ and $w = -v, \ldots, v$. Terms with $2u \leq v$ and $u + w \leq v$ already exist in $\hat{S}_j$. Terms with $2u \leq v$ and $u + w > v$ can be computed using (3) from the terms in the same row. When $2u > v$ and $-v \leq u + w \leq v$, these terms can be computed using (9) as occurred when computing $\hat{S}_{2j}$. However, if $2u > v$ and $u + w > v$, we require another recurrence.

We can compute the Newton coefficients $a_{1,j+1}, \ldots, a_{n-1,j+1}$ from $\hat{S}_j$. We can then subsequently and progressively compute the terms s_{jm+l} where $m, l > v$ by using the recurrence

$$s_{jm+l} = a_{1,j+1}s_{j(m-1)+l-1} - a_{2,j+1}s_{j(m-2)+l-2} + \cdots + (-1)^{n-1}s_{j(m-n)+l-n} \tag{10}$$

Pictorially, the new terms are calculated diagonally about $\hat{S}_j$ instead of vertically as in the double step.

We now formally state the DDA algorithm.

Algorithm 11. DDA

INPUT: A positive integer k
OUTPUT: $\hat{S}_k$

1. Let w and k_i for $i = 0, \ldots, w$ be such that $k = \sum_{i=0}^{w} k_i 2^i$.
2. $B \leftarrow \hat{S}_1$, $j \leftarrow 1$.
3. For i from $w - 2$ down to 0 do
   3.1 Compute the Newton coefficients $a_{1,j}, \ldots, a_{(n-1)/2,j}$.
   3.2 If $k_i = 0$, then $B \leftarrow S_{2j}$, $j \leftarrow 2j$.
   3.3 If $k_i = 1$
      3.3.1 Compute the Newton coefficients $a_{1,j+1}, \ldots, a_{(n-1)/2,j+1}$.
      3.3.2 $B \leftarrow \hat{S}_{2j+1}$, $j \leftarrow 2j + 1$.
4. Output $B = \hat{S}_k$.

---

4 The Computational Cost of the DDA

In this section, we examine the computational cost of the DDA. We start by examining the cost of simple operations.

4.1 Measuring Operations

We will measure the cost of the DDA in terms of the number of multiplications in $GF(p)$ it uses. Additions and subtractions and are not as costly as multiplications and so will not be counted. We shall also assume that k as roughly the same number of 0's as 1's in its binary representation.

The following lemma due to Lenstra and Verheul [8], details the costs of basic operations in $GF(q)$.

Lemma 1 (Lenstra, Verheul). *Suppose* $p \equiv 2 \pmod 3$. *Let* $x, y, z \in GF(p^2)$ *and* $c \in GF(p)$.

- *The p-th power* x^p *is for free.*
- *The squaring* x^2 *requires 2 multiplications in* $GF(p)$.
- *The multiplication* xy *requires* 3 *multiplications in* $GF(p)$.
- *The joint multiplication* $xz - yz^p$ *requires* 4 *multiplications in* $GF(p)$.
- *The scalar multiplication* cx *requires* 2 *multiplications in* $GF(p)$.

This lemma, and the fact that $a_{i,j} = a_{n-i,j}^{p^n}$ tells us that computing a single seuence term from (3), (5), (9), or (refeq:lirecdiag) requires $2(n-2)$ and $3n/2$ multiplications in $GF(p)$ if n is odd and even respectively.

From (7), we see that the i-th Newton coefficients $a_{i,j}$ where $1 \leq i \leq \lfloor n/2 \rfloor$ requires $3(i-1)$ multiplications in $GF(p)$ in addition to the mltiplication by i^{-1}. When i is a power of 2, division by i can essentially be done by a shift of the bits in the representation the elements. Hence it may considered free of cost. Otherwise, $i^{-1} \pmod p$ can be precomputed and requires 2 multiplications in $GF(p)$. Calculating the total cost is now straightforward by summing over i. It is listed in Table 1.

n	# of Multiplications in $GF(p)$
n even	$(3n^2 + 2n - 8 - 16\log n)/8$
n odd	$(3n^2 - n + 1 - 16\log n)/8$

Table 1. The cost of computing the Newton coefficients $a_{1,j}, \ldots, a_{\lfloor n/2 \rfloor, j}$.

4.2 The Computational Cost of the DDA

We now calculate the total cost of the DDA. Please note that some parts of this analysis are excessively tedious. When this is the case, we will omit some of the

details and give the total computational cost. A complete detailed analysis will appear in the full paper.

The cost of the Newton coefficients in Steps 3.1 and 3.3.1 are listed in Table 1.

We calculate $(n-1)/2$ and $n/2$ new rows in Step 3.2 for the case of n odd and even respectively. Each row has n new terms, each of which is derived from an application of (9). Thus, the total cost of this step is the cost of computing $n(n-1)/2$ and $n^2/2$ new terms using (9). It is listed in Table 2.

n	# of Multiplications in $GF(p)$
n even	$3n^3/4$
n odd	$n^3 - 2n^2 + n$

Table 2. The cost of Step 3.2 in the DDA.

The only remaining step to be analyzed is Step 3.3.2. There are $\lfloor n/2 \rfloor$ new rows added to the bottom of $\hat{S}_j$. The new rows of even index have n terms. However, the odd-indexed rows are only needed to calculate the even-indexed terms. Because the right-most term is calculated using (10), odd-indexed rows need to contain only $n-1$ terms. The total cost of the new rows is listed in Table 3.

n	# odd rows	# even rows	total cost (in mults)
$n \equiv 1 \pmod 4$	$(n-1)/4$	$(n-1)/4$	$(2n^3 - 5n^2 + 4n - 1)/2$
$n \equiv 2 \pmod 4$	$(n-2)/4$	$(n+2)/4$	$(6n^3 - 3n^2 + 6n)/8$
$n \equiv 3 \pmod 4$	$(n-3)/4$	$(n+1)/4$	$(2n^3 - 5n^2 + 6n - 3)/2$
$n \equiv 0 \pmod 4$	$n/4$	$n/4$	$(6n^3 - 3n^2)/8$

Table 3. Cost of Adding New Rows in Step 2.3.2 of the DDA

Step 3.3.2 must also compute terms of the form s_{jm+l} with $m \leq v$ and $l > v$. We shall refer to these terms as side terms. Side terms in even-indexed rows are needed to form $\hat{S}_{2j+1}$. Side terms in odd-indexed rows are needed for use in (10) to compute terms in the rows below it. A proper analysis of the cost of these type of terms is extremely tedious. It will be shown in the full paper. The total cost of these terms is listed in Table 4.

Putting these all together, we get the total cost of the DDA as listed in Table 5.

5 Fiduccia's Algorithm

To evaluate the efficiency of the DDA algorithm, we will compare it to the current standard for efficient computation of linear recurrences, Fiduccia's algorithm. In

n	row index	# multiplications in $GF(p)$
$n \equiv 1 \pmod 4$	even	$(n^3+n^2-5n+3)/16$
$n \equiv 2 \pmod 4$	even	$(3n^3-12n)/64$
$n \equiv 3 \pmod 4$	even	$(n^3-3n^2-n+3)/16$
$n \equiv 4 \pmod 4$	even	$(3n^3+12n^2)/64$
$n \equiv 1 \pmod 4$	odd	$(n^3-11n^2+27n-17+8(-1)^{(n-5)/4})/32$
$n \equiv 2 \pmod 4$	odd	$(3n^3+12n^2-12n+24n(-1)^{(n+2)/4})/128$
$n \equiv 3 \pmod 4$	odd	$(n^3+n^2-9n+7+8(-1)^{(n+1)/4})/32$
$n \equiv 4 \pmod 4$	odd	$(3n^3-24n^2+24n+(-1)^{(n-4)/4})/128$

Table 4. Cost of Adding Side Terms in Step 3.3.2 of the DDA

n	# of Multiplications in $GF(p)$
$n \equiv 1 \pmod 4$	$(67n^3-117n^2+65n-207-192\log n+8(n-1)(-1)^{(n-5)/4})/64\log k$
$n \equiv 2 \pmod 4$	$(201n^3-108n^2+156n-384-768\log n+24n(-1)^{(n+2)/4})/256\log k$
$n \equiv 3 \pmod 4$	$(67n^3-113n^2+69n-215-192\log n+8(n-1)(-1)^{(n+1)/4})/64\log k$
$n \equiv 0 \pmod 4$	$(201n^3-84n^2+144n-384-768\log n+24n(-1)^{(n-4)/4})/256\log k$

Table 5. The cost of DDA algorithm.

this section, we describe Fiduccia's algorithm in detail and then analyze its computational cost. Let us start with some background.

The companion matrix C of the linear recurrence (3) is defined as

$$C = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ (-1)^{n+1} & (-1)^n a_{n-1} & (-1)^{n-1} a_{n-2} & \cdots & a_1 \end{bmatrix}$$

The characteristic polynomial of C is $f(x)$ as in (1).

We define the sequence vector $\boldsymbol{s}_j$ as

$$\boldsymbol{s}_j = \begin{bmatrix} s_j \\ s_{j+1} \\ \vdots \\ s_{j+n-1} \end{bmatrix}$$

Observe that $C\boldsymbol{s}_j = \boldsymbol{s}_{j+1}$ whence for any positive integer k, $C^k \boldsymbol{s}_j = \boldsymbol{s}_{j+k}$.

Fiduccia makes use of the Cayley-Hamilton theorem.

Theorem 2 (Cayley-Hamilton). *Let $\lambda(x)$ be the characteristic polynomial of an $n \times n$ matrix M. Then $\lambda(M) = 0_n$ where 0_n is the $n \times n$ zero matrix.*

This theorem tells us that $C^k = r(C)$ where $r(x)$ is any polynomial such that $r(x) \equiv x^k \pmod{f(x)}$. Thus, Fiduccia's algorithm computes $r(x)$ in this way, then uses $r(C)$ to get $\boldsymbol{s}_k$. We now state it formally.

Algorithm 12. Fiduccia

INPUT: A positive integer k
OUTPUT: $\boldsymbol{s}_k$

1. $d(x) \leftarrow x$.
2. For i from $w-2$ down to 0 do
   2.1 $d(x) \leftarrow d(x) \times d(x) \bmod f(x)$.
   2.2 If $k_i = 1$, $d(x) \leftarrow d(x) \times x \bmod f(x)$.
3. $D \leftarrow d(C)$ and $\boldsymbol{s}_k \leftarrow D\boldsymbol{s}_0$.

Remark 1. Fiduccia also gave a speedup of Step 3 which only calculated one column of $d(C)$. However, this will not affect the asymptotic complexity of the algorithm so we will not list it here.

Step 2.1 requires a polynomial squaring with reduction modulo $f(x)$. The squaring can be done by using Karatsuba [7] multiplication for a total cost of $n^2 + n$ multiplications in $GF(p)$. The reduction modulo $f(x)$ would cost a total of $3n^2 - 6n + 3$ multiplications.

Step 2.2 requires a multiplication by x with reduction modulo $f(x)$. The multiplication has no cost since we merely shift coefficients. The reduction costs $3(n-1)$ multiplications.

Putting this all together, the total cost of Fiduccia's algorithm is $(8n^2 - 7n + 3)/2 \log k$ multiplications in $GF(p)$.

6 The Efficiency of the DDA

As Fiduccia's algorithm has a highest term of $n^2 \log k$ while the DDA has a highest term $n^3 \log k$, Fiduccia's algorithm is the faster asymptotically for large values of n and for all types of sequences.

The question of which is more efficient for small values of n requires further examination. Table 6 lists the values obtained by substituting $n = 2, 3, 4, 5, 6, 7$ into the cost of Fiduccia's algorithm and Table 5 for the DDA.

n	Fiduccia	DDA
2	10.5 mult	6 mult
3	27 mult	12 mult
4	51.5 mult	52.5 mult
5	84 mult	84.5 mult
6	124.5 mult	183 mult
7	173 mult	274.5 mult

Table 6. Comparison of Fiduccia's Algorithm and the DDA for Small Values of n (in $\log k$ operations)

Table 6 shows that for characteristic sequences where $n \geq 6$, Fiduccia is more efficient. For the cases where $n = 2, 3$, the DDA is faster. We note here that in these cases, the DDA is essentially the same as given in the LUC cryptosystem for $n = 2$ and the XTR cryptosystem for $n = 3$. However, in these two cases no Newton coefficients need to be computed since $a_{1,k} = s_k$. The real power of the DDA occurs when $n \geq 4$ since this is where non-trivial Newton coefficients are computed.

For the cases of $n = 4, 5$, the cost is very close between the two algorithms. However, there are some speedups available for the DDA. We examine them in the next two subsections.

For some terms during the course of the DDA, (5) can be made more efficient than the standard count by making several observations. We examine these individually for each n.

6.1 Fourth-Order Sequences

For fourth-order sequences, since $a_{1,k} = s_k$ and $s_0 = 4$, we can rewrite (5) for the following terms.

$$s_{3k} = a_{1,k}s_{2k} - a_{2,k}s_k + a_{1,k}s_0 - s_k = s_k(s_{2k} - a_{2,k} + 3)$$

$$s_{3k+3} = a_{1,k+1}s_{2k+2} - a_{2,k+1}s_{k+1} + a_{1,k+1}s_0 - s_{k+1} = s_{k+1}(s_{2k+2} - a_{2,k+1}) + 3s_{k+1}$$

The cost of 6 multiplications in $GF(p)$ changes to 3 for s_{3k} and s_{3k+3}. Note also that the cost of computing the Newton coefficients $a_{2,k}$ and $a_{2,k+1}$ changes 3 multiplications to 2 multiplications in $GF(p)$.

Thus, the total cost of the DDA becomes $48 \log k$ multiplications.

6.2 Fifth-Order Sequences

For fifth-order sequences, (5) requires 4 multiplications in $GF(q)$ for characteristic sequences or 8 multiplications in $GF(p)$ for LV sequences. Again since $a_{1,k} = s_k$ and $s_0 = 5$, we can rewrite (5) for the following terms.

$$s_{3k} = s_k(s_{2k} - a_{2,k} + 5a_{2,k}^p - (s_k^p)^2 + s_{2k}^p$$

$$s_{3k+3} = s_{k+1}(s_{2k+2} - a_{2,k+1} + 5a_{2,k+1}^p - (s_{k+1}^p)^2 + s_{2k+2}^p$$

$$s_{4k} = s_k(s_{3k} + a_{2,k}^p) - a_{2,k}s_{2k} - 4s_k^p$$

$$s_{4k+4} = s_{k+1}(s_{3k+3} + a_{2,k+1}^p) - a_{2,k+1}s_{2k+2} - 4s_{k+1}^p$$

This reduces from a cost of 8 multiplications in $GF(p)$ to 4 for s_{3k} and s_{3k+3} and 6 for s_{4k} and s_{4k+4}. Note also that the cost of computing the Newton coefficients $a_{2,k}$ and $a_{2,k+1}$ (and their negatives) changes from 3 multiplications to 2 multiplications in $GF(p)$.

Thus the total cost of the DDA becomes $74 \log k$ multiplications. It should be noted that Quoos and Mjølsnes [15] also gave an algorithm for computing fifth-order sequences of this type. However, the computational cost of their algorithm was found to be $102 \log k$ multiplications in $GF(p)$.

6.3 Summary

These figures are listed in Table 7.

n	Fiduccia	DDA
4	51.5 mult	48 mult
5	84 mult	74 mult

Table 7. Updates Comparison of Fiduccia and the DDA for Small Values of n (in $\log k$ operations)

Hence, the DDA is the faster algorithm for $n = 4, 5$.

References

1. Fiduccia, C.M.: An Efficient Formula for Linear Recurrences. SIAM J. Comput. **14** (1985) 106–112.
2. Giuliani, K., Gong, G.: Analogues to the Gong-Harn and XTR Cryptosystems. Combinatorics and Optimization Research Report CORR 2003-34, University of Waterloo (2003).
3. Giuliani, K., Gong, G.: Efficient Key Agreement and Signature Schemes Using Compact Representations in $GF(p^{10})$. In: Proceedings of the 2004 IEEE International Symposium on Information Theory - ISIT 2004. Chicago (2004) 13–13.
4. Giuliani, K., Gong, G.: New LFSR-Based Cryptosystems and the Trace Discrete Log Problem (Trace-DLP). In: Sequence and Their Applications – SETA 2004. Lecture Notes In Computer Science, Vol. 3486. Springer-Verlag, Berlin Heidelberg New York (2005) 298–312.
5. Gong, G., Harn, L.: Public-Key Cryptosystems Based on Cubic Finite Field Extensions. IEEE Trans. IT. **24** (1999) 2601–2605.
6. Gries, D., Levin, D.: Computing Fibonacci Numbers (and Similarly Defined Functions) in Log Time. Information Processing Letters **11** (1980) 68–69.
7. Karatsuba, A., Ofman, Y.: Mulplication of Many-Digital Numbers by Automatic Computers. Physics-Daklady **7** (1963) 595–596.
8. Lenstra, A., Verheul, E.: The XTR Public Key System. In: Advances in Cryptology – Crypto 2000. Lecture Notes In Computer Science, Vol. 1880. Springer-Verlag, Berlin Heidelberg New York (2000) 1–19.
9. Lidl, N., Niederreiter, H.: Finite Fields. Addison-Wesley, Reading (1983).
10. Miller, J. C. P., Spencer-Brown, D. J.: An Algorithm For Evaluation of Remote Terms in a Linear Recurrence Sequence. Computer Journal **9** (1966/67) 188–190.
11. Müller, W. B., Nobauer, R.: Cryptanalysis of the Dickson scheme. In: Advances in Cryptology – Eurocrypt 1985. Lecture Notes In Computer Science, Vol. 219. Springer-Verlag, Berlin Heidelberg New York (1986) 50–61.
12. Niederreiter, H.: A Public-Key Cryptosystem Based on Shift-Register Sequences. In: Advances in Cryptology – Eurocrypt 1985. Lecture Notes In Computer Science, Vol. 219. Springer-Verlag, Berlin Heidelberg New York (1986) 35–39.
13. Niederreiter, H.: Some New Cryptosystems Based on Feedback Shift Register Sequences. Math. J. Okayama Univ. **30** (1988) 121-149.

14. Niederreiter, H.: Finite Fields and Cryptology. In: Finite Fields, Coding Theory, and Advances in Communications and Computing. M. Dekker, New York (1993) 359–373.
15. Quoos, L., Mjølsnes, S.-F.: Public Key Systems Based on Finite Field Extensions of Degree Five. Presented at Fq7 conference (2003).
16. Shortt, J.: An Iterative Algorithm to Calculate Fibonacci Numbers in $O(\log n)$ Arithmetic Operations. Information Processing Letters **7** (1978) 299–303.
17. Smith, P., Skinner, C.: A Public-Key Cryptosystem and a Digital Signature System Based on the Lucas Function Analogue to Discrete Logarithms. In: Advances in Cryptology – Asiacrypt '94. Lecture Notes In Computer Science, Vol. 917. Springer-Verlag, Berlin Heidelberg New York (1994) 357–364.
18. Urbanek, F. J.: An ($O(\log n)$ Algorithm for Computing the nth Element of a Solution of a Difference Equation. Information Processing Letters **11** (1980) 66–67.
19. Ward, M.: The Algebra of Recurring Series. Annals of Math **32** (1931) 1–9.
20. Wilson, T. C., Shortt, J.: An $O(\log n)$ Algorithm for Computing General Order-k Fibonacci Numbers. Information Processing Letters **10** (1980) 68–75.