

AN ABSTRACT OF THE DISSERTATION OF

Berk Sunar for the degree of Doctor of Philosophy in

Electrical and Computer Engineering presented on November 6, 1998.

Title: Fast Galois Field Arithmetic for Elliptic Curve Cryptography
and Error Control Codes

Abstract approved: _____
Redacted for Privacy
Ç. K. Koç

Today's computer and network communication systems rely on authenticated and secure transmission of information, which requires computationally efficient and low bandwidth cryptographic algorithms. Among these cryptographic algorithms are the elliptic curve cryptosystems which use the arithmetic of finite fields. Furthermore, the fields of characteristic two are preferred since they provide carry-free arithmetic and at the same time a simple way to represent field elements on current processor architectures.

Arithmetic in finite field is analogous to the arithmetic of integers. When performing the multiplication operation, the finite field arithmetic uses reduction modulo the generating polynomial. The generating polynomial is an irreducible polynomial over $GF(2)$, and the degree of this polynomial determines the size of the field, thus the bit-lengths of the operands.

The fundamental arithmetic operations in finite fields are addition, multiplication, and inversion operations. The sum of two field elements is computed very easily. However, multiplication operation requires considerably more effort compared to addition. On the other hand, the inversion of a field element requires much

more computational effort in terms of time and space. Therefore, we are mainly interested in obtaining implementations of field multiplication and inversion.

In this dissertation, we present several new bit-parallel hardware architectures with low space and time complexity. Furthermore, an analysis and refinement of the complexity of an existing hardware algorithm and a software method highly efficient and suitable for implementation on many 32-bit processor architectures are also described.

©Copyright by Berk Sunar

November 6, 1998

All rights reserved

Fast Galois Field Arithmetic for Elliptic Curve Cryptography and Error Control
Codes

by

Berk Sunar

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Completed November 6, 1998
Commencement June 1999

Doctor of Philosophy dissertation of Berk Sunar presented on November 6, 1998

APPROVED:

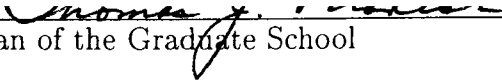
Redacted for Privacy

Major Professor, representing Electrical and Computer Engineering

Redacted for Privacy

Chair of the Department of Electrical & Computer Engineering

Redacted for Privacy


Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Redacted for Privacy

Berk Sunar, Author

ACKNOWLEDGMENT

I would like express my gratitude to my advisor Çetin K. Koç for his continued guidance and support throughout this work.

TABLE OF CONTENTS

	<u>Page</u>
1. INTRODUCTION	1
2. LOW-COMPLEXITY BIT-PARALLEL CANONICAL AND NORMAL BASIS MULTIPLIERS FOR A CLASS OF FINITE FIELDS	4
2.1. Introduction	4
2.2. Canonical Basis Multiplier.....	5
2.3. Normal Basis Multiplier	11
2.4. Conclusions	13
3. FAST FINITE FIELD ARITHMETIC AND ELLIPTIC CURVE CRYPTOG- RAPHY OVER COMPOSITE FIELDS	16
3.1. Introduction	16
3.2. Representation of Field Elements in Composite Fields.....	17
3.3. Representation and Arithmetic in $GF(2^n)$	19
3.4. Representation and Arithmetic in $GF(2^{n \cdot m})$	21
3.4.1. Squaring.....	22
3.4.2. Multiplication	23
3.4.3. Inversion	25
3.5. Elliptic Curve Operations	28
3.6. Implementation Results and Conclusions.....	30
4. MASTROVITO MULTIPLIER FOR ALL TRINOMIALS	33
4.1. Introduction	33
4.2. The Reduction Process	34
4.3. The Multiplier Architecture	40
4.4. Space and Time Complexity.....	43
4.5. An Example	45
4.6. The Special Case of $n = m/2$	51
4.7. Acknowledgements	54
5. A NEW MULTIPLIER FOR OPTIMAL NORMAL BASIS OF TYPE II	55

TABLE OF CONTENTS (Continued)

	<u>Page</u>
5.1. Introduction	55
5.2. Representation of Field Elements	56
5.3. New Multiplication Algorithm	58
5.4. Details of Multiplication and Complexity Analysis	61
5.5. An Example	63
5.6. Conclusions	66
 BIBLIOGRAPHY	 68

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
2.1 The proposed canonical basis multiplier.....	6
2.2 The connection diagram for computing D	9
2.3 The rewiring modules used in the connection diagram.....	10
2.4 The proposed normal basis multiplier.....	12

LIST OF TABLES

<u>Table</u>	<u>Page</u>
2.1 Comparing canonical basis multipliers with generating AOPs.	14
2.2 Comparing normal basis multipliers with generating AOPs.	14
3.1 Possible values of n , m , and the field size nm	21
3.2 The timing results for the field and elliptic curve operations.	31
4.1 The reduction array.	36
5.1 The construction of C_1	59
5.2 The construction of D_1	60
5.3 The construction of D_2	61
5.4 The construction of C_1 , D_1 , D_2 , and C in $GF(2^5)$	67

I would like to dedicate this dissertation to my parents,
Sabiha and Fevzi and finally to my love.

FAST GALOIS FIELD ARITHMETIC FOR ELLIPTIC CURVE CRYPTOGRAPHY AND ERROR CONTROL CODES

1. INTRODUCTION

Recent advances in networking and wireless communication technologies stimulated research in areas such as error control codes and cryptography. The most practical and common codes, for example, the Reed-Solomon, BCH, and cyclic codes are based on finite field arithmetic. Furthermore, the proliferation of cryptanalysis rendered RSA -the de facto algorithm of public-key cryptography- impractical due to increasing bit-lengths which is the primary concern in applications where bandwidth is limited. A short time ago elliptic curve cryptography was introduced as an alternative, providing the same amount of security at much lower bit-lengths. Elliptic curve cryptography employs finite field arithmetic.

The main emphasis of this dissertation is on efficient software and hardware implementations of finite field arithmetic operations, which are crucial in certain applications, e.g., elliptic curve cryptosystems and error control codes. In the dissertation we only consider the operations in fields of characteristic 2. These type of fields are preferred due to their carry-free binary nature, which is the basis for efficiency in microprocessor implementations.

In our study, we mainly concentrate on the implementation of the multiplication and inversion operations. There are several designs and algorithms invented to perform these computations for short bit lengths. However, due to recent advances in cryptanalysis, longer bit lengths are required for security. In addition, the

increase in the wordsize of microprocessors results in an increase in the length of the codes used in error correction. Consequently, arithmetic operations with longer operands have become a necessity. Therefore, we believe that all current algorithms should be reevaluated and improved if possible.

In the second chapter, we present a new low-complexity bit-parallel canonical basis multiplier for the field $GF(2^m)$ generated by an all-one-polynomial. The proposed canonical basis multiplier requires $m^2 - 1$ XOR gates and m^2 AND gates. We also extend this canonical basis multiplier to obtain a new bit-parallel normal basis multiplier. This kind of designs are used in applications requiring fast parallel field multiplication such as error control mechanisms in memory units and cryptographic smart cards.

In the third chapter, we present a methodology for efficient software implementation of the arithmetic operations in $GF(2^k)$ where $k = n \cdot m$ is a composite number. The elements of the composite field $GF(2^{n \cdot m})$ are represented using an optimal normal basis in $GF(2^m)$ with the ground field as $GF(2^n)$. The field operations in $GF(2^n)$ are performed using the logarithmic table lookup method. We give detailed description of algorithms for the multiplication, squaring, and inversion operations in $GF(2^{n \cdot m})$. The presented methodology can be used to obtain efficient software implementations of the elliptic curve cryptographic operations over composite fields. The specific case of $GF(2^{16 \cdot 11})$ is treated in detail, and implementation results for the finite field and elliptic curve operations are presented. The timing results show that our implementation is slightly faster than that of [1], in which a polynomial basis in $GF(2^m)$ with the ground field as $GF(2^n)$ is used to represent the elements of $GF(2^{n \cdot m})$.

In the fourth chapter, we describe yet another new design. An efficient algorithm for the multiplication in $GF(2^m)$ was introduced by Mastrovito. The space

complexity of the Mastrovito multiplier for the irreducible trinomial $x^m + x + 1$ was given as $m^2 - 1$ XOR and m^2 AND gates. In this study, we describe an architecture based on a new formulation of the multiplication matrix, and show that the Mastrovito multiplier for the generating trinomial $x^m + x^n + 1$, where $m \neq 2n$, also requires $m^2 - 1$ XOR and m^2 AND gates. However, $m^2 - m/2$ XOR gates are sufficient when the generating trinomial is of the form $x^m + x^{m/2} + 1$ for an even m . We also calculate the time complexity of the proposed Mastrovito multiplier, and give design examples for the trinomials $x^8 + x^5 + 1$ and $x^6 + x^3 + 1$.

In the fifth chapter we present a new multiplier for the field $GF(2^m)$ whose elements are represented using the optimal normal basis of type II. The proposed multiplier requires $1.5(m^2 - m)$ XOR gates, as compared to $2(m^2 - m)$ XOR gates required by the Massey-Omura multiplier which is the only other multiplier working in this basis. The time complexities of the proposed and the Massey-Omura multipliers are similar.

2. LOW-COMPLEXITY BIT-PARALLEL CANONICAL AND NORMAL BASIS MULTIPLIERS FOR A CLASS OF FINITE FIELDS

2.1. Introduction

The arithmetic operations in the Galois field $\text{GF}(2^m)$ have several applications in coding theory, computer algebra, and cryptography [13, 12]. In these applications, time and area efficient algorithms and hardware structures are desired for addition, multiplication, squaring, and exponentiation operations. The performance of these operations is closely related to the representation of the field elements. An important advance in this area has been the introduction of the Massey-Omura algorithm [18], which is based on the normal basis representation of the field elements. One advantage of the normal basis is that the squaring of an element is computed by a cyclic shift of the binary representation. Efficient algorithms for the multiplication operation in the canonical basis have also been proposed [27, 3]. The space and time complexities of these bit-parallel canonical basis multipliers are much less than those of the Massey-Omura multiplier.

In this paper, we present an alternative design for multiplication in the canonical basis for the field $\text{GF}(2^m)$ generated by an all-one-polynomial (AOP). The time complexity of our design is significantly less than similar bit-parallel multiplier designs for the canonical basis [27, 3, 2]. Furthermore, we use the proposed canonical basis multiplier to design a normal basis multiplier, whose space and time complexities are nearly the same as those of the modified Massey-Omura multiplier [26] given for the field $\text{GF}(2^m)$ with an AOP. Nevertheless, the proposed normal basis multiplier is based on a different construction from the ones already known.

2.2. Canonical Basis Multiplier

It is customary to view the field $\text{GF}(2^m)$ as an m -dimensional vector space defined over the ground field $\text{GF}(2)$. We need a set of m linearly independent elements from $\text{GF}(2^m)$ in order to represent the elements of $\text{GF}(2^m)$. This set serves as the basis of the vector space. A basis of the form $S = \{1, \alpha, \alpha^2, \dots, \alpha^{m-1}\}$, where $\alpha \in \text{GF}(2^m)$ is a root of the generating polynomial of degree m , is called a canonical basis. In order to reduce the complexity of the field multiplication, special classes of irreducible polynomials have been suggested [3, 27]. In particular, the AOP $p(x) = 1 + x + x^2 + \dots + x^m$ has been shown to be very useful. This polynomial is irreducible, and thus, generates the field $\text{GF}(2^m)$, if and only if $m+1$ is prime and 2 is primitive modulo $m+1$ [13]. For $m \leq 100$, the AOP is irreducible for the following values of m : 2, 4, 10, 12, 18, 28, 36, 52, 58, 60, 66, 82, and 100.

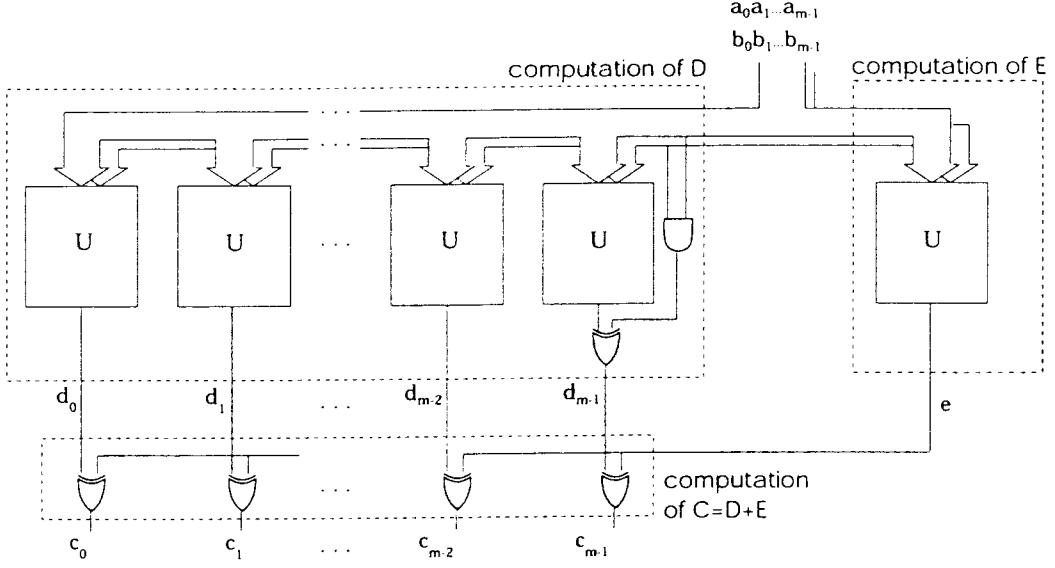
We now briefly describe the Mastrovito multiplier [27] for computing the products of two elements A and B in $\text{GF}(2^m)$ expressed in the canonical basis, which are respectively represented as

$$\begin{aligned} A &= \sum_{i=0}^{m-1} a_i x^i = [a_0 a_1 \dots a_{m-1}]^T \\ B &= \sum_{i=0}^{m-1} b_i x^i = [b_0 b_1 \dots b_{m-1}]^T. \end{aligned}$$

The Mastrovito multiplier uses two matrices in the design process. The $(m-1) \times m$ basis reduction matrix $Q = [q_{ij}]$ satisfies the equality

$$\begin{bmatrix} x^m \\ x^{m+1} \\ \vdots \\ x^{2m-2} \end{bmatrix} = Q \begin{bmatrix} 1 \\ x \\ \vdots \\ x^{m-1} \end{bmatrix}$$

FIGURE 2.1: The proposed canonical basis multiplier.



The $m \times m$ product matrix $Z = f(A, Q) = [z_{ij}]$ is defined as

$$z_{ij} = \begin{cases} a_i, & \text{for } j = 0; i = 0, 1, \dots, m-1, \\ u(i-j)a_{i-j} + \sum_{t=0}^{j-1} q_{j-1-t,i} a_{m-1-t}, & \text{for } j = 1, 2, \dots, m-1; i = 0, 1, \dots, m-1, \end{cases} \quad (2.1)$$

where the step function $u(t)$ is defined as

$$u(t) = \begin{cases} 1 & \text{for } t \geq 0, \\ 0 & \text{else.} \end{cases}$$

The product $C = AB$ is found by multiplying the matrix Z by the vector B in

the ground field $\text{GF}(2)$. The Mastrovito algorithm directly computes this product $C = ZB$.

We introduce a new canonical basis multiplication algorithm for the field $\text{GF}(2^m)$ generated using an AOP by decomposing the matrix Z into the matrices Z_1 and Z_2 as $Z = Z_1 + Z_2$. The idea of decomposing a matrix has proved to be useful in many similar designs [26]. In order to construct these matrices, we first write the matrix equality (2.2) for the matrix Q in the field $\text{GF}(2^m)$ with an AOP using the identity $x^{m+1} = 1$ as

$$\begin{bmatrix} x^m \\ x^{m+1} \\ x^{m+2} \\ \vdots \\ x^{2m-2} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^{m-1} \end{bmatrix} \quad (2.2)$$

Using the definition (2.2) of Q and the definition (2.1) of Z , we construct the product matrix Z for the field $\text{GF}(2^m)$ with an AOP as the sum of two matrices Z_1 and Z_2 which are given as follows:

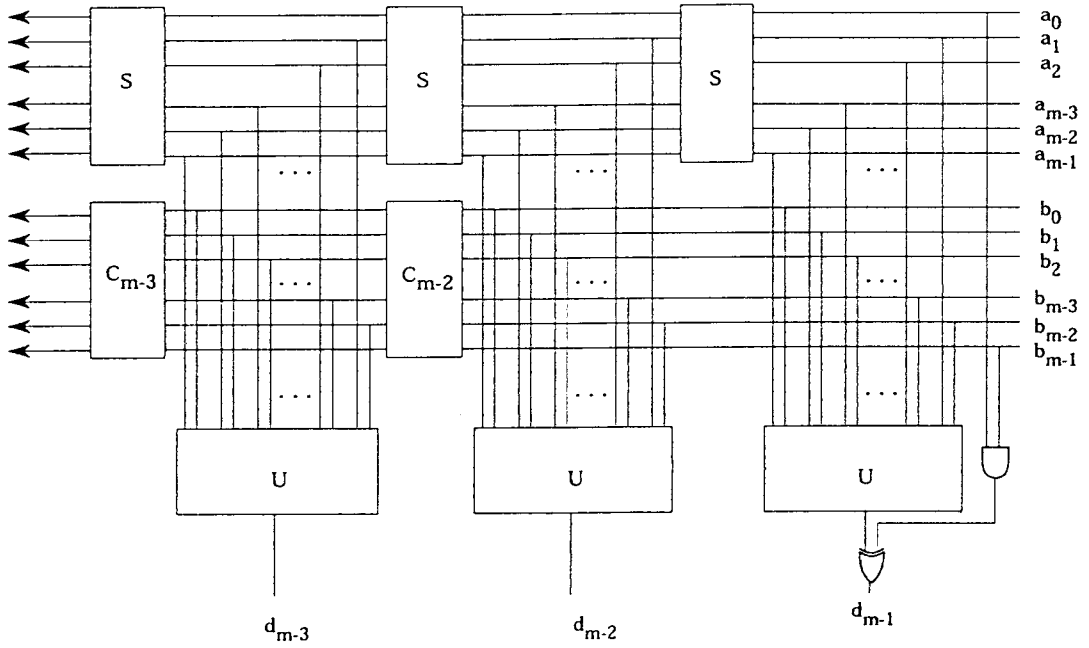
$$Z_1 = \begin{bmatrix} a_0 & 0 & a_{m-1} & a_{m-2} & \cdots & a_2 \\ a_1 & a_0 & 0 & a_{m-1} & \cdots & a_3 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m-2} & a_{m-3} & a_{m-4} & a_{m-5} & \cdots & 0 \\ a_{m-1} & a_{m-2} & a_{m-3} & a_{m-4} & \cdots & a_0 \end{bmatrix},$$

$$Z_2 = \begin{bmatrix} 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_1 \\ 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_1 \\ 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_1 \end{bmatrix}.$$

In order to compute $C = ZB = (Z_1 + Z_2)B$, we first compute $D = Z_1B$ and $E = Z_2B$ in parallel, and then compute the result $C = D + E$. The product of the last row of Z_1 and B is computed using the rightmost U circuit with two additional gates which take care of the nonzero element of the last row of Z_1 . The architecture of the canonical basis multiplier is shown in Figure 2.1. The module which computes the vector $D = Z_1B$ consists of m identical U circuits, an AND, and an XOR gate. The circuit U computes the innerproduct of two vectors of length $m - 1$. Since one element in each row of Z_1 is zero, except in the last row, the innerproduct operation needs to be of length $m - 1$. The vector A is shifted according to the place of the zero element in each row of Z_1 , while

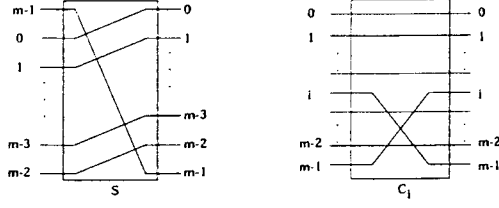
the vector B is fed to the i th U module by skipping the i th bit. The connection diagram of the part of the multiplier computing D is shown in Figure 2.2. The basic rewiring modules used in the connection diagram are defined in Figure 2.3.

The structure of the module U is very simple. The innerproduct of two vectors is computed by first generating the products in parallel using AND gates, and then by adding the partial products using a binary XOR tree. In order to generate the products $m - 1$ AND gates are needed, whereas $m - 2$ XOR gates are used to accumulate the products. The depth of the binary XOR tree is given as $\lceil \log_2(m - 1) \rceil$. The total delay of the circuit U is equal to $T_A + \lceil \log_2(m - 1) \rceil T_X$, where T_A and T_X are the delays of AND and XOR gates, respectively. The computation of

FIGURE 2.2: The connection diagram for computing D .

d_{m-1} requires an additional XOR gate delay.

FIGURE 2.3: The rewiring modules used in the connection diagram.



Thus, the computation of D requires a total of $T_A + (1 + \lceil \log_2(m-1) \rceil)T_X$ delays.

In order to compute $E = Z_2 B$, we need a single U module with inputs according to the definition of Z_2 given above. Since Z_2 has identical rows, the computation of $Z_2 B$ is accomplished by computing the innerproduct of a row of Z_2 and the vector B , and then replicating this resulting bit m times, i.e., $E = [eee \cdots e]$, where e is repeated m times. After $E = Z_2 B$ is computed, the result $C = Z_1 B + Z_2 B = D + E$ is obtained using m XOR gates, as shown in the bottom part of Figure 2.1.

The proposed canonical basis multiplier architecture requires a total of $(m - 2)(m + 1) + 1 + m = m^2 - 1$ XOR gates and $(m - 1)(m + 1) + 1 = m^2$ AND gates. The total delay of the circuit is found as $T_A + (2 + \lceil \log_2(m - 1) \rceil)T_X$. These time and space complexity values are computed assuming only 2-input gates are available.

2.3. Normal Basis Multiplier

A basis of the form

$$N = \{\beta, \beta^2, \beta^{2^2}, \dots, \beta^{2^{m-1}}\} ,$$

is called a normal basis, where $\beta \in \text{GF}(2^m)$ and m is the degree of the generating polynomial. The root of an irreducible AOP has the following property that $\beta^{m+1} = 1$. Furthermore, if the generating polynomial is an AOP and also if 2 is primitive in Z_{m+1} , then we have

$$N = \{\beta, \beta^2, \beta^3, \dots, \beta^m\} . \quad (2.3)$$

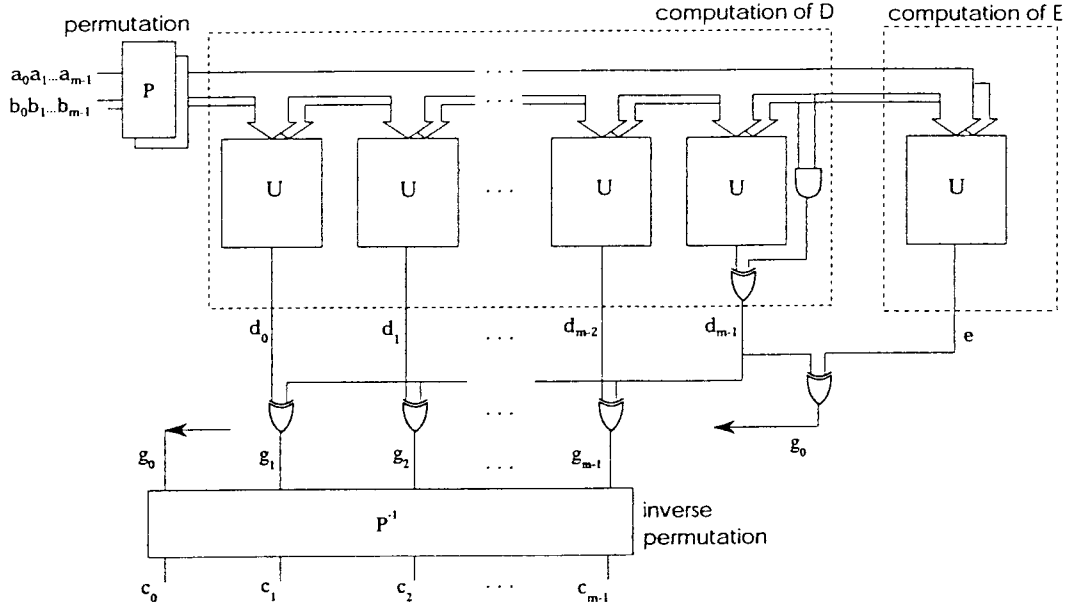
For further information, the reader is referred to [13, page 99]. Since the set (2.3) is also a basis, it can be used to represent the elements of $\text{GF}(2^m)$. This basis is a shifted version of the canonical basis. An element of $\text{GF}(2^m)$ in the normal basis representation can easily be converted to the shifted canonical representation. This is accomplished using a permutation of the binary representation. With the help of the identity $\beta^{m+1} = 1$, we perform the conversion

$$A = \sum_{i=0}^{m-1} a_i \beta^{2^i} = \sum_{i=1}^m a'_i \beta^i ,$$

using the permutation P given as

$$a'_{2^i \bmod (m+1)} = a_i \quad \text{for} \quad i = 0, 1, \dots, m - 1 .$$

FIGURE 2.4: The proposed normal basis multiplier.



In order to perform a normal basis multiplication, we take the inputs A and B represented in the normal basis, convert them to the shifted canonical basis using the permutation P , and then perform a canonical basis multiplication. At the end of this computation, we obtain $F = AB/\beta^2$ represented in the canonical basis as

$$F = f_0 + f_1\beta + f_2\beta^2 + \dots + f_{m-1}\beta^{m-1} .$$

Note that the values f_i are the outputs of the canonical basis multiplier shown in Figure 2.1, and therefore, we have $f_i = d_i + e$ for $i = 0, 1, \dots, m-1$. We then multiply F by β^2 , and obtain $G = F\beta^2$ as

$$G = (d_0 + e)\beta^2 + (d_1 + e)\beta^3 + \dots + (d_{m-1} + e)\beta^{m+1} .$$

We now need to represent this number in the shifted canonical basis. Since

$$\beta^{m+1} = \beta + \beta^2 + \cdots + \beta^m$$

the coefficient $(d_{m-1} + e)$ is added to the coefficients of all the other terms. We can write the final expression as

$$\begin{aligned} G &= (d_{m-1} + e)\beta + (d_0 + e + d_{m-1} + e)\beta^2 + \\ &\quad (d_1 + e + d_{m-1} + e)\beta^3 + \cdots \\ &\quad \cdots + (d_{m-2} + e + d_{m-1} + e)\beta^m \\ &= (d_{m-1} + e)\beta + (d_0 + d_{m-1})\beta^2 + \\ &\quad (d_1 + d_{m-1})\beta^3 + \cdots + (d_{m-2} + d_{m-1})\beta^m, \end{aligned}$$

which gives $g_0 = d_{m-1} + e$ and $g_i = d_{i-1} + d_{m-1}$ for $i = 1, 2, \dots, m-1$. Thus, we have obtained the representation of the number G in the shifted canonical basis. We now apply the inverse of the permutation P to G and obtain the bits of the number C in the normal basis. The architecture of the normal basis multiplier is given in Figure 2.4. It is very similar to that of the canonical basis multiplier.

The implementation of the permutation and inverse permutation operations are accomplished by wiring. Therefore, the normal basis multiplier requires exactly the same number AND and XOR gates as that of the canonical basis multiplier in Figure 2.1. Furthermore, the time complexity of the normal basis multiplier is equal to that of the canonical basis multiplier.

2.4. Conclusions

The time complexity of the proposed canonical basis multiplier is significantly less than previously proposed similar multipliers for the field $\text{GF}(2^m)$ generated by

TABLE 2.1: Comparing canonical basis multipliers with generating AOPs.

	XOR Gates	AND Gates	Delay
Itoh-Tsujii [3]	$m^2 + 2m$	$m^2 + 2m + 1$	$T_A + \lceil \log_2 m + \log_2(m + 2) \rceil T_X$
Hasan-Wang-Bhargava [2]	$m^2 + m - 2$	m^2	$T_A + (m + \lceil \log_2(m - 1) \rceil) T_X$
Proposed design (Figure 2.1.)	$m^2 - 1$	m^2	$T_A + (2 + \lceil \log_2(m - 1) \rceil) T_X$

TABLE 2.2: Comparing normal basis multipliers with generating AOPs.

	XOR Gates	AND Gates	Delay
Massey-Omura [4]	$2m^2 - 2m$	m^2	$T_A + (1 + \lceil \log_2(m - 1) \rceil) T_X$
Hasan-Wang-Bhargava [26]	$m^2 - 1$	m^2	$T_A + (1 + \lceil \log_2(m - 1) \rceil) T_X$
Proposed design (Figure 2.4.)	$m^2 - 1$	m^2	$T_A + (2 + \lceil \log_2(m - 1) \rceil) T_X$

an AOP. The structure of the canonical basis multiplier is very regular: it consists of $m + 1$ identical modules, and some additional XOR and AND gates. It is more regular than the Mastrovito multiplier, and requires significantly less gate delays. The proposed canonical basis multiplier requires m^2 AND gates and $m^2 - 1$ XOR gates. The Mastrovito multiplier requires $m^2 - 1$ XOR gates and m^2 AND gates, and has a delay less than $T_A + 2\lceil \log_2 m \rceil T_X$ if the generating polynomial is a primitive trinomial of the form $x^m + x + 1$ [27, 23]. The XOR and AND complexities of the Mastrovito multiplier for a general trinomial or an AOP are not known. However,

the number of XOR gates for a general trinomial is conjectured to be $\geq m^2 - 1$ in [23].

The normal basis multiplier proposed here and the modified Massey-Omura multiplier [26] require the same number of XOR and AND gates, which is about half of the number of gates required by the Massey-Omura multiplier for the field $\text{GF}(2^m)$ with an AOP. The design proposed in this paper requires only 1 more XOR delay than the modified Massey-Omura multiplier. Nevertheless, it is an alternative design, and is based on an entirely different construction. Another advantage is that it is highly modular. Since the proposed normal basis multiplier is based on a canonical basis multiplier, any advances made in canonical basis multiplication using AOPs can be utilized in this design to further reduce the complexity or timing requirements.

3. FAST FINITE FIELD ARITHMETIC AND ELLIPTIC CURVE CRYPTOGRAPHY OVER COMPOSITE FIELDS

3.1. Introduction

An important category of cryptographic algorithms are the elliptic curve cryptosystems [15, 8, 14, 10] defined over the finite field $GF(2^k)$. Elliptic curve cryptographic applications require fast hardware and software implementations of the arithmetic operations in $GF(2^k)$ for large values of k . Recently, there has been a growing interest to develop software methods for implementing $GF(2^k)$ arithmetic operations and elliptic curve cryptographic operations [19, 1]. In this paper, we are also interested in obtaining efficient software implementations of the arithmetic operations in $GF(2^k)$ and the elliptic curve operations over the field $GF(2^k)$. We are particularly interested in the case where k is a composite number $k = n \cdot m$.

An implementation method for this case was presented in [1], where the authors propose to use the logarithmic table lookup method for the ground field $GF(2^n)$ operations. The field $GF(2^{n \cdot m})$ is then constructed using the polynomial basis, where the elements of $GF(2^{n \cdot m})$ are polynomials of degree $m - 1$ whose coefficients are from the ground field $GF(2^n)$. The field multiplication is performed by first multiplying the input polynomials, and reducing the resulting polynomial by a degree- m irreducible trinomial.

In this paper, we propose a similar methodology for implementing the arithmetic operations in $GF(2^{n \cdot m})$. Our main difference is that we use an optimal normal basis in $GF(2^m)$ to represent the elements of $GF(2^{n \cdot m})$ by taking the ground field as $GF(2^n)$. The resulting field operations, multiplication and squaring are quite efficient, and they do not involve modular reductions. Our implementation results

indicate that the arithmetic operations in the proposed method are slightly faster than those of [1].

This paper shows that the optimal normal bases with the ground field as $GF(2^n)$ for $n > 1$ can be efficiently implemented in software. Another contribution of this paper is the squaring algorithm for $GF(2^{n \cdot m})$, where operations in $GF(2^n)$ are performed using the table lookup method, and an optimal normal basis in $GF(2^m)$ is used to represent the elements of $GF(2^{n \cdot m})$. We also give detailed algorithms for performing field multiplication and inversion operations. The timing results for the finite field and elliptic curve operations in the special case of $k = 176$, $n = 16$, and $m = 11$ are presented.

3.2. Representation of Field Elements in Composite Fields

It is customary to view the finite field $GF(2^k)$ as a k -dimensional vector space defined over the field $GF(2)$. The field over which the vector space defined is generally called the ground field. If we take the ground field as $GF(2)$, then the elements of the k -dimensional vector space are bit strings of length k , i.e., $A = (a_0, a_1, \dots, a_{k-2}, a_{k-1})$, where $A \in GF(2^k)$ and the entries $a_i \in GF(2)$. However, if $n > 1$ divides k , then it is also possible to select the ground field as $GF(2^n)$. If we take $nm = k$, then, effectively we are constructing an m -dimensional vector space over $GF(2^n)$. The elements of the ground field are represented as bit-strings of length n ; the elements of $GF(2^{n \cdot m})$ are represented as

$$A = (A_0, A_1, \dots, A_{m-2}, A_{m-1}) ,$$

where the entries $A_i \in GF(2^n)$. These two representations of $GF(2^{n \cdot m})$, the nm -dimensional vector space over $GF(2)$ and the m -dimensional vector space over

$GF(2^n)$, are equivalent. However, the latter representation is based on n -bit words, and it is more advantageous for implementation on microprocessors, particularly when n is selected properly. For example, $n = 8$ and $n = 16$ have been suggested in [5, 1].

An efficient methodology for representing field elements and performing arithmetic in $GF(2^{n \cdot m})$ has been proposed in [1]. This methodology uses the values of n as 8 or 16. A basis for $GF(2^8)$ or $GF(2^{16})$ can be chosen; however, this is not important since the field arithmetic is performed using the logarithmic table lookup method. An element of $GF(2^{n \cdot m})$ is represented using a degree- $(m - 1)$ polynomial whose coefficients are from the field $GF(2^n)$. The multiplication in $GF(2^{n \cdot m})$ is performed by first multiplying the operands to obtain the twice-sized polynomial, and then reducing the resulting polynomial by a degree- m irreducible polynomial. In general, this degree- m irreducible polynomial needs to have its coefficients from the field $GF(2^n)$, however, it is well-known [12] that an irreducible polynomial over $GF(2)$ of degree m remains irreducible over $GF(2^n)$ if and only if $\gcd(n, m) = 1$. Therefore, one can select an irreducible polynomial of degree m with coefficients from $GF(2)$ rather than $GF(2^n)$ if $\gcd(n, m) = 1$. Furthermore, the use of a special irreducible polynomial is suggested in [1]. This special polynomial is a trinomial of the form $x^m + x^t + 1$, where $t \leq \lfloor m/2 \rfloor$. Such special trinomials are easy to find; the paper [1] lists them for a few values of m .

In this paper, we propose another methodology which is similar to the one proposed in [1] in terms of representing the elements of and performing the arithmetic in $GF(2^{n \cdot m})$. We also propose to select n as 8 or 16. However, in our methodology, we represent the elements of $GF(2^{n \cdot m})$ using an (optimal) normal basis for the field $GF(2^m)$. In this representation, the elements of $GF(2^{n \cdot m})$ are m -dimensional vectors whose entries are in $GF(2^n)$. A degree- m irreducible polynomial is selected

such that the root of this polynomial generates an optimal normal basis [17, 13]. Furthermore, if $\gcd(n, m) = 1$, then the selected degree- m irreducible polynomial will have its coefficients from $GF(2)$ rather than $GF(2^n)$. This selection provides a much simpler multiplication method. Since $\gcd(n, m) = 1$ and $n = 8$ or 16 , we need to have an odd m . This requires that we use an optimal normal basis of type II in the field $GF(2^m)$. For optimal normal bases of type I, m would be even because $m + 1$ is a prime number [13].

The normal bases are thought to be inefficient for composite fields in this setting [1]. The advantages of (optimal) normal bases seem to disappear for $n > 1$, particularly for squaring operation. However, we show that if the ground field operations are computed fast (e.g., using the table lookup method), then the composite field operations can be performed very efficiently in the normal basis. In the following, we present the mathematical and algorithmic details of our methodology.

3.3. Representation and Arithmetic in $GF(2^n)$

The logarithmic table lookup method for performing arithmetic in $GF(2^n)$ for small values of n has been proposed before [5, 1]. A primitive element $g \in GF(2^n)$ is selected to serve as the generator of the field $GF(2^n)$, so that an element A in this field can be written in the form $A = g^i$, where $0 \leq i \leq 2^n - 1$. Then, the powers of the primitive element, g^i 's, are computed for $i = 0, 1, \dots, 2^n - 1$, and 2^n pairs of the form (A, i) are obtained.

We construct two tables sorting these pairs in two different ways: *the log table* sorted with respect to A and *the alog table* sorted with respect to index i . For example, if $i = 5$ and $A = g^5$, then we have $\log[A] = 5$ and $\text{alog}[5] = A$. These tables are then used for performing field multiplications and inversions; they

are particularly very useful in software implementations. In order to perform the multiplication $C = A \cdot B$, we perform the following operations:

$$\begin{aligned} i &:= \log[A] \\ j &:= \log[B] \\ k &:= i + j \pmod{2^n - 1} \\ C &:= \text{alog}[k] \end{aligned}$$

This is due to the fact that $C = A \cdot B = g^i \cdot g^j = g^{i+j \bmod 2^n - 1}$. The squaring of an element A is slightly easier; only two table reads are required for computing $C = A^2$, as illustrated below:

$$\begin{aligned} i &:= \log[A] \\ k &:= 2i \pmod{2^n - 1} \\ C &:= \text{alog}[k] \end{aligned}$$

Similarly, the inversion of an element also requires two table reads:

$$\begin{aligned} i &:= \log[A] \\ k &:= 2^n - 1 - i \\ C &:= \text{alog}[k] \end{aligned}$$

because of the relationship $C = A^{-1} = (g^i)^{-1} = g^{-i} = g^{2^n - 1 - i}$.

When $n = 8$, each one of these tables is of size $2^8 = 256$ bytes. This is very reasonable and fits well into the cache memory of most microprocessors. When $n = 16$, then the table becomes of size 2^{16} words, or 128 kilobytes, which is still reasonable. Higher values of n soon become excessive in terms of memory requirements. In our methodology, we propose to use $n = 8$ or $n = 16$.

3.4. Representation and Arithmetic in $GF(2^{n \cdot m})$

As was mentioned, our approach uses $n = 8$ or $n = 16$, and requires that $\gcd(n, m) = 1$. Furthermore, an optimal normal basis of type II needs to exist for the selected m , which is another restriction. The list of m for which an optimal basis of type II exists is found in [13, page 100]. The first 20 values of m are

$$2, 3, 5, 6, 9, 11, 14, 18, 23, 26, 29, 30, 33, 35, 39, 41, 50, 51, 53, 65 .$$

After we eliminate those m not relatively prime to 8 or 16 (i.e., we keep only the odd m), we obtain

$$3, 5, 9, 11, 23, 29, 33, 35, 39, 41, 51, 53, 65 .$$

This gives the following field sizes for up to 232. As an example, we have imple-

TABLE 3.1: Possible values of n , m , and the field size nm .

nm	24	40	48	72	80	88	144	176	184	232
n	8	8	16	8	16	8	16	16	8	8
m	3	5	3	9	5	11	9	11	23	29

mented $n = 16$ and $m = 11$, and thus, $nm = 176$. In the following sections, we will describe this specific case to further illustrate the properties of the proposed approach.

Once the values of n and m are determined, we need to construct an irreducible polynomial of degree m , generating an optimal normal basis of type II for the field

$GF(2^m)$. A recursive method of construction is given in [13, page 99] as follows:

$$\begin{aligned} f_0(x) &= 1 \\ f_1(x) &= 1 + x \\ f_i(x) &= x f_{i-1}(x) + f_{i-2}(x) \text{ for } i = 2, 3, 4, \dots \end{aligned}$$

We iterate the above formulae to obtain the irreducible polynomial for $m = 11$ as

$$f_{11}(x) = x^{11} + x^{10} + x^8 + x^4 + x^3 + x^2 + 1. \quad (3.1)$$

The above polynomial is defined over the field $GF(2)$. Since the degree of the ground field $n = 16$ is relatively prime to $m = 11$, it is also irreducible over $GF(2^{16})$. Let β be a root of this polynomial then $N = \{\beta, \beta^2, \beta^{2^2}, \dots, \beta^{2^{10}}\}$ is the basis of representation. We use this basis to represent the elements of $GF(2^{16 \cdot 11})$. Each $A \in GF(2^{176})$ is represented as an 11-dimensional vector such that

$$\begin{aligned} A &= (A_0, A_1, A_2, \dots, A_8, A_9, A_{10}) \\ &= A_0\beta + A_1\beta^2 + A_2\beta^{2^2} + \dots + A_8\beta^{2^8} + A_9\beta^{2^9} + A_{10}\beta^{2^{10}}, \end{aligned}$$

and $A_i \in GF(2^{16})$ for $i = 0, 1, 2, \dots, 10$. Therefore, we store each field element as an 11-word long array, where the size of each array element is 16 bits.

We now give the details for performing the arithmetic operations addition, squaring, multiplication, and inversion in the composite field $GF(2^{n \cdot m})$.

3.4.1. Squaring

Let $A \in GF(2^{n \cdot m})$ be represented using an m -dimensional vector as

$$A = (A_0, A_1, \dots, A_{m-2}, A_{m-1})$$

where $A_i \in GF(2^n)$ for $i = 0, 1, \dots, m-1$. The squaring of A is computed using

$$A^2 = \left(\sum_{i=0}^{m-1} A_i \beta^{2^i} \right)^2 = \sum_{i=0}^{m-1} A_i^2 \beta^{2^{i+1}}$$

$$= (A_{m-1}^2, A_0^2, A_1^2, \dots, A_{m-3}^2, A_{m-2}^2) \quad (3.2)$$

since $\beta^{2^m} = \beta$. Therefore, the squaring is performed using only a cyclic shift after each word is squared in the ground field $GF(2^n)$. Since the ground field operations are performed using the logarithmic tables, each word squaring requires a table read followed by an addition modulo $2^n - 1$, and another table read. We then perform a cyclic shift of these m words to obtain A^2 in $GF(2^{n \cdot m})$. The proposed squaring operation is significantly more efficient than that of [1] because it uses table lookup operations and it does not require modular reduction.

3.4.2. Multiplication

The multiplication algorithm is the software analogue of the the Massey-Omura multiplier [18]. In order to obtain the product $C = AB$ in $GF(2^m)$, we need to obtain the multiplication matrix, i.e., the coefficients λ_{ij} in

$$C_r = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \lambda_{ij} a_{i+r} b_{j+r} ,$$

where $a_i, b_i, \lambda_{ij} \in GF(2)$. These λ_{ij} coefficients are obtained by expanding the cross products $\beta^{2^i+2^j}$ in the normal basis, i.e., the linear sum of the basis elements $\{\beta, \beta^2, \beta^4, \dots, \beta^{2^{m-1}}\}$. The number of nonzero λ_{ij} s gives the complexity of the multiplication; when the complexity is exactly $2m - 1$, then the normal basis is said to be optimal [17].

The coefficients λ_{ij} belong to the ground field. In our case, the ground field is $GF(2^n)$, however, the coefficients λ_{ij} are in $GF(2)$ since the irreducible polynomial of degree m is also irreducible over $GF(2^n)$ when $\gcd(n, m) = 1$. This gives a simpler optimal normal basis multiplication algorithm involving $A_i, B_i \in GF(2^n)$ for $i = 0, 1, 2, \dots, m - 1$, where the coefficients λ_{ij} are either 1 or 0. Following the usual method of construction [13], we obtain the coefficient matrix λ_{ij} for the

degree-11 irreducible polynomial given by (3.1). The multiplication matrix yields a formulae for the computation of C_0 as follows:

$$\begin{aligned} C_0 = & A_0 B_1 + A_1(B_0 + B_8) + A_2(B_6 + B_8) + A_3(B_4 + B_5) + \\ & A_4(B_3 + B_9) + A_5(B_3 + B_7) + A_6(B_2 + B_9) + A_7(B_5 + B_{10}) + \\ & A_8(B_1 + B_2) + A_9(B_4 + B_6) + A_{10}(B_7 + B_{10}) \end{aligned} \quad (3.3)$$

We now write the computation of C_0 as a function $F(\cdot)$ as

$$C_0 = F(A_0, A_1, A_2, \dots, A_{10}; B_0, B_1, B_2, \dots, B_{10}) . \quad (3.4)$$

It is well-known that if the ground field is $GF(2)$, then C_r can be computed using the same function for computing C_0 with the r -bit shifted input set:

$$C_r = F(A_r, A_{r+1}, \dots, A_{10}, A_0, A_1, \dots, A_{r-1}; B_r, B_{r+1}, \dots, B_{10}, B_0, B_1, \dots, B_{r-1}) . \quad (3.5)$$

We now prove that this argument is also valid when the ground field is $GF(2^n)$, i.e., C_r can be computed using the function for C_0 with the r -word shifted input set. It follows from the squaring formulae (3.2) in §3.4.1. that the (2^r) th power of C is given as

$$C^{2^r} = (C_r^{2^r}, C_{r+1}^{2^r}, \dots, C_{m-2}^{2^r}, C_{m-1}^{2^r}, C_0^{2^r}, \dots, C_{r-2}^{2^r}, C_{r-1}^{2^r}) ,$$

for $1 \leq r \leq m-1$. Let $C = AB$, then $C^{2^r} = (AB)^{2^r} = A^{2^r} B^{2^r}$, which implies

$$\begin{aligned} (C_r^{2^r}, C_{r+1}^{2^r}, \dots, C_{m-1}^{2^r}, C_0^{2^r}, \dots, C_{r-2}^{2^r}, C_{r-1}^{2^r}) = \\ (A_r^{2^r}, A_{r+1}^{2^r}, \dots, A_{m-1}^{2^r}, A_0^{2^r}, \dots, A_{r-2}^{2^r}, A_{r-1}^{2^r}) (B_r^{2^r}, B_{r+1}^{2^r}, \dots, B_{m-1}^{2^r}, B_0^{2^r}, \dots, B_{r-2}^{2^r}, B_{r-1}^{2^r}) \end{aligned}$$

from which we write the formulae for the leftmost term $C_r^{2^r}$ as

$$C_r^{2^r} = F(A_r^{2^r}, A_{r+1}^{2^r}, \dots, A_{m-1}^{2^r}, A_0^{2^r}, \dots, A_{r-1}^{2^r}; B_r^{2^r}, B_{r+1}^{2^r}, \dots, B_{m-1}^{2^r}, B_0^{2^r}, \dots, B_{r-1}^{2^r}) \quad (3.6)$$

Since the coefficients in $F(x_0, \dots, x_{m-1}; y_0, \dots, y_{m-1})$ are all in $GF(2)$, we have

$$F^2(x_0, \dots, x_{m-1}; y_0, \dots, y_{m-1}) = F(x_0^2, \dots, x_{m-1}^2; y_0^2, \dots, y_{m-1}^2) ,$$

or more generally

$$F^{2^r}(x_0, \dots, x_{m-1}; y_0, \dots, y_{m-1}) = F(x_0^{2^r}, \dots, x_{m-1}^{2^r}; y_0^{2^r}, \dots, y_{m-1}^{2^r}) . \quad (3.7)$$

Therefore, it follows from (3.7) and (3.6) that C_r can be computed using the function for C_0 with the r -word shifted input set as

$$C_r = F(A_r, A_{r+1}, \dots, A_{m-1}, A_0, A_1, \dots, A_{r-1}; B_r, B_{r+1}, \dots, B_{m-1}, B_0, B_1, \dots, B_{r-1}) . \quad (3.8)$$

The multiplication algorithm is implemented in software by writing a routine for the function $F(\cdot)$ as implied by (3.3) and (3.4) and calling it 11 times to compute C_r for $r = 0, 1, \dots, 10$. The resulting function is very simple: It requires 20 additions in $GF(2^{16})$ (i.e., XOR operations with 16-bit operands) and 11 multiplications in $GF(2^{16})$ using the logarithmic tables as described in §3.3.. The word level shift operations are often avoided by writing the computed word in its required location. An advantage of this multiplication algorithm over the one in [1] is that it does not perform modular reduction.

3.4.3. Inversion

The inversion of a field element is critical for high-speed implementation of elliptic curve point operations. Here we use the Itoh and Tsujii method [6, 23] for computing the inverse of an element of the composite field. The inversion algorithm is based on the identity

$$A^{-1} = (A^r)^{-1} A^{r-1} . \quad (3.9)$$

where A is an element of $GF(2^{n \cdot m})$. It is established [11] that if

$$r = \frac{2^{mn} - 1}{2^n - 1} , \quad (3.10)$$

then A^r belongs to the ground field $GF(2^n)$. Hence the inverse can be computed very efficiently in the ground field using the table lookup method. The inversion algorithm uses the identity

$$r - 1 = (2^{nm} - 1)/(2^n - 1) - 1 = 2^n + 2^{2n} + 2^{3n} + \dots + 2^{(m-2)n} + 2^{(m-1)n} .$$

In the particular case of $mn = 176 = 16 \cdot 11$, this is given as

$$r - 1 = (2^{11n} - 1)/(2^n - 1) - 1 = 2^n + 2^{2n} + 2^{3n} + \dots + 2^{9n} + 2^{10n} ,$$

where $n = 16$. This identity implies a method for computing A^{r-1} using the table lookup method for multiplication and squaring operations. The step by step computation of A^{r-1} is illustrated below:

$$\begin{aligned} B &= A^{2^n} \\ C &= B^{2^n} = A^{2^{2n}} \\ D &= A \cdot B = A^{2^n + 2^{2n}} \\ E &= D^{2^{2n}} = A^{2^{3n} + 2^{4n}} \\ F &= D \cdot E = A^{2^n + 2^{2n} + 2^{3n} + 2^{4n}} \\ G &= F^{2^{4n}} = A^{2^{5n} + 2^{6n} + 2^{7n} + 2^{8n}} \\ H &= G \cdot F = A^{2^n + 2^{2n} + 2^{3n} + 2^{4n} + 2^{5n} + 2^{6n} + 2^{7n} + 2^{8n}} \\ J &= D^{2^{8n}} = A^{2^{9n} + 2^{10n}} \\ A^{r-1} &= H \cdot J = A^{2^n + 2^{2n} + 2^{3n} + 2^{4n} + 2^{5n} + 2^{6n} + 2^{7n} + 2^{8n} + 2^{9n} + 2^{10n}} \end{aligned}$$

Therefore, the computation of A^{r-1} requires 4 multiplications and 5 exponentiations. These exponentiations are A^{2^n} , B^{2^n} , $D^{2^{2n}}$, $F^{2^{4n}}$, and $D^{2^{8n}}$. As was explained in §3.4.1. and 3.4.2., the (2^i) th power of an element of $GF(2^{n \cdot m})$ is computed by taking

each vector element (which belongs to $GF(2^n)$) to the power 2^i using the table lookup method, and then by cyclically shifting the 11-word vector i times. If $X \in GF(2^n)$ is a vector element, then

$$X^{2^n} = X^{2^{2n}} = X^{2^{4n}} = X^{2^{8n}} = X ,$$

and therefore, we do not need to compute the powers of the individual vector elements. Thus, an exponentiation of the form $A^{2^{in}}$ is computed by performing ($in \bmod 11$) cyclic word-level shifts on the 11-dimensional vector A . The number of shifts for each the operands above are as follows:

$$\begin{aligned} A^{2^n} & : (1 \cdot 16 \bmod 11) = 5 \\ B^{2^n} & : (1 \cdot 16 \bmod 11) = 5 \\ D^{2^{2n}} & : (2 \cdot 16 \bmod 11) = 10 \\ F^{2^{4n}} & : (4 \cdot 16 \bmod 11) = 9 \\ H^{2^{8n}} & : (8 \cdot 16 \bmod 11) = 7 \end{aligned}$$

In practice, these word-level shifts are not performed. The elements are computed and placed in their required location after a multiplication operation. Therefore, the computation of $A^{r^{-1}}$ essentially requires 4 multiplications in $GF(2^{n \cdot m})$.

The next step is to compute $A^r = A \cdot A^{r^{-1}}$. Since $A^r \in GF(2^n)$, it is sufficient to compute the first word of $A \cdot A^{r^{-1}}$. The remaining words of A^r will be identical. Therefore, this step is not as complicated as a full multiplication in $GF(2^{n \cdot m})$. We then compute the inverse of A^r using the table lookup method since $(A^r)^{-1}$ is also in $GF(2^n)$.

The final step is the computation of $A^{-1} = (A^r)^{-1} \cdot A^{r^{-1}}$. Since $(A^r)^{-1} \in GF(2^n)$ and $A^{r^{-1}} \in GF(2^{n \cdot m})$, we write them as

$$\begin{aligned} (A^r)^{-1} & = (A_0, A_0, A_0, \dots, A_0) \\ A^{r^{-1}} & = (B_0, B_1, B_2, \dots, B_{10}) \end{aligned}$$

We then obtain a simplified version of the multiplication formula (3.3) using these operands. Since all A_0 terms are equal, the product $A^{-1} = (A^r)^{-1} \cdot A^{r-1}$, can be calculated using

$$A^{-1} = (A_0 \cdot B_0, A_0 \cdot B_1, A_0 \cdot B_2, \dots, A_0 \cdot B_{10}) ,$$

where each one of these products $A_0 \cdot B_i$ for $i = 0, 1, \dots, 10$ are computed in $GF(2^n)$ using the table lookup method. Therefore, the final step requires 11 multiplications in $GF(2^n)$. In summary, the computation of A^{-1} in $GF(2^{176})$ requires 4 multiplications in $GF(2^{176})$ and a few more multiplications in $GF(2^{16})$, which are performed using the logarithmic table lookup method.

3.5. Elliptic Curve Operations

The proposed methodology for performing fast arithmetic over the composite fields find its best use in elliptic curve cryptography. Let $y^2 + xy = x^3 + ax^2 + b$ be the elliptic curve equation defined over the field $GF(2^k)$, where a and b are elements of the field with $b \neq 0$. The solutions of this equations in the field $GF(2^k)$ together with a special point O (point at infinity) form an additive group E . Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two distinct points (i.e., $x_1 \neq x_2$) on the curve, neither of which is O . The elliptic curve point addition operation takes these two points P and Q in E , and obtains a third point R in E such that $R = P + Q$. The coordinates of $R = (x_3, y_3)$ are calculated using

$$\begin{aligned} \lambda &= (y_1 + y_2)(x_1 + x_2)^{-1} \\ x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1 \end{aligned} \tag{3.11}$$

This computation requires 1 inversion, 1 squaring, 2 multiplication, and 9 addition operations in $GF(2^k)$. Similarly, the elliptic curve point doubling is defined as the computation of $R = P + P = 2P$. The coordinates of $R = (x_3, y_3)$ are computed using

$$\begin{aligned}\lambda &= x_1 + y_1(x_1)^{-1} \\ x_3 &= \lambda^2 + \lambda + a \\ y_3 &= x_1^2 + (\lambda + 1)x_3\end{aligned}\tag{3.12}$$

The computation of $2P$ is accomplished using 1 inversion, 2 squaring, 2 multiplication, and 5 addition operations in $GF(2^k)$. In our implementation, we did not use the projective coordinate system [14] since the inversion can be performed using about 4 multiplications in $GF(2^{176})$. The projective coordinate system would yield a slower implementation because it performs 15 field multiplications instead of a field inversion in order to compute $P + Q$.

Using the operations of elliptic curve point addition and doubling, one can define eP as the elliptic curve ‘point multiplication’ operation. This can be performed using the well-known binary method of computing exponentiations. This computation can be further accelerated using more advanced exponentiation algorithms: addition chains or addition-subtraction chains [7, 16, 9]. In our implementation, we used the canonical recoding binary exponentiation algorithm, which computes eP by taking P and $-P$ as input. If $P = (x, y)$, then the inverse of P is easily computed using $-P = (x, x + y)$. The canonical recoding binary exponentiation algorithm obtains a signed-digit number f over the digit set $\{0, 1, -1\}$, which is equal to e but contains fewer nonzero digits in the average. It is established that the average number of nonzero digits in f is about $L/3$, where L is the number of bits in e .

Therefore, the canonical recoding point multiplication algorithm computes eP using L elliptic curve doublings and about $L/3$ elliptic curve additions.

3.6. Implementation Results and Conclusions

We have implemented the addition, multiplication, and inversion operations in $GF(2^{176})$, and also the elliptic curve point doubling, addition, and multiplication operations over $GF(2^{176})$. The programs were written in C++ using Microsoft Visual C++ Version 5.0, and executed on a PC with the 300-MHz Pentium II processor, running the operating system Windows NT 4.0. Our timing results are given in the first column of Table 3.2. These timing results are for

- Field Multiplication: $a, b, c \in GF(2^{176})$ such that $c := a \cdot b$
- Field Squaring: $a, c \in GF(2^{176})$ such that $c := a \cdot a$
- Field Inversion: $a, c \in GF(2^{176})$ such that $c := a^{-1}$
- EC Addition: $P, Q, R \in E$ such that $R := P + Q$
- EC Doubling: $P, R \in E$ such that $R := P + P$
- EC Multiplication: $P, R \in E$ and 176-bit integer e such that $R := eP$

The elliptic curve operations are performed using the affine coordinate system in which a point is represented using two field elements $P = (x, y)$. We use the elliptic curve addition and doubling operations defined by the equations (3.11) and (3.12). The field parameters $a, b \in GF(2^{176})$ are selected randomly. The point multiplication algorithm uses the canonical recoding binary method in which the signed-digit recoding of the 176-bit randomly chosen integer e is used. The Hamming

weight of e is $176/2 = 88$ and the Hamming weight of its signed-digit representation is $176/3 \approx 59$.

TABLE 3.2: The timing results for the field and elliptic curve operations.

Operation	Our Method (300-MHz Pentium II)	Reproduced Method [1] (300-MHz Pentium II)	Original Timings [1] (133-MHz Pentium)
Field Multiplication	12 μ sec	15 μ sec	(62.7+1.8) μ sec
Field Squaring	1.5 μ sec	2.5 μ sec	(5.9+1.8) μ sec
Field Inversion	60 μ sec	63 μ sec	160 μ sec
EC Addition	80 μ sec	83 μ sec	306 μ sec
EC Doubling	80 μ sec	85 μ sec	309 μ sec
EC Multiplication	25 msec	30 msec	72 msec

We also include the timing results of [1] for comparison. The original timing results of [1] were obtained on a 133-MHz Pentium. We have re-developed the programs of [1] by investing reasonable efforts to optimize the code. The timing results given in the second column are our reproduction of their method. Since the 300-MHz Pentium II processor is about 2.25 times faster than the 133-MHz Pentium, it seems that our reproduction software of the methods of [1] is about 50 % faster than their implementation. In the third column, we also give their original timings on the 133-MHz Pentium processor. We have obtained all our timings by actual implementation. The results in Table 3.2. shows that the proposed methodology is slightly faster than our reproduction of their method [1], and about 50 % faster than their reported timings, taking into account the speed difference of the processors.

We conclude that the proposed method of using the optimal normal basis in $GF(2^m)$ with the ground field as $GF(2^n)$ provides efficient software implementations of the arithmetic operations in $GF(2^{n \cdot m})$ and elliptic curve cryptographic algorithms over $GF(2^{n \cdot m})$, particularly when the ground field $GF(2^n)$ is selected properly. The values of $n = 8$ or 16 seem suitable since the tables are reasonably sized.

A significant result of this research is that the optimal normal bases would provide efficient software implementations of the arithmetic operations in composite fields, contrary to the (seemingly) commonly held belief that the advantages of the optimal normal bases disappear for $n > 1$.

4. MASTROVITO MULTIPLIER FOR ALL TRINOMIALS

4.1. Introduction

The standard basis multiplication operation in $GF(2^m)$ can be accomplished in two steps: polynomial multiplication and modular reduction. Let $a(x), b(x), c(x) \in GF(2^m)$ and $p(x)$ be the irreducible polynomial generating $GF(2^m)$. In order to compute $c(x) = a(x)b(x) \bmod p(x)$, we first obtain the product polynomial $d(x)$ which is of degree (at most) $2m - 2$ as

$$d(x) = a(x)b(x) = \left(\sum_{i=0}^{m-1} a_i x^i \right) \left(\sum_{i=0}^{m-1} b_i x^i \right). \quad (4.1)$$

The next step is then the reduction operation $c(x) = d(x) \bmod p(x)$ to obtain the $m-1$ degree polynomial $c(x)$. In practice, the multiplication and the reduction steps are often combined for efficiency reasons. An architecture for performing the field multiplication was proposed by Mastrovito [27, 22]. In this method, we represent the computation of $d(x)$ as a matrix-vector product $d = Mb$, where $(2m-1) \times m$ dimensional matrix M consists of the coefficients of the polynomial $a(x)$. We then obtain an $m \times m$ dimensional matrix Z by reducing the matrix M using the generating polynomial $p(x)$. The product $c(x)$ is computed using the matrix-vector product $c = Zb$.

The space complexity of the multiplier for the special generating trinomial $x^m + x + 1$ is shown to be $m^2 - 1$ XOR and m^2 AND gates [27, 22, 23, 24]. Paar [25] conjectured that the space complexity of the Mastrovito multiplier would be the same for all trinomials $x^m + x^n + 1$, where $m - 1 \leq n \leq 1$. In this paper, we describe an architecture for the Mastrovito type multiplier using a general trinomial of the form $x^m + x^n + 1$, and show that the proposed architecture requires $m^2 - 1$

XOR and m^2 AND gates when $n \neq m/2$. However, when m is even and $n = m/2$ there is further reduction: The proposed architecture requires only $m^2 - m/2$ XOR gates.

In the following sections, we give a formulation of the Mastrovito matrix Z , and describe an architecture to compute Z . We show that it is sufficient to compute Z_n (the n th row of Z). The remaining elements can be obtained by rewiring, i.e., without using any gates. We then give an analysis of the multiplier architecture and calculate its space and time complexities.

4.2. The Reduction Process

The entries of the Mastrovito matrix Z are functions of the coefficients of the generating polynomial $p(x)$ and the elements of the original multiplication matrix M , which consists of the coefficients of $a(x)$. The matrix M gives the relationship between the coefficients of $b(x)$ and $d(x)$ in terms of the coefficients of $a(x)$, as

follows:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{m-2} \\ d_{m-1} \\ d_m \\ d_{m+1} \\ \vdots \\ d_{2m-3} \\ d_{2m-2} \end{bmatrix} = \begin{bmatrix} a_0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ a_1 & a_0 & 0 & 0 & & 0 & 0 \\ a_2 & a_1 & a_0 & 0 & \cdots & 0 & 0 \\ \vdots & & \vdots & & \ddots & \vdots & \vdots \\ a_{m-3} & a_{m-4} & a_{m-5} & a_{m-6} & & 0 & 0 \\ a_{m-2} & a_{m-3} & a_{m-4} & a_{m-5} & \cdots & a_0 & 0 \\ a_{m-1} & a_{m-2} & a_{m-3} & a_{m-4} & & a_1 & a_0 \\ 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_2 & a_1 \\ 0 & 0 & a_{m-1} & a_{m-2} & & a_3 & a_2 \\ \vdots & & \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & a_{m-1} & a_{m-2} \\ 0 & 0 & 0 & 0 & \cdots & 0 & a_{m-1} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{m-2} \\ b_{m-1} \end{bmatrix}. \quad (4.2)$$

The product polynomial $d(x)$ contains terms with degrees larger than $m - 1$. These terms need to be reduced using the modulus polynomial $p(x)$ in order to obtain the polynomial representation of the field element in $GF(2^m)$. Since we are considering the generating polynomial $p(x) = x^m + x^n + 1$, we use the identity $x^m = x^n + 1$ to reduce the higher order terms in $d(x)$. A particular term x^i for $i \geq m$ may need to be reduced several times. For example, let $m = 8$ and $p(x) = x^8 + x^6 + 1$. The terms x^8 and x^9 need to be reduced only once: $x^8 = x^6 + 1$ and $x^9 = x^7 + x$. However, the term x^{10} will need two reductions $x^{10} = x^8 + x^2 = x^6 + 1 + x^2$. For a specific basis element, the number of reductions depends solely on the degree of the element and on the order of the middle term of the generating trinomial. The maximum number of reductions are performed on the highest order basis element x^{2m-2} . Let k be the number of reductions required to bring this element to its proper range $[0, m - 1]$. This integer k has the property $2m - 2 - k(m - n) < m$, which implies $k > \frac{m-2}{m-n}$.

Therefore, we have

$$k = \left\lfloor \frac{m-2}{m-n} \right\rfloor + 1 . \quad (4.3)$$

Our objective is to obtain the $m \times m$ matrix Z by systematically reducing the last $m-1$ rows of the $(2m-1) \times m$ matrix M using the generating trinomial $x^m + x^n + 1$. In order to accomplish this task, we define *the reduction array* which is the array of $(m-1)$ rows produced by the reduction of the higher order basis elements $x^m, x^{m+1}, \dots, x^{2m-2}$, as shown in Table 4.1.

TABLE 4.1: The reduction array.

x^m	$= 1$	$+x^n$		
x^{m+1}	$= x$	$+x^{n+1}$		
\vdots	\vdots			
x^{2m-n-1}	$= x^{m-n-1}$	$+x^{m-1}$		
x^{2m-n}	$= x^{m-n}$	$+1$	$+x^n$	
\vdots	\vdots			
$x^{3m-2n-1}$	$= x^{2m-2n-1}$	$+x^{m-n-1}$	$+x^{m-1}$	
x^{3m-2n}	$= x^{2m-2n}$	$+x^{m-n}$	$+1$	$+x^n$
\vdots	\vdots			
$x^{4m-3n-1}$	$= x^{3m-3n-1}$	$+x^{2m-2n-1}$	$+x^{m-n-1}$	$+x^{m-1}$
\vdots	\vdots			
$x^{km-(k-1)n}$	$= x^{(k-1)m-(k-1)n}$	$+x^{(k-2)m-(k-2)n}$	$+ \dots$	$+x^0$
\vdots	\vdots			
x^{2m-2}	$= x^{m-2}$	$+x^{n-2}$	$+ \dots$	$+x^{(k-1)n-(k-2)m-2} + x^{kn-(k-2)m-2}$

The rows defined by the reduction array are added to the rows of M in order to eliminate the last $m - 1$ rows of M . The exponent on the left-hand side provides the index to the source row, which will be added to the rows determined by the exponents on the right-hand side. Initially, we take the first m rows of M as the Z matrix, and use the rows above to add certain rows of M to certain other rows of Z in order to obtain the final Z matrix. Let Z_i and M_j denote the i th and j th rows of the matrices Z and M , respectively. The first reduction is determined by the first row of the reduction array as adding M_m to Z_0 and Z_n , since $x^m = 1 + x^n$. The reduction array is given in Table 4.1.

The rows of the reduction array can be divided into groups consisting of rows with equal number of reductions. Since the number of reductions is k , there are k partitions in the array. Since the degree of a term decreases by $m - n$ after each reduction, the first $m - n$ rows, which have degrees ranging from m to $2m - n - 1$, are reduced only once. Thus, the first $m - n$ rows form the first partition. The next partition is the next set of $m - n$ rows. This continues in the same fashion until the k th partition which will have $m - n$ or fewer rows. We enumerate the partitions in increasing order beginning from the topmost as the 0th partition. In general, the i th partition consists of the rows starting with the term $x^{m+i(m-n)}$ and ending with the term $x^{m+(i+1)(m-n)-1}$.

Also, the columns of the reduction array possesses certain properties. The first column on the right-hand side is special: it is the sequence of increasing powers of x as $1, x, x^2, \dots, x^{m-2}$. The second column contains two sequences: the sequence $x^n, x^{n+1}, \dots, x^{m-1}$ followed by the sequence $1, x^{m-n-1}, x^{m-n}, \dots, x^{n-2}$. The third column is obtained by shifting down the second column $m - n$ positions. The fourth column is obtained by shifting down the third column $m - n$ positions, and so on.

Following the construction method proposed in [21], we decompose the Mas-trovito matrix Z as the sum of two $m \times m$ matrices X and Y , i.e., $Z = X + Y$, where X is the upper m rows of the matrix M . The matrix X is an $m \times m$ Toeplitz matrix, i.e., a matrix whose entries are constant along each diagonal [20]. Furthermore, X is lower triangular. On the other hand, the $m \times m$ matrix Y represents the terms obtained through reduction, and is constructed using the reduction array. We will show that the matrix Y is made of two Toeplitz matrices.

Theorem 1 *The $m \times m$ dimensional matrix Y is partitioned into two Toeplitz matrices. The upper n rows form an $n \times m$ Toeplitz matrix while the lower $m - n$ rows form an $(m - n) \times m$ Toeplitz matrix.*

Proof The first column of the reduction array is the sequence $1, x, \dots, x^{m-2}$ corresponding to the left-hand side $x^m, x^{m+1}, \dots, x^{2m-2}$. This implies that we add the rows $M_m, M_{m+1}, \dots, M_{2m-2}$ to the rows Z_0, Z_1, \dots, Z_{m-2} , respectively. We represent this computation using the $m \times m$ matrix T given as

$$T = \begin{bmatrix} 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_2 & a_1 \\ 0 & 0 & a_{m-1} & a_{m-2} & & a_3 & a_2 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & a_{m-1} & a_{m-2} \\ 0 & 0 & 0 & 0 & \cdots & 0 & a_{m-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}, \quad (4.4)$$

obtained from the matrix M in Equation (4.2). The matrix T which is an $m \times m$ Toeplitz matrix is the initial value of the matrix Y as $Y := T$. After this computation, we need to accumulate the contributions of the remaining columns of the reduction array. We first consider the contribution of the sequence $x^n, x^{n+1}, \dots, x^{m-1}$ which is the starting sequence in the second column of the reduction array (and also

all columns thereafter). This sequence implies that we add the rows $M_m, M_{m+1}, \dots, M_{2m-n-1}$ to the rows $Z_n, Z_{n+1}, \dots, Z_{m-1}$, respectively. This contribution to the matrix Y is represented using the $m \times m$ matrix U as $Y := Y + U$. The matrix U is obtained from the matrix T by shifting down n rows as follows:

$$U = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & a_{m-1} & a_{m-2} & \cdots & a_n & \cdots & a_2 & a_1 \\ 0 & 0 & a_{m-1} & \cdots & a_{n+1} & \cdots & a_3 & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{m-1} & \cdots & a_{m-n+1} & a_{m-n} \end{bmatrix} \begin{matrix} 0 \\ \vdots \\ n-1 \\ n \\ n+1 \\ \vdots \\ m-1 \end{matrix} \quad (4.5)$$

Note that the matrix U is composed of two matrices. Its upper n rows constitute an $n \times m$ zero matrix and its lower $m - n$ rows constitute an $(m - n) \times m$ Toeplitz matrix.

Let $T[\uparrow i]$ represent the matrix T shifted up i rows by feeding i rows of zeros from bottom. Let also $U[\rightarrow i]$ represent the matrix U shifted right i columns by feeding i columns of zeros from left. The contribution of the first column of the reduction array (i.e., the sequence $1, x, x^2, \dots, x^{m-2}$) to the Y matrix is given as $T[\uparrow 0] = T$. The contribution of the second column of the reduction array has two components: the starter sequence $x^n, x^{n+1}, \dots, x^{m-1}$ contributes the U matrix and the remainder sequence $1, x^{m-n-1}, x^{m-n}, \dots, x^{n-2}$ contributes the matrix T shifted up $m - n$ rows, i.e., the matrix $T[\uparrow (m - n)]$. Similarly, the starter sequence in the third column contributes the matrix $U[\rightarrow (m - n)]$, while the remainder the sequence contributes $T[\uparrow 2(m - n)]$. Adding these contributions for $i = 0$ to $k - 1$,

we obtain

$$Y = \sum_{i=0}^{k-1} T[\uparrow i(m-n)] + \sum_{i=0}^{k-1} U[\rightarrow i(m-n)] \quad (4.6)$$

Note that T and thus $T[\uparrow i]$ for all $i \geq 0$ are Toeplitz matrices. Their sum is also a Toeplitz matrix. The first n rows of the matrix U are zero, so are the first n rows of $U[\rightarrow i]$ for $i \geq 0$. Therefore, we conclude that the upper n rows of the matrix Y form an $n \times m$ Toeplitz matrix. Furthermore, the last $m-n$ rows the matrix U form an $(m-n) \times m$ Toeplitz matrix. Similarly, the last $m-n$ rows of all $U[\rightarrow i]$ are $(m-n) \times m$ Toeplitz matrices. Therefore, the last $m-n$ rows of the matrix Y form an $(m-n) \times m$ Toeplitz matrix.

4.3. The Multiplier Architecture

Since X is an $m \times m$ Toeplitz matrix and Y can be partitioned into two Toeplitz matrices, and $Z = X + Y$, we conclude that Z matrix can also be partitioned into two Toeplitz matrices. In other words, the upper n rows and the lower $m-n$ rows form two Toeplitz matrices of dimension $n \times m$ and $(m-n) \times m$, respectively. We will use this fact in the design of our multiplier.

First we make three important observations about the construction of Z_i for $0 \leq i \leq m-1$ and $k \neq n$ using the row Z_n without using any gates:

1. The rows Z_i for $1 \leq i \leq n-1$ can be obtained from Z_0 by rewiring.
2. The rows Z_i for $n+1 \leq i \leq m-1$ can be obtained from Z_n by rewiring.
3. The row Z_0 can be obtained from (an intermediate step of) Z_n by rewiring.

The proof of Property 1 is straightforward. Since the first n rows of Z form an $n \times m$ Toeplitz matrix, each position in the upper triangular region contains

diagonally the same value. We first implement the first row, and then obtain the other values in the upper triangular region of the $n \times m$ matrix by rewiring the values from the first row. On the other hand, the lower triangular part of Y is filled with zeros, and thus, the only contribution to the lower triangular part of Z comes from X , which consists of single terms. Therefore, the input bits will simply be wired to obtain the lower triangular part of Z .

In order to prove Property 2, we note that the last $m-n$ rows of Z form an $(m-n) \times m$ dimensional Toeplitz matrix. The elements in the upper triangular region of this submatrix are diagonally the same, and therefore, they can be obtained from Z_n by rewiring. All the remaining entries (in the lower triangular region) contains single terms coming from X , which are obtained from the inputs by rewiring.

Property 3 is proved as follows: On the right-hand side of the reduction array whenever there is the term 1 in a particular row, there is also the term x^n , which shows that the set of contributions from the reduction array to Z_n covers the set of contributions to Z_0 . The remaining terms come from X . However, X_0 contains all zero entries except the single term a_0 in the leftmost position. Since this position in Y contains a zero, this term is from the input. The other entries of Z_0 are obtained from Z_n .

The complexity of the multiplier solely depends on Z_n , which is explicitly given as

$$Z_n = (a_n \ a_{n-1} \ \cdots \ a_1 \ a_0 \ 0 \ \cdots \ 0) + (0 \ \cdots \ 0 \ a_{m-1} \ \cdots \ a_{n+1}) + \sum_{i=0}^{k-1} M_m[\rightarrow i(m-n)] .$$

The first term (vector) in Equation (4.7) is the n th row of X . The second term (vector) comes from the x^n term in the first column on the right-hand side of the reduction array. The other terms (the terms inside the summation) come from the x^n terms on the top of each column. Let the sum of the first two vectors be denoted

as

$$W = (a_n \ a_{n-1} \ \cdots \ a_1 \ a_0 \ a_{m-1} \ \cdots \ a_{n+1}) ,$$

then, we can write Z_n as

$$Z_n = W + \sum_{i=0}^{k-1} M_m[\rightarrow i(m-n)] . \quad (4.7)$$

The summation (4.7) has important properties which we will use to construct the proposed architecture. In the addition of $W + M_m[\rightarrow 0]$, the element a_i in W and the element a_{i+m-n} in $M_m[\rightarrow 0]$ are aligned for $i = n-1, n-2, \dots, 1$ as

$$\begin{array}{cccc} a_n & \boxed{a_{n-1} \ a_{n-2} \ \cdots \ a_1} & a_0 & a_{m-1} \ \cdots \ a_{n+1} \\ 0 & \boxed{a_{m-1} \ a_{m-2} \ \cdots \ a_{m-n+1}} & a_{m-n} & a_{m-n-1} \ \cdots \ a_1 \end{array}$$

Furthermore, in the addition of $M_m[\rightarrow 0] + M_m[\rightarrow (m-n)]$, the element a_i in $M_m[\rightarrow 0]$ and the element a_{i+m-n} in $M_m[\rightarrow (m-n)]$ are aligned for $i = n-1, n-2, \dots, 1$ as

$$\begin{array}{cccc} 0 & a_{m-1} \ \cdots \ a_n & \boxed{a_{n-1} \ a_{n-2} \ \cdots \ a_1} & \\ 0 & 0 \ \cdots \ 0 & \boxed{a_{m-1} \ a_{m-2} \ \cdots \ a_{m-n+1}} & \end{array}$$

Therefore, the addition of the subvector $(a_{n-1}a_{n-2}\cdots a_1)$ of W to the corresponding part in $M_m[\rightarrow 0]$ is contained in the sum $M_m[\rightarrow 0] + M_m[\rightarrow (m-n)]$. Hence this part of W need not to be separately added. It can be obtained from the summation term in (4.7) by rewiring.

We stack the m -dimensional row vectors $M_m[\rightarrow i(m-n)]$ on top of one another to obtain the $k \times m$ matrix C as

$$C = \begin{bmatrix} M_m[\rightarrow 0] \\ M_m[\rightarrow (m-n)] \\ M_m[\rightarrow 2(m-n)] \\ \vdots \\ M_m[\rightarrow (k-1)(m-n)] \end{bmatrix} . \quad (4.8)$$

The computation of the sum in (4.7) is equivalent to the summation of the *columns* of the matrix C . Let $C_i = (C_{0,i} \ C_{1,i} \ \cdots \ C_{k-1,i})^T$ be the i th column of C indexed from left to right as $i = 0, 1, \dots, m-1$. Since the matrix C is obtained by first writing M_m to the first row, and then shifting this row $(m-n)$ times to the right to obtain the remaining rows, the sum $\sum_{j=0}^{k-1} C_{j,i}$ is fully contained in the sum $\sum_{j=0}^{k-1} C_{j,i+(m-n)}$. Therefore, it suffices to obtain the individual sums of the last $m-n$ rows of the matrix C . The remaining column sums are obtained as byproducts. Also, the first element a_n of W need not be added either since the first column C_0 is zero column; we simply rewire this element from the input.

Furthermore, among the last $m-n$ columns some C_i columns are of length k while some other are of length $k-1$. This is because when M_m is shifted $(k-1)(m-n)$ times to the right, The leftmost side of $M_m[\rightarrow (k-1)(m-n)]$ is filled with zeros; only the last $\alpha = (m-1) - (k-1)(m-n)$ entries will be the individual a_i terms. Therefore, the last α columns are of length k , and the remaining $(m-n) - \alpha$ columns are of length $k-1$.

4.4. Space and Time Complexity

It follows from the analysis in the preceding section that we need to compute the individual sum of the last α columns C_i for $i = m-1-\alpha+1, m-1-\alpha, \dots, m-1$, which are of length k . A single column sum requires $k-1$ XOR gates. All α columns require $\alpha(k-1)$ XOR gates. The remaining $(m-n) - \alpha$ columns are of length $k-1$, which requires $k-2$ XOR gates to obtain each column sum. Therefore, we need $((m-n)-\alpha)(k-2)$ XOR gates to obtain these column sums. Hence, the computation

of the individual sums of the last $m - n$ columns of C requires a total of

$$\alpha(k - 1) + ((m - n) - \alpha)(k - 2) = n - 1$$

XOR gates. The rest of the column sums are obtained from these $m - n$ column sums as byproducts. We then need to add the vector W except its subvector $(a_{n-1}a_{n-2}\cdots a_1)$. Also, the first element a_n of W need not be added; it can be rewired from the input. Since W is of length m , we need $m - (n - 1) - 1$ XOR gates to add the row vector W to the final sum. This gives total number of XOR gates to compute Z_n as

$$n - 1 + m - (n - 1) - 1 = m - 1 .$$

Therefore, the generation of the matrix Z requires a total of $m - 1$ XOR gates. The matrix multiplication $c = Zb$, where b is of dimension $m \times 1$ and Z is of dimension $m \times m$, requires m^2 two-input AND gates and $m(m - 1)$ XOR gates. This gives the total number of AND and XOR gates to obtain the product $c(x) = a(x)b(x) \pmod{x^m + x^n + 1}$ as

$$\# \text{ AND} = m^2$$

$$\# \text{ XOR} = (m - 1) + m(m - 1) = m^2 - 1$$

It is interesting to notice that the space complexity is not a function of n . On the other hand, the time complexity depends on n , as we will show now. The longest signal path in the architecture is defined as the time complexity of the multiplier. We will denote the delay of a 2-input AND and XOR gates by T_A and T_X , respectively. The longest delay occurs in the calculation of the last element Z_n , which requires the sum of the last element of W , and all k elements of the column vector C_{m-1} . Since some of the suffix (or prefix) elements of the summation is needed, we use a length k linear XOR chain to compute this sum, using a total of kT_X delays to compute

$$x^{10} = x^2 + x^7$$

The second group of 3 rows contains 2 reductions for each expression:

$$x^{11} = x^3 + x^8 = x^3 + 1 + x^5 \quad .$$

$$x^{12} = x^4 + x^9 = x^4 + x + x^6$$

$$x^{13} = x^5 + x^{10} = x^5 + x^2 + x^7$$

The last group contains only 1 row with 3 reductions:

$$x^{14} = x^6 + x^{11} = x^6 + x^3 + x^8 = x^6 + x^3 + 1 + x^5$$

The reduction array in its final form is as follows:

$$x^8 = 1 + x^5$$

$$x^9 = x + x^6$$

$$x^{10} = x^2 + x^7$$

$$x^{11} = x^3 + 1 + x^5$$

$$x^{12} = x^4 + x + x^6$$

$$x^{13} = x^5 + x^2 + x^7$$

$$x^{14} = x^6 + x^3 + 1 + x^5$$

In order to obtain the 8×8 matrix Y , we write the expression

$$\begin{aligned} Y &= \sum_{i=0}^2 T[\uparrow i(8-5)] + \sum_{i=0}^2 U[\rightarrow i(8-5)] \\ &= T[\uparrow 0] + T[\uparrow 3] + T[\uparrow 6] + U[\rightarrow 0] + U[\rightarrow 3] + U[\rightarrow 6] \quad . \end{aligned}$$

Similarly, we obtain the 8×8 matrices $U[\rightarrow 3i]$ for $i = 0, 1, 2$ as

$$U = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 \\ 0 & 0 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 \\ 0 & 0 & 0 & a_7 & a_6 & a_5 & a_4 & a_3 \end{bmatrix}, \quad U[\rightarrow 3] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_7 & a_6 & a_5 & a_4 \\ 0 & 0 & 0 & 0 & 0 & a_7 & a_6 & a_5 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_7 & a_6 \end{bmatrix},$$

$$U[\rightarrow 6] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Finally, we obtain the 8×8 matrices X and Y as

$$X = \begin{bmatrix} a_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_1 & a_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_2 & a_1 & a_0 & 0 & 0 & 0 & 0 & 0 \\ a_3 & a_2 & a_1 & a_0 & 0 & 0 & 0 & 0 \\ a_4 & a_3 & a_2 & a_1 & a_0 & 0 & 0 & 0 \\ a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & 0 & 0 \\ a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & 0 \\ a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 \end{bmatrix},$$

$$Y = \begin{bmatrix} 0 & a_7 & a_6 & a_5 & a_4 + a_7 & a_3 + a_6 & a_2 + a_5 & a_1 + a_4 + a_7 \\ 0 & 0 & a_7 & a_6 & a_5 & a_4 + a_7 & a_3 + a_6 & a_2 + a_5 \\ 0 & 0 & 0 & a_7 & a_6 & a_5 & a_4 + a_7 & a_3 + a_6 \\ 0 & 0 & 0 & 0 & a_7 & a_6 & a_5 & a_4 + a_7 \\ 0 & 0 & 0 & 0 & 0 & a_7 & a_6 & a_5 \\ 0 & a_7 & a_6 & a_5 & a_4 + a_7 & a_3 + a_6 & a_2 + a_5 & a_1 + a_4 + a_7 \\ 0 & 0 & a_7 & a_6 & a_5 & a_4 + a_7 & a_3 + a_6 & a_2 + a_5 \\ 0 & 0 & 0 & a_7 & a_6 & a_5 & a_4 + a_7 & a_3 + a_6 \end{bmatrix}$$

We proved that Z_i for $i = 1, 2, 3, 4$ can be obtained from Z_0 and from the input by rewiring, since the computed terms in Z_0 cover all other computed terms as easily seen below:

$$Z_0: a_0 \ a_7 \ a_6 \ a_5 \ a_4 + a_7 \ a_3 + a_6 \ a_2 + a_5 \ a_1 + a_4 + a_7$$

$$Z_1: a_1 \ a_0 \ a_7 \ a_6 \ a_5 \ a_4 + a_7 \ a_3 + a_6 \ a_2 + a_5$$

$$Z_2: a_2 \ a_1 \ a_0 \ a_7 \ a_6 \ a_5 \ a_4 + a_7 \ a_3 + a_6$$

$$Z_3: a_3 \ a_2 \ a_1 \ a_0 \ a_7 \ a_6 \ a_5 \ a_4 + a_7$$

$$Z_4: a_4 \ a_3 \ a_2 \ a_1 \ a_0 \ a_7 \ a_6 \ a_5$$

We also proved that Z_i for $i = 6, 7$ can be obtained from Z_5 and from the input by rewiring, which is seen as

$$Z_5: a_5 \ a_4 + a_7 \ a_3 + a_6 \ a_2 + a_5 \ a_1 + a_4 + a_7 \ a_0 + a_3 + a_6 \ a_7 + a_2 + a_5 \ a_6 + a_1 + a_4 + a_7$$

$$Z_6: a_6 \ a_5 \quad a_4 + a_7 \ a_3 + a_6 \ a_2 + a_5 \quad a_1 + a_4 + a_7 \ a_0 + a_3 + a_6 \ a_7 + a_2 + a_5$$

$$Z_7: a_7 \ a_6 \quad a_5 \quad a_4 + a_7 \ a_3 + a_6 \quad a_2 + a_5 \quad a_1 + a_4 + a_7 \ a_0 + a_3 + a_6$$

Furthermore, we proved that Z_0 can be obtained from an intermediate step of Z_5 by rewiring:

$$Z_0: a_0 \ a_7 \quad a_6 \quad a_5 \quad a_4 + a_7 \quad a_3 + a_6 \quad a_2 + a_5 \quad a_1 + a_4 + a_7$$

$$Z_5: a_5 \ a_4 + a_7 \ a_3 + a_6 \ a_2 + a_5 \ a_1 + \underline{a_4 + a_7} \ a_0 + \underline{a_3 + a_6} \ a_7 + \underline{a_2 + a_5} \ a_6 + \underline{a_1 + a_4 + a_7}$$

In order to illustrate the computation of $Z_n = Z_5$, we write the sum (4.7) by expanding into individual terms:

$$\begin{array}{l} W: a_5 \ \underline{a_4 \ a_3 \ a_2 \ a_1} \ a_0 \ a_7 \ a_6 \\ M_m[\rightarrow 0]: 0 \ a_7 \ a_6 \ a_5 \ \underline{a_4 \ a_3 \ a_2 \ a_1} \\ M_m[\rightarrow 3]: 0 \ 0 \ 0 \ 0 \ a_7 \ a_6 \ a_5 \ a_4 \\ M_m[\rightarrow 6]: 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ a_7 \\ C_0 \ C_1 \ C_2 \ C_3 \ C_4 \ C_5 \ C_6 \ C_7 \end{array}$$

As underlined above, the addition of the subvector $(a_4 \ a_3 \ a_2 \ a_1)$ of W to the corresponding part in $M_m[\rightarrow 0]$ is also present in the sum $M_m[\rightarrow 0] + M_m[\rightarrow 3]$. Hence this part of W does not to be separately computed. Furthermore, we do not need to add a_5 to the final sum vector since the first column C_0 is zero column, and the element a_5 can be rewired from the input.

We need to compute the individual sums of the last $m - n = 3$ columns C_5, C_6, C_7 . Among these 3 columns $\alpha = (m - 1) - (k - 1)(m - n) = 7 - 2(3) = 1$ of them (the column C_7) is of length $k = 3$, while the remaining $m - n - \alpha = 3 - 1 = 2$ of them (the columns C_5 and C_6) are of length $k - 1 = 2$, as easily seen above.

Therefore, the computation of C_7 requires 2 XOR gates while C_5 and C_6 require 1 XOR gate each. The remaining individual column sums are obtained as byproducts. We then add the row W to the final sum vector, except the first element a_5 , and the subvector $(a_4 \ a_3 \ a_2 \ a_1)$, which means, we add the last $m - 1 - (n - 1) = 8 - 1 - 4 = 3$ elements using 3 XOR gates. Therefore, the construction of Z_5 requires a total of 7 XOR gates.

The remaining vectors Z_i for $i \neq 5$ are obtained from Z_5 , as we have shown. What remains is the computation of the matrix-vector product $c = Zb$, which requires $m^2 = 8^2 = 64$ AND gates and $m(m - 1) = 8(8 - 1) = 56$ XOR gates. We therefore, conclude that the computation of $c(x) = a(x)b(x) \pmod{x^8 + x^5 + 1}$ requires 64 AND gates and 63 XOR gates.

4.6. The Special Case of $n = m/2$

In this section, we show that when m is even and n is equal to $m/2$, the Mastrovito multiplier architecture described in this paper further simplifies. It is known [13] that a trinomial of the form $x^m + x^{m/2} + 1$ is irreducible over $GF(2)$ if m is of the form $m = 2 \cdot 3^r$ for some $r \geq 0$. When $n = m/2$, we find the number of reductions k as

$$k = \left\lfloor \frac{m - 2}{m - m/2} \right\rfloor + 1 = \left\lfloor 2 - \frac{4}{m} \right\rfloor + 1 = 2 . \quad (4.10)$$

Since $k = 2$, we write the vector $Z_n = Z_{m/2}$ from (4.7) as

$$Z_n = W + M[\rightarrow 0] + M[\rightarrow m/2] ,$$

which is explicitly given as

$$\begin{aligned} Z_n = Z_{m/2} = & \begin{pmatrix} a_{m/2} & a_{m/2-1} & \cdots & a_1 & a_0 & a_{m-1} & \cdots & a_{m/2+1} \end{pmatrix} + \\ & \begin{pmatrix} 0 & a_{m-1} & \cdots & a_{m/2+1} & a_{m/2} & a_{m/2-1} & \cdots & a_1 \end{pmatrix} + \\ & \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 & a_{m-1} & \cdots & a_{m/2+1} \end{pmatrix} . \end{aligned}$$

We notice that last $m/2 - 1$ elements starting from a_{m-1} and ending with $a_{m/2+1}$ of the vectors W and $M[\rightarrow m/2]$ are exactly the same, and therefore, their sum is equal to zero. We remove these elements from the sum, and obtain

$$\begin{aligned} Z_n = Z_{m/2} = & \begin{pmatrix} a_{m/2} & a_{m/2-1} & \cdots & a_1 & a_0 & 0 & \cdots & 0 \end{pmatrix} + \\ & \begin{pmatrix} 0 & a_{m-1} & \cdots & a_{m/2+1} & a_{m/2} & a_{m/2-1} & \cdots & a_1 \end{pmatrix} + \\ & \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \end{pmatrix} . \end{aligned}$$

In other words, $M_m[\rightarrow m/2]$ makes no contribution, and can be removed from the sum to obtain Z_n . Therefore, the computation of $Z_{m/2}$ requires only the addition of the subvectors

$$(a_{m/2-1} \cdots a_1 a_0) + (a_{m-1} \cdots a_{m/2+1} a_{m/2}) ,$$

which requires only $m/2$ gates. Thus, we conclude that the construction of the vector Z_n in the case $n = m/2$ requires only $m/2$ XOR gates instead of $m - 1$ XOR gates. This brings the total number of XOR gates required to perform the multiplication to $m(m - 1) + m/2 = m^2 - m/2$. The number of AND gates is the same as before.

Furthermore, the time complexity also simplifies since the construction of the vector Z_n now requires a single T_X delay instead of $k T_X$ delay. In the special case of the trinomial $x^m + x^{m/2} + 1$, the time complexity of the proposed architecture is found as

$$T_X + T_A + \lceil \log_2(m - 1) \rceil T_X = T_A + (1 + \lceil \log_2(m - 1) \rceil) T_X . \quad (4.11)$$

We exemplify this case using the irreducible trinomial $x^6 + x^3 + 1$ generating the field $GF(2^6)$ in the following. Since $m = 6$ and $n = 3$, we find the vector $Z_n = Z_3 = W + M[\rightarrow 0] + M[\rightarrow 3]$ as

$$\begin{aligned} Z_3 = & (a_3 \ a_2 \ a_1 \ a_0 \ a_5 \ a_4) + \\ & (0 \ a_5 \ a_4 \ a_3 \ a_2 \ a_1) + \\ & (0 \ 0 \ 0 \ 0 \ a_5 \ a_4) \end{aligned}$$

We remove the subvector $(a_{m-1} \cdots a_{m/2+1}) = (a_5 \ a_4)$ from the vectors W and $M[\rightarrow 3]$, and obtain

$$\begin{aligned} Z_3 = & (a_3 \ a_2 \ a_1 \ a_0 \ 0 \ 0) + \\ & (0 \ a_5 \ a_4 \ a_3 \ a_2 \ a_1) + \\ & (0 \ 0 \ 0 \ 0 \ 0 \ 0) \end{aligned}$$

Hence we can obtain Z_3 using only $m/2 = 3$ XOR gates as

$$Z_3 = (a_3, a_2 + a_5, a_1 + a_4, a_0 + a_3, a_2, a_1)$$

Following the previous analysis, we conclude that the remaining Z_i vectors for $i \neq 3$ can be constructed from Z_3 without using any additional gates. The final reduction array and the required row operations in the 11×6 dimensional matrix M are illustrated below:

$$\begin{aligned} x^6 = 1 + x^3 & \longrightarrow M_0 := M_0 + M_6 \quad \text{and} \quad M_3 := M_3 + M_6 \\ x^7 = x + x^4 & \longrightarrow M_1 := M_1 + M_7 \quad \text{and} \quad M_4 := M_4 + M_7 \\ x^8 = x^2 + x^5 & \longrightarrow M_2 := M_2 + M_8 \quad \text{and} \quad M_5 := M_5 + M_8 \\ x^9 = 1 & \longrightarrow M_0 := M_0 + M_9 \\ x^{10} = x & \longrightarrow M_1 := M_1 + M_{10} \end{aligned}$$

From these row operations, we obtain the final 6×6 dimensional Z matrix as follows:

$$Z = \begin{bmatrix} a_0 & a_5 & a_4 & a_3 & a_2 + a_5 & a_1 + a_4 \\ a_1 & a_0 & a_5 & a_4 & a_3 & a_2 + a_5 \\ a_2 & a_1 & a_0 & a_5 & a_4 & a_3 \\ a_3 & a_2 + a_5 & a_1 + a_4 & a_0 + a_3 & a_2 & a_1 \\ a_4 & a_3 & a_2 + a_5 & a_1 + a_4 & a_0 + a_3 & a_2 \\ a_5 & a_4 & a_3 & a_2 + a_5 & a_1 + a_4 & a_0 + a_3 \end{bmatrix}.$$

As seen in the matrix Z above, it is necessary and sufficient to compute the terms

$$a_2 + a_5, a_1 + a_4, a_0 + a_3$$

in order to construct the entire 6×6 matrix Z . These operations require only 3 XOR gates. In order to perform the multiplication $c(x) = a(x)b(x) \bmod x^6 + x^3 + 1$, we need to perform the matrix vector product $c = Zb$, for which an additional $m(m - 1) = 6(6 - 1) = 30$ XOR gates and $m^2 = 36$ AND gates are required. Therefore, the total number of XOR gates is $3 + 30 = 33$.

4.7. Acknowledgements

The authors would like to thank Professor Christof Paar of Worcester Polytechnic Institute for helpful discussion in relation to this work.

5. A NEW MULTIPLIER FOR OPTIMAL NORMAL BASIS OF TYPE II

5.1. Introduction

The arithmetic operations in the Galois field $GF(2^m)$ have several applications in coding theory, computer algebra, and cryptography [13, 12]. In these applications, time- and area-efficient algorithms and hardware structures are desired for addition, multiplication, squaring, and exponentiation operations. The performance of these operations is closely related to the representation of the field elements. An important advance in this area is the introduction of the Massey-Omura algorithm [18], which is based on the normal basis representation of the field elements. One advantage of the normal basis is that the squaring of an element is computed by a cyclic shift of the binary representation. Efficient algorithms for the multiplication operation in the canonical basis have also been proposed [27, 3, 21]. The space and time complexities of these canonical basis multipliers are much less than those of the Massey-Omura multiplier.

In recent years, efficient normal basis multipliers for special classes of finite fields have been proposed [26, 21]. These multipliers work only for the optimal normal basis of type I. To the best of our knowledge, the Massey-Omura algorithm is the only algorithm which works for the optimal normal basis of type II. However, its parallel space complexity is about twice of these special multipliers. The parallel Massey-Omura algorithm requires $2(m^2 - m)$ XOR gates while both of the special multipliers in [26, 21] require $m^2 - 1$ XOR gates. As enumerated in Table 5.1. of [13], in the range $m \in [2, 2000]$, there are 118 and 218 m values for which an optimal normal basis of type I and type II exist, respectively. In other words, the optimal

normal basis of type II is more likely to occur, and thus, efficient algorithms for this representation are highly desired.

This paper presents a new multiplication algorithm for the field $GF(2^m)$ whose elements are represented using the optimal normal basis of type II. The bit-parallel multiplier proposed in this paper requires 25 % fewer XOR gates than the Massey-Omura multiplier.

5.2. Representation of Field Elements

It is customary to view the field $GF(2^m)$ as an m -dimensional vector space defined over $GF(2)$. In this case, a set of m linearly independent vectors (elements of $GF(2^m)$) are chosen to serve as the basis of representation. If the set of elements $N = \{\beta, \beta^2, \beta^4, \dots, \beta^{2^{m-1}}\}$ forms a basis for some $\beta \in GF(2^m)$, then the basis N is called normal basis and the element β is called normal element. The introduction of the Massey-Omura multiplier [18] was followed by the definition of a special type of normal basis called optimal normal basis. This type of basis minimizes the complexity of the Massey-Omura multiplier. There exists two types of optimal normal basis, as classified in [13]. An optimal normal basis of type II for the field $GF(2^m)$ is constructed by selecting a primitive $(2m + 1)$ th root of unity, i.e., an element γ of $GF(2^m)$ such that $\gamma^{2m+1} = 1$ and furthermore no other power of γ less than $2m + 1$ is equal to 1. Then, $\beta = \gamma + \gamma^{-1}$ serves as the normal element of the basis.

We now show that there exists another basis N' which is obtained by a simple permutation of the basis elements in N , and construct a new parallel multiplication algorithm in this new basis. If 2 is primitive modulo $2m + 1$, then the set of powers

of 2 modulo $2m + 1$

$$S = \{2, 2^2, 2^3, \dots, 2^{2m-1}, 2^{2m}\} \quad (5.1)$$

is equivalent to

$$S' = \{1, 2, 3, 4, \dots, 2m\} . \quad (5.2)$$

Therefore, a basis element of the form $\gamma^{2^i} + \gamma^{-2^i}$ can be written as $\gamma^j + \gamma^{-j}$ for $j \in [1, 2m]$. Furthermore, if $j \geq m + 1$, then it is possible to write $\gamma^j + \gamma^{-j} = \gamma^{(2m+1)-j} + \gamma^{-(2m+1)+j}$ since $\gamma^{2m+1} = 1$. This brings all powers of γ to the range $[1, m]$, which means all basis elements of the form $\gamma^i + \gamma^{-i}$, where $i \in [1, 2m]$, can be written as $\gamma^j + \gamma^{-j}$ with $j \in [1, m]$. Therefore, the bases N and N' given as

$$N = \{\gamma + \gamma^{-1}, \gamma^2 + \gamma^{-2}, \gamma^{2^2} + \gamma^{-2^2}, \dots, \gamma^{2^{(m-1)}} + \gamma^{-2^{(m-1)}}\} \quad (5.3)$$

$$N' = \{\gamma + \gamma^{-1}, \gamma^2 + \gamma^{-2}, \gamma^3 + \gamma^{-3}, \dots, \gamma^m + \gamma^{-m}\} \quad (5.4)$$

are equivalent. The basis N' is obtained from the basis N using a simple permutation. Let A be expressed in the basis N as

$$A = a_1\beta + a_2\beta^2 + a_3\beta^{2^2} + \dots + a_m\beta^{2^{m-1}} , \quad (5.5)$$

where $\beta = \gamma + \gamma^{-1}$. The representation of A in the basis N' given as

$$A = a'_1\alpha_1 + a'_2\alpha_2 + a'_3\alpha_3 + \dots + a'_m\alpha_m , \quad (5.6)$$

where $\alpha_i = \gamma^i + \gamma^{-i}$. There is a permutation between the coefficients $a'_j = a_i$, which is expressed as

$$j = \begin{cases} k & \text{if } k \in [1, m] , \\ (2m + 1) - k & \text{if } k \in [m + 1, 2m] , \end{cases} \quad (5.7)$$

where $k = 2^{i-1} \pmod{2m + 1}$. The basis N' is no longer a normal basis, however, it proves to be useful. We construct a new low-complexity, bit-parallel multiplier in the following section using this new basis.

5.3. New Multiplication Algorithm

We propose a new algorithm for multiplying the elements of $GF(2^m)$ in the basis N as follows:

1. Convert the elements represented in the basis N to the the basis N' using the permutation.
2. Multiply the elements in the basis N' .
3. Convert the result back to the basis N using the inverse permutation.

The first and third steps are implemented without any gates since the permutation operation requires a simple rewiring. The second step is a multiplication operation in the basis N' , which we present below. Let $A, B \in GF(2^m)$ be represented in the basis N' as

$$A = \sum_{i=1}^m a_i(\gamma^i + \gamma^{-i}) \quad \text{and} \quad B = \sum_{i=1}^m b_i(\gamma^i + \gamma^{-i}) . \quad (5.8)$$

The product of these two numbers $C = A \cdot B$ is written as

$$C = A \cdot B = \left(\sum_{i=1}^m a_i(\gamma^i + \gamma^{-i}) \right) \left(\sum_{j=1}^m b_j(\gamma^j + \gamma^{-j}) \right) . \quad (5.9)$$

This product can be transformed to the following form:

$$C = \sum_{i=1}^m \sum_{j=1}^m a_i b_j (\gamma^{i-j} + \gamma^{-(i-j)}) + \sum_{i=1}^m \sum_{j=1}^m a_i b_j (\gamma^{i+j} + \gamma^{-(i+j)}) = C_1 + C_2 . \quad (5.10)$$

For future reference, the two double summations are denoted as C_1 and C_2 as shown above. The term C_1 has the property that the exponent $(i-j)$ of γ is already within the proper range, i.e., $-m \leq (i-j) \leq m$ for all $i, j \in [1, m]$. Furthermore, if $i = j$,

then $\gamma^{i-j} + \gamma^{-(i-j)} = \gamma^0 + \gamma^0 = 0$. Thus, we can write C_1 as

$$C_1 = \sum_{i=1}^m \sum_{j=1}^m a_i b_j (\gamma^{i-j} + \gamma^{-(i-j)}) = \sum_{\substack{1 \leq i, j \leq m \\ i \neq j}} a_i b_j (\gamma^{i-j} + \gamma^{-(i-j)}) . \quad (5.11)$$

If $k = |i - j|$, then the product $a_i b_j$ contributes to the basis element $\alpha_k = \gamma^k + \gamma^{-k}$. For example, the coefficients of α_1 are the sum of all $a_i b_j$ for which $|i - j| = 1$. Table 5.1. shows the elements contributed by the summation C_1 arranged in terms of the order of the basis elements.

TABLE 5.1: The construction of C_1 .

α_1	α_2	$\cdots \alpha_{m-2}$	α_{m-1}	α_m
$a_1 b_2 + a_2 b_1$	$a_1 b_3 + a_3 b_1$	$\cdots a_1 b_{m-1} + a_{m-1} b_1$	$a_1 b_m + a_m b_1$	
$a_2 b_3 + a_3 b_2$	$a_2 b_4 + a_4 b_2$	$\cdots a_2 b_m + a_m b_2$		
\vdots	\vdots			
$a_{m-2} b_{m-1} + a_{m-1} b_{m-2}$	$a_{m-2} b_m + a_m b_{m-2}$			
$a_{m-1} b_m + a_m b_{m-1}$				

Furthermore, the term C_2 is transformed into the following:

$$\begin{aligned} C_2 &= \sum_{i=1}^m \sum_{j=1}^m a_i b_j (\gamma^{i+j} + \gamma^{-(i+j)}) \\ &= \sum_{i=1}^m \sum_{j=1}^{m-i} a_i b_j (\gamma^{i+j} + \gamma^{-(i+j)}) + \sum_{i=1}^m \sum_{j=m-i+1}^m a_i b_j (\gamma^{i+j} + \gamma^{-(i+j)}) \\ &= D_1 + D_2 . \end{aligned} \quad (5.12)$$

The double summations are denoted by D_1 and D_2 , respectively. The exponents of the basis elements $\gamma^{i+j} + \gamma^{-(i+j)}$ in D_1 are guaranteed to be in the proper range $1 \leq (i+j) \leq m$ for $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, m-i$. If $k = i+j$, then product $a_i b_j$ contributes to the basis element α_k as i and j take these values. Table 5.2. shows the construction of the summation D_1 .

TABLE 5.2: The construction of D_1 .

α_1	α_2	α_3	\cdots	α_{m-2}	α_{m-1}	α_m
	$a_1 b_1$	$a_1 b_2$	\cdots	$a_1 b_{m-3}$	$a_1 b_{m-2}$	$a_1 b_{m-1}$
		$a_2 b_1$	\cdots	$a_2 b_{m-4}$	$a_2 b_{m-3}$	$a_2 b_{m-2}$
			\vdots	\vdots	\vdots	
				$a_{m-3} b_1$	$a_{m-3} b_2$	$a_{m-3} b_3$
					$a_{m-2} b_1$	$a_{m-2} b_2$
						$a_{m-1} b_1$

On the other hand, the basis elements of D_2 are all out of range. We use the identity $\gamma^{2m+1} = 1$ to bring them to the proper range:

$$D_2 = \sum_{i=1}^m \sum_{j=m-i+1}^m a_i b_j (\gamma^{i+j} + \gamma^{-(i+j)}) = \sum_{i=1}^m \sum_{j=m-i+1}^m a_i b_j (\gamma^{2m+1-(i+j)} + \gamma^{-(2m+1-(i+j))}) . \quad (5.13)$$

Therefore, if $k = i+j > m$, we replace α_k by α_{2m+1-k} . For example, the term $a_m b_m$ contributes to the basis element α_1 since $2m+1-(m+m) = 1$. Table 5.3. shows the explicit construction of D_2 .

TABLE 5.3: The construction of D_2 .

α_1	α_2	α_3	$\cdots \alpha_{m-2}$	α_{m-1}	α_m
$a_m b_m$	$a_{m-1} b_m$	$a_{m-2} b_m$	$\cdots a_3 b_m$	$a_2 b_m$	$a_1 b_m$
	$a_m b_{m-1}$	$a_{m-1} b_{m-1}$	$\cdots a_4 b_{m-1}$	$a_3 b_{m-1}$	$a_2 b_{m-1}$
		$a_m b_{m-2}$	$\cdots a_5 b_{m-2}$	$a_4 b_{m-2}$	$a_3 b_{m-2}$
			\vdots	\vdots	\vdots
				$a_{m-1} b_4$	$a_{m-2} b_4$
				$a_m b_3$	$a_{m-1} b_3$
					$a_{m-2} b_3$
					$a_m b_2$
					$a_{m-1} b_2$
					$a_m b_1$

The multiplication algorithm in the basis N' constructs C_1 , D_1 , and D_2 , and sums the appropriate terms in order to obtain the product $C = C_1 + D_1 + D_2$. The details of the multiplication operation and its complexity analysis is given in the following section.

5.4. Details of Multiplication and Complexity Analysis

If these three arrays C_1 , D_1 , and D_2 are inspected closely, the following observations can be made:

1. All three arrays are composed of the elements of the form $a_i b_j$ for $i, j \in [1, m]$.
2. The height of the i th column in the array C_1 is $2(m - i)$ for $i = 1, 2, \dots, m$.
This is the number of terms of the form $a_i b_j$ to be summed in the i th column.
3. The height of the i th column in the array D_1 is equal to $i - 1$.
4. The height of the i th column in the array D_2 is equal to i .
5. Therefore, the height of the i th column in the entire array representing the total sum $C = C_1 + D_1 + D_2$ is found as

$$2(m - i) + i - 1 + i = 2m - 1, \quad (5.14)$$

which follows from Observations 2, 3, and 4.

6. If there is an element $a_i b_j$ is present in a column, then the element $a_j b_i$ is also present in the same column. This is true for all of the three arrays C_1 , D_1 , and D_2 .
7. An element of the form $a_i b_i$ is present only once in a column of either D_1 or D_2 .
8. Because of the observations 5, 6, and 7, a column of the entire array representing the total sum C contains a single element of the form $a_i b_i$ and $2m - 2$ elements of the form $a_i b_j$, where $a_j b_i$ is also present.

The proposed multiplication algorithm first computes the terms $a_i b_j$ for $i, j \in [1, m]$ using exactly m^2 two-input AND gates. This requires a single AND gate delay T_A because of the parallelism.

Let $t_{ij} = a_i b_j + a_j b_i$ for $i = 1, 2, \dots, m$ and $j = i + 1, i + 2, \dots, m$. We first compute the terms t_{ij} using

$$(m-1) + (m-2) + \dots + 2 + 1 = \frac{1}{2}m(m-1) \quad (5.15)$$

two-input XOR gates and a single XOR gate delay T_X . The i th column of the entire array contains exactly $\frac{1}{2}(2m-2) = m-1$ terms of the form t_{ij} and also a single element of the form $a_i b_i$. These m numbers are summed using a binary XOR tree, which requires $m-1$ XOR gates and a total delay of $\lceil \log_2 m \rceil T_X$. Due to parallelism, all m columns require $m(m-1)$ XOR gates and the same amount of the delay. Therefore, the construction of the product C requires:

$$\begin{aligned} \# \text{ AND} &= m^2, \\ \# \text{ XOR} &= \frac{1}{2}m(m-1) + m(m-1) = \frac{3}{2}m(m-1), \\ \text{Delay} &= T_A + T_X + \lceil \log_2 m \rceil T_X = T_A + (1 + \lceil \log_2 m \rceil)T_X, \end{aligned}$$

On the other hand, the parallel Massey-Omura algorithm uses m^2 AND gates and $2m(m-1)$ XOR gates, and computes the product in $T_A + (1 + \lceil \log_2(m-1) \rceil)T_X$ gate delays. The new algorithm requires 25 % fewer XOR gates than the Massey-Omura algorithm.

5.5. An Example

In this section, we illustrate the construction of the basis N' and the new multiplication algorithm for the field $GF(2^5)$. Since $2m+1 = 2 \cdot 5 + 1 = 11$ and 2 is primitive in Z_{11} , there exists an optimal basis of type II for the field $GF(2^5)$, which is of the form $N = \{\beta, \beta^2, \beta^4, \beta^8, \beta^{16}\}$, where $\beta = \gamma + \gamma^{-1}$. Using the identity $\gamma^{11} = 1$, we convert this basis to a basis similar to the polynomial basis. The first

three exponents 1, 2, and 4 are in the proper range. We have $8 = 8 \pmod{11}$ and $16 = 5 \pmod{11}$, which brings the exponent 5 to the proper range. In order to bring 8 to the range $[1, m]$, we use $\gamma^8 = \gamma^{8-11} = \gamma^{-3}$. Thus, we can write

$$\begin{aligned}\beta &= \gamma + \gamma^{-1} = \gamma + \gamma^{-1} = \alpha_1 , \\ \beta^2 &= \gamma^2 + \gamma^{-2} = \gamma^2 + \gamma^{-2} = \alpha_2 , \\ \beta^4 &= \gamma^4 + \gamma^{-4} = \gamma^4 + \gamma^{-4} = \alpha_4 , \\ \beta^8 &= \gamma^8 + \gamma^{-8} = \gamma^{-3} + \gamma^3 = \alpha_3 , \\ \beta^{16} &= \gamma^{16} + \gamma^{-16} = \gamma^5 + \gamma^{-5} = \alpha_5 .\end{aligned}$$

which is of the form $N' = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$. The conversions are accomplished using a permutation. Assuming, A is

$$A = a_1\beta + a_2\beta^2 + a_3\beta^4 + a_4\beta^8 + a_5\beta^{16} ,$$

A in N' as $A = a_1\alpha_1 + a_2\alpha_2 + a_3\alpha_3 + a_4\alpha_4 + a_5\alpha_5$. This is $(a'_1, a'_2, a'_3, a'_4, a'_5) = (a_1, a_2, a_4, a_3, a_5)$.

construction of the multiplication circuit. For simplicity, we let A and B in the field $GF(2^5)$ are already given in the basis

$$A = (a_1, a_2, a_3, a_4, a_5) ,$$

$$B = (b_1, b_2, b_3, b_4, b_5) .$$

We are assuming these elements are already converted to the basis N' using the permutation rule above if they were initially given in basis N . The computation of the product $C = (c_1, c_2, c_3, c_4, c_5)$ is accomplished as follows:

- First generate the product terms $a_i b_j$ for $i = 1, 2, 3, 4, 5$ and $j = 1, 2, 3, 4, 5$ using $m^2 = 5^2 = 25$ AND gates. This computation requires a single AND gate delay T_A .

- Then generate the terms $t_{ij} = a_i b_j + a_j b_i$ for $i = 1, 2, 3, 4, 5$ and $j = i + 1, i + 2, \dots, 5$. Thus, we compute

$$t_{12} = a_1 b_2 + a_2 b_1$$

$$t_{13} = a_1 b_3 + a_3 b_1 \quad t_{23} = a_2 b_3 + a_3 b_2$$

$$t_{14} = a_1 b_4 + a_4 b_1 \quad t_{24} = a_2 b_4 + a_4 b_2 \quad t_{34} = a_3 b_4 + a_4 b_3$$

$$t_{15} = a_1 b_5 + a_5 b_1 \quad t_{25} = a_2 b_5 + a_5 b_2 \quad t_{35} = a_3 b_5 + a_5 b_3 \quad t_{45} = a_4 b_5 + a_5 b_4$$

This computation requires $\frac{1}{2}m(m-1) = 10$ XOR gates and a single XOR gate delay T_X .

- Finally the elements of the product are obtained as follows:

$$c_1 = t_{12} + t_{23} + t_{34} + t_{45} + a_5 b_5$$

$$c_2 = t_{13} + t_{24} + t_{35} + t_{45} + a_1 b_1$$

$$c_3 = t_{14} + t_{25} + t_{12} + t_{35} + a_4 b_4$$

$$c_4 = t_{15} + t_{13} + t_{25} + t_{34} + a_2 b_2$$

$$c_5 = t_{14} + t_{23} + t_{15} + t_{24} + a_3 b_3$$

This step requires an additional $m^2 - m = 20$ XOR gates. This computation is accomplished using additional delay of $\lceil \log_2 5 \rceil T_X = 3T_X$.

The result is expressed in the basis N' which can be converted to the basis N using the inverse permutation. The multiplication circuit requires a total of $m^2 = 25$ AND gates and $1.5(m^2 - m) = 30$ XOR gates. The total computation is performed using $T_A + 4T_X$ gate delays. In Table 5.4., we illustrate the construction of the arrays C_1 , D_1 , D_2 , and the final array C .

5.6. Conclusions

We have presented a new multiplier for the field $GF(2^m)$ whose elements are represented using the optimal normal basis of type II. The proposed bit-parallel multiplier requires $1.5(m^2 - m)$ XOR gates while the Massey-Omura multiplier requires $2(m^2 - m)$ XOR gates, which is the only other multiplier working in this basis. The time complexities of these two multipliers are similar: the parallel Massey-Omura multiplier requires $T_A + (1 + \lceil \log_2(m-1) \rceil)T_X$ delays while the delay of the proposed multiplier is $T_A + (1 + \lceil \log_2 m \rceil)T_X$.

TABLE 5.4: The construction of C_1 , D_1 , D_2 , and C in $GF(2^5)$.

α_1	α_2	α_3	α_4	α_5
$a_1b_2 + a_2b_1$ $a_1b_3 + a_3b_1$ $a_1b_4 + a_4b_1$ $a_1b_5 + a_5b_1$ $a_2b_3 + a_3b_2$ $a_2b_4 + a_4b_2$ $a_2b_5 + a_5b_2$ $a_3b_4 + a_4b_3$ $a_3b_5 + a_5b_3$ $a_4b_5 + a_5b_4$				
	a_1b_1	a_1b_2	a_1b_3	a_1b_4
		a_2b_1	a_2b_2	a_2b_3
			a_3b_1	a_3b_2
				a_4b_1
a_5b_5	a_4b_5	a_3b_5	a_2b_5	a_1b_5
	a_5b_4	a_4b_4	a_3b_4	a_2b_4
		a_5b_3	a_4b_3	a_3b_3
			a_5b_2	a_4b_2
				a_5b_1
t_{12}	t_{13}	t_{14}	t_{15}	t_{14}
t_{23}	t_{24}	t_{25}	t_{13}	t_{23}
t_{34}	t_{35}	t_{12}	t_{25}	t_{15}
t_{45}	t_{45}	t_{35}	t_{34}	t_{24}
a_5b_5	a_1b_1	a_4b_4	a_2b_2	a_3b_3

BIBLIOGRAPHY

1. E. De Win, A. Bosselaers, S. Vandenberghe, P. De Gersem, and J. Vandewalle. A fast software implementation for arithmetic operations in $GF(2^n)$. In K. Kim and T. Matsumoto, editors, *Advances in Cryptology — ASIACRYPT 96*, Lecture Notes in Computer Science, No. 1163, pages 65–76. New York, NY: Springer-Verlag, 1996.
2. M. A. Hasan, M. Z. Wang, and V. K. Bhargava. “Modular construction of low complexity parallel multipliers for a class of finite fields $GF(2^m)$,” *IEEE Transactions on Computers*, vol. 41, no. 8, pp. 962–971, August 1992.
3. T. Itoh and S. Tsujii. “Structure of parallel multipliers for a class of finite fields $GF(2^m)$,” *Information and Computation*, vol. 83, pp. 21–40, 1989.
4. C. C. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed. “VLSI architecture for computing multiplications and inverses in $GF(2^m)$,” *IEEE Transactions on Computers*, vol. 34, no. 8, pp. 709–717, August 1985.
5. G. Harper, A. Menezes, and S. Vanstone. Public-key cryptosystems with very small key lengths. In R. A. Rueppel, editor, *Advances in Cryptology — EUROCRYPT 92*, Lecture Notes in Computer Science, No. 658, pages 163–173. New York, NY: Springer-Verlag, 1992.
6. T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. *Information and Computation*, 78(3):171–177, September 1988.
7. D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Reading, MA: Addison-Wesley, Second edition, 1981.
8. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987.
9. N. Koblitz. CM-curves with good cryptographic properties. In J. Feigenbaum, editor, *Advances in Cryptology — CRYPTO 91, Proceedings*, Lecture Notes in Computer Science, No. 576, pages 279–287. New York, NY: Springer-Verlag, 1991.
10. N. Koblitz. *A Course in Number Theory and Cryptography*. New York, NY: Springer-Verlag, Second edition, 1994.
11. R. Lidl and H. Niederreiter. *Finite Fields*. Encyclopedia of Mathematics and its Applications, Volume 20. Reading, MA: Addison-Wesley, 1983.