Frequency Domain Finite Field Arithmetic for Elliptic Curve Cryptography

by

Selçuk Baktır

A Dissertation
Submitted to the Faculty
of the

Worcester Polytechnic Institute

in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy
in

Electrical and Computer Engineering

by

April, 2008

Approved:

Dr. Berk Sunar Dissertation Advisor ECE Department	Dr. Stanley Selkow Dissertation Committee Computer Science Department
Dr. Kaveh Pahlavan Dissertation Committee ECE Department	Dr. Fred J. Looft Department Head ECE Department

April, 2008

To my family;
to my parents Ayşe and Mehmet Baktır,
to my sisters Elif and Zeynep,
and to my brothers Selim and Oğuz

Abstract

Efficient implementation of the *number theoretic transform (NTT)*, also known as the *discrete Fourier transform (DFT)* over a finite field, has been studied actively for decades and found many applications in digital signal processing. In 1971 Schönhage and Strassen proposed an NTT based asymptotically fast multiplication method with the asymptotic complexity $O(m \log m \log \log m)$ for multiplication of m-bit integers or $(m-1)^{st}$ degree polynomials. Schönhage and Strassen's algorithm was known to be the asymptotically fastest multiplication algorithm until Fürer improved upon it in 2007. However, unfortunately, both algorithms bear significant overhead due to the conversions between the time and frequency domains which makes them impractical for small operands, e.g. less than 1000 bits in length as used in many applications. With this work we investigate for the first time the practical application of the NTT, which found applications in digital signal processing, to finite field multiplication with an emphasis on *elliptic curve cryptography (ECC)*. We present efficient parameters for practical application of NTT based finite field multiplication to ECC which requires key and operand sizes as short as 160 bits. With this work, for the first time, the use of NTT based finite field arithmetic is proposed for ECC and shown to be efficient.

We introduce an efficient algorithm, named *DFT modular multiplication*, for computing Montgomery products of polynomials in the frequency domain which facilitates efficient multiplication in $GF(p^m)$. Our algorithm performs the entire modular multiplication, including modular reduction, in the frequency domain, and thus eliminates costly back and forth conversions between the frequency and time domains. We show that, especially in computationally constrained platforms, multiplication of finite field elements may be achieved more efficiently in the frequency domain than in the time domain for operand sizes relevant to ECC.

This work presents the first hardware implementation of a frequency domain multiplier suitable for ECC and the first hardware implementation of ECC in the frequency domain.

We introduce a novel area/time efficient ECC processor architecture which performs all finite field arithmetic operations in the frequency domain utilizing *DFT modular multiplication* over a class of *Optimal Extension Fields (OEF)*. The proposed architecture achieves extension field modular multiplication in the frequency domain with only a *linear* number of base field $GF(p)$ multiplications in addition to a quadratic number of simpler operations such as addition and bitwise rotation. With its low area and high speed, the proposed architecture is well suited for ECC in small device environments such as smart cards and wireless sensor networks nodes.

Finally, we propose an adaptation of the *Itoh-Tsujii algorithm* to the frequency domain which can achieve efficient inversion in a class of OEFs relevant to ECC. This is the first time a frequency domain finite field inversion algorithm is proposed for ECC and we believe our algorithm will be well suited for efficient constrained hardware implementations of ECC in affine coordinates.

Acknowledgements

This dissertation describes the research I conducted for my Ph.D. at Worcester Polytechnic Institute during which I was supported by many people I would like to thank.

Firstly, I am indebted to my advisor Prof. Berk Sunar for his advising and support throughout my graduate studies. I would like to thank him for helping make my Ph.D. a very rich and enjoyable experience. I am also honored to have Prof. Stanley Selkow and Prof. Kaveh Pahlavan in my dissertation committee, and would like to thank them for all their valuable time and suggestions, and for generously sharing their wisdom with me, which significantly improved the quality of my dissertation as well as my Ph.D. experience. I would like to thank Prof. Selkow for always being supportive and encouraging, and Prof. Pahlavan for all his help including being my *honorary advisor* during my advisor's absence on sabbatical in the 2006/2007 academic year.

I would like to thank every person, with whom I have discussed the ideas presented in this dissertation, as well as the anonymous reviewers of the papers [68, 66, 69, 64, 70], whose comments and suggestions helped improve the quality of this dissertation. Any remaining errors are my own. I would also like to acknowledge the National Science Foundation for supporting this work through the grants no. NSF-ANI-0112829 and NSF-ANI-0133297.

While pursuing my Ph.D. I have been very fortunate to have opportunities to visit other research groups and work with many bright and interesting people which helped make my Ph.D. an exciting and nurturing experience.

I would like to thank Prof. Christof Paar for all his help and for giving me the invaluable opportunity to visit his research group at Ruhr-University, Bochum, Germany, during the summers of 2003 and 2005 which resulted in enjoyable and fruitful collaborations. I would like to thank my collaborators in Prof. Paar's group Thomas Wollinger, Jan Pelzl and Sandeep Kumar for all our great work as well as for their friendship and the fun time together. Our collaboration with Jan and Thomas on *Optimal Tower Fields for Hyperelliptic Curve Cryptosystems* resulted in [65], and with Sandeep Kumar on *Elliptic Curve Cryptography in the Frequency Domain* resulted in [64]. In Prof. Paar's research group, I would also like to thank the team assistant Irmgard Kühn and other fellow researchers HoWon Kim, Marco Macchetti, Andy Rupp, Axel Poschmann, Marko Wolf, Dario Carluccio, Kai Schramm and André Weimerskirch for their hospitality and helping make my stay in Bochum an enjoyable one.

I am grateful for having had the opportunity to do an internship in the Internet Security Group at IBM T. J. Watson Research Center in Hawthorne, NY, and be acquainted with many bright and nice people during the summer of 2006. I would like to express my gratitude to my mentor Dr. Pankaj Rohatgi for giving me the invaluable opportunity to work with him and for all his help and guidance. I would like to thank Dr. Rohatgi and my other co-author Dr. Dakshi Agrawal for the great work on *Trojan Detection Using IC Fingerprinting* which resulted in [6]. I am also thankful to the other members of the Internet Security Group Dr. Pau-Chen Cheng, Dr. Suresh Chari, Dr. Charanjit Jutla and Dr. Josyula R. Rao for their hospitality, valuable suggestions and intellectually stimulating conversations.

I have always been interested in learning new things in diverse fields. During my Ph.D. I was fortunate to have the opportunity to conduct industrial research on lossless data compression at Intel Corporation, Hudson, MA, which exposed me to new research problems and helped broaden my perspective. I would like to thank my manager Paul Posco for giving me the opportunity to work for him and his guidance. I would also like to thank Vinodh Gopal for presenting me with some challenging problems in data compression, which gave me the opportunity to have a fun and productive internship and write seven patent documents. It was a satisfying experience to see the potential of my ideas to touch people's lives by going into products. I would also like to thank the other members of our group Christopher F. Clark, Wajdi K. Feghali, Robert P. Ottavi, Prashant Paliwal and Gilbert Wolrich for their company.

I would like to thank all my teachers and professors, from WPI and earlier, who helped keep me inspired and motivated to pursue the highest academic degree. I would like to thank the electrical & computer engineering department head Prof. Fred Looft and the administrative staff Catherine Emmerton, Brenda McDonald and Colleen Sweeney for all their help and creating a friendly atmosphere in our department. I would also like to thank the former members of the CRIS laboratory Gunnar Gaubatz, Colleen Marie O'Rourke, Adam Elbirt, Jens Peter Kaps, Kaan Yüksel, Shiwangi Despande and Serdar Pehlivanoglu, and the present members Kahraman Akdemir, Berk Birand, Ghaith Hammouri, Yin Hu, Deniz Karakoyunlu and Erdinç Öztürk for their friendship.

Life is always more meaningful when there are loving and caring people around you. I would like to thank all friends whose presence helped improve the quality of my life during my Ph.D.

Finally, and most importantly, I would like to express my deepest gratitude to my dear parents Ayşe and Mehmet Baktır, my dear sisters Elif and Zeynep, and my dear brothers Selim and Oğuz. I learned a lot from them, and they have had the greatest influence on me in good ways. Without their unconditional love and support, this work would not have been possible. I would especially like to thank my dear father Mehmet Baktır who has encouraged me the most in my pursuit of having a Ph.D.

Worcester, Massachusetts Selçuk Baktır
April 2008

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Background

The use of elliptic curves in *public key cryptography* was first proposed independently by Koblitz [35] and Miller [55] in 1980s. Since then, *elliptic curve cryptography (ECC)* has been the focus of a lot of attention and gained great popularity due to the current best security estimates indicating that ECC provides the same level of security with much smaller key sizes compared with conventional public key cryptosystems [44, 45].

The standard protocols in cryptography that utilize the discrete logarithm problem have analogues in ECC. The standard discrete logarithm problem has sub-exponential complexity, e.g. using a general number sieve method a discrete logarithm problem in $GF(q)*$ can be solved in sub-exponential time [54]. Whereas, a discrete logarithm on an elliptic curve $E(GF(q))$ has exponential complexity in the size $n = \lceil \log_2 q \rceil$ of the field elements, e.g. using the Pollard's Rho method, one of the best methods for solving discrete logarithm problem on elliptic curves, one can solve the discrete logarithm problem only in time $O(2^{\frac{n}{2}})$ [15].

Elliptic curve cryptosystems are computationally more efficient and offer better security with smaller key sizes compared with traditional public key cryptosystems such as RSA [73] and discrete logarithm based systems such as ElGamal [27] and Diffie-Hellman [24] algorithms. This makes them a better choice especially for constrained environments such as smart cards and wireless devices where resources such as power, processing time and memory are limited.

1.2 Motivation

ECC relies on efficient algorithms for finite field arithmetic operations such as addition, multiplication and inversion. *Optimal Extension Fields (OEF)* [9, 8] have been found to be successful in ECC implementations where resources such as computational power and memory are constrained [84, 41]. The arithmetic operations in OEFs are much more efficient than in characteristic two extensions or prime fields due to the use of a large characteristic base field and the selection of a binomial as the field polynomial.

In the elliptic curve scalar point multiplication, a large number of field multiplications are computed. Multiplication in $GF(p^m)$ is an expensive operation and normally achieved with a quadratic number of multiplications and additions in the base field $GF(p)$ using the *classical schoolbook method* for polynomial multiplication. Integer multiplication is inherently much more complex than other integer operations such as addition, and it is usually slower and consumes more power in hardware. This poses a significant problem in constrained environments such as wireless sensor network nodes and radio frequency identification tags where computational power is quite limited and the requirement for a large number of complex operations is not preferred. The *Karatsuba algorithm* [34] reduces the complexity of multiplication in $GF(p^m)$ by requiring only a subquadratic number of multiplications in $GF(p)$ in exchange for an increased number of $GF(p)$ additions. However, the Karatsuba algorithm is not desirable due to its recursive nature and implementation complexity.

In this dissertation, we address this issue by investigating the practical application of the *number theoretic transform (NTT)* based frequency domain multiplication techniques for operand sizes, as small as 160-bits in length, relevant to ECC. For decades, efficient implementation of the NTT, also known as the *discrete Fourier transform (DFT)* over a finite field, has been an active research venue and found many practical applications in digital signal processing [72, 71, 2, 3, 4, 48, 58, 1, 20, 52, 10, 40, 57, 14, 76, 46, 42, 85, 39, 25, 50, 49, 79] and coding theory [81, 17, 39, 51]. Furthermore, the NTT is known to be very efficient for performing large integer arithmetic. The NTT based multiplication method originally proposed in 1971 by Schönhage and Strassen [75] for integer multiplication provides an efficient method with the asymptotic complexity $O(m \log m \log\log m)$, for multiplication of m-bit integers or $(m-1)^{st}$ degree polynomials [23]. This algorithm was long known to be asymptotically the fastest, until Führer improved upon it in 2007 [28]. However, unfortunately, both methods bear significant overhead due to the conversions between the

time and frequency domains which makes them impractical for small operands, e.g. less than 1000 bits in length as used in many applications. To our knowledge, until our work no implementation of a cryptosystem had yet been achieved using an NTT based frequency domain multiplication algorithm. With this dissertation, we investigate for the first time the practical application of the NTT to a public key cryptosystem, namely ECC and present efficient parameters for practical application of NTT based finite field multiplication to the implementation of ECC in constrained environments. This is the first time the use of NTT based finite field arithmetic is proposed for ECC and shown to be efficient with both theoretical and hardware implementation results.

1.3 Contribution of the Dissertation

1. The NTT based methods have found many practical applications mostly in digital signal processing. The asymptotically fastest multiplication algorithms [75, 28] for integer or polynomial multiplication are also known to be based on the NTT. However, unfortunately, these methods bear significant overhead due to the conversions between the time and frequency domains which makes them impractical for small operands, e.g. less than 1000 bits in length, as used in many applications. In this dissertation, for the first time, we investigate the application of NTT based multiplication to finite fields for implementation of ECC in constrained environments with operand sizes as small as 160 bits in length, and present practical parameters for its efficient application. This work is published in [68, 66].

2. We introduce an efficient algorithm, named *DFT modular multiplication*, for computing Montgomery products of polynomials in the frequency domain. Our algorithm performs the entire modular multiplication in the frequency domain. It achieves multiplication in $GF(p^m)$ with only a linear number of base field $GF(p)$ multiplications in addition to a quadratic number of simpler base field operations such as additions/subtractions and bitwise rotations. We show that, especially in computationally constrained platforms, multiplication of finite field elements may be achieved more efficiently in the frequency domain than in the time domain for operand sizes relevant to ECC. This work is published in [69, 66].

3. We present the first hardware implementation of a frequency domain multiplier suitable for ECC and the first hardware implementation of ECC in the frequency domain. We present a novel area/time efficient ECC processor architecture which utilizes DFT modular multiplication and performs all finite field arithmetic operations in the frequency domain in a class of OEFs $GF(p^m)$. Our architecture achieves areas between 25k and 50k equivalent gates for implementations of ECC over OEFs of size 169, 289 and 361 bits. With its low area and high speed, the proposed architecture would be well suited for ECC in small device environments such as wireless sensor networks. This work is published in [64].

4. We propose an adaptation of the *Itoh-Tsujii algorithm* to the frequency domain which can achieve OEF inversion with only a single inversion, $O(m \log m)$ multiplications and constant multiplications, $O(m^2 \log m)$ additions and $O(m^2 \log m)$ fixed bitwise rotations in the base field $GF(p)$ for a class of OEFs $GF(p^m)$. To the best of our knowledge, this is the first time a frequency domain finite field inversion algorithm is proposed for ECC. With its low computational complexity, the proposed algorithm would be well suited especially for efficient low-power hardware implementation of ECC using affine coordinates in constrained small devices. This work is published in [70].

1.4 Outline of the Dissertation

Chapter 2 reviews OEFs and their arithmetic. It also presents a brief overview of elliptic curves over OEFs, ECC and the NTT. Furthermore, it explains the NTT for representing OEF elements in the frequency domain.

Chapter 3 gives an overview of multiplication in $GF(p^m)$ using the NTT. It establishes the relationship between the NTT and the *residue number system (RNS)*, and proves that multiplication in $GF(p^m)$ using the NTT is equivalent to an optimal case of multiplication in $GF(p^m)$ using the RNS.

In Chapter 4 special NTTs are investigated and efficient parameters are presented for their application to finite field multiplication with small operands, e.g. as small as 160 bits in length, relevant to ECC.

In Chapter 5, a new algorithm, named *DFT modular multiplication*, is introduced. It is shown that DFT modular multiplication improves upon the straightforward NTT based

approach presented in Chapter 3 by performing an entire modular multiplication, including modular reduction, in the frequency domain. It is shown that in constrained platforms multiplication of finite field elements may be achieved more efficiently in the frequency domain, using DFT modular multiplication, than in the time domain even for small operands relevant to ECC.

In Chapter 6, an efficient hardware architecture for the DFT modular multiplication algorithm is proposed and an efficient elliptic curve cryptographic processor architecture is presented which utilizes the proposed multiplier and runs completely in the frequency domain.

Finally, in Chapter 7 an adaptation of the *Itoh-Tsujii algorithm* is proposed for the frequency domain. The proposed algorithm achieves efficient inversion in the frequency domain in a class of OEFs relevant to ECC.

Chapter 2

Background

2.1 Introduction

In this chapter we present some background information on finite fields and the optimal extension field representation. We describe optimal extension field arithmetic and give a brief overview on the use of elliptic curves for public key cryptography, with an emphasis on curves defined over optimal extension fields. Finally, we describe the representation of optimal extension field elements in the frequency domain using the number theoretic transform.

2.2 Finite Fields and Polynomial Representation

A field with a finite number of elements is called a *finite field* or *Galois field*, denoted by $\mathbb{F}_q$ or $GF(q)$, where q stands for the number of elements in the field [47]. The number of elements in a finite field is always a prime or a prime power, i.e., $q = p$ or $q = p^m$,where the prime number p is called the characteristic of the finite field. When q is a prime, i.e. $q = p$, the finite field $GF(p)$ is called a *prime field*. The prime field $GF(p)$ is the field of residue classes modulo p and its elements are represented by the integers in $\{0, 1, 2, \ldots, p-1\}$. When q is a prime power, i.e. $q = p^m$, the finite field $GF(p^m)$ is called an *extension field*. The extension field $GF(p^m)$ is generated by using an m^{th} degree irreducible polynomial over $GF(p)$ and it is the field of residue classes modulo the irreducible field generating polynomial. Hence, in polynomial representation the elements of $GF(p^m)$ are represented by polynomials of degree at most $m-1$ with coefficients in $GF(p)$.

2.2.1 Optimal Extension Fields and their Arithmetic

Optimal extension fields (OEF), introduced by Bailey and Paar in [8, 9], are a special class of finite extension fields which use a field generating polynomial of the form $f(x) = x^m - w$ and have a *pseudo-Mersenne prime* field characteristic given in the form $p = 2^n \pm c$ with $\log_2 c < \lfloor \frac{n}{2} \rfloor$. Theorem 1 provides a simple means to identify irreducible binomials that can be used in OEF construction.

Theorem 1 *[47] Let $m \geq 2$ be an integer and $w \in GF(p)^*$. Then the binomial $x^m - w$ is irreducible in $GF(p)[x]$ if and only if the following three conditions are satisfied:*

1. *each prime factor of m divides the order e of w in $GF(p)^*$;*
2. *the prime factors of m do not divide $\frac{p-1}{e}$;*
3. *$p = 1 \bmod 4$ if $m = 0 \bmod 4$.*

In OEFs the pseudo-Mersenne prime field characteristic allows efficient reduction in the base field $GF(p)$ operations and the binary field generating polynomial allows for efficient reduction in the extension field. OEFs are found to be successful in elliptic curve cryptography implementations where resources such as computational power and memory are constrained [84, 41]. In OEFs, the *standard basis* is utilized for representing finite field elements. An OEF element $A \in GF(p^m)$ is represented in standard basis by a polynomial of degree at most $m - 1$ as

$$A = \sum_{i=0}^{m-1} a_i x^i = a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-1} x^{m-1},$$

where $a_i \in GF(p)$ for $0 \leq i \leq m - 1$. OEF arithmetic is performed as follows.

Addition/Subtraction:

The addition/subtraction of two field elements $A, B \in GF(p^m)$ is performed by adding/subtracting the polynomial coefficients in $GF(p)$ as follows:

$$A \pm B = \sum_{i=0}^{m-1} a_i x^i \pm \sum_{i=0}^{m-1} b_i x^i = \sum_{i=0}^{m-1} (a_i \pm b_i) x^i$$

Multiplication:

For $A, B \in GF(p^m)$, the product $C = A \cdot B$ is computed in two steps:

1. Polynomial multiplication:
$$C' = A \cdot B = \sum_{i=0}^{2m-2} c'_i x^i \tag{2.1}$$

2. Modular reduction:

$$\begin{aligned} C &= C' \bmod f(x) && (2.2)\\ &= \sum_{i=0}^{2m-2} c'_i x^i \bmod x^m - w \\ &= \sum_{i=0}^{m-1} (c'_i + w c'_{i+m}) x^i \ , && (2.3) \end{aligned}$$

where $c'_{2m-1} = 0$.

In the first step the ordinary product of two polynomials is obtained by computing m^2 coefficient multiplications and $(m-1)^2$ coefficient additions. In the reduction step the binomial $f(x) = x^m - w$ facilitates efficient reduction which may be realized by using only $m-1$ constant coefficient multiplications with w and $m-1$ additions.

Inversion:

An elegant method for inversion was introduced by Itoh and Tsujii [33]. For $A \in GF(p^m)$, where $A \neq 0$, A^{-1} is computed in four steps as follows

1. Compute the exponentiation A^{e-1} in $GF(p^m)$, where $e = \frac{p^m-1}{p-1}$;

2. Compute the product $A^e = (A^{e-1}) \cdot A$;

3. Compute the inversion $(A^e)^{-1}$ in $GF(p)$;

4. Compute the product $A^{e-1} \cdot (A^e)^{-1} = A^{-1}$.

For the particular choice of

$$e = \frac{p^m - 1}{p - 1} \ ,$$

A^e belongs to the base field $GF(p)$ [47]. This allows the inversion in Step 3 to be computed in $GF(p)$ instead of the larger field $GF(p^m)$. For the exponentiation A^{e-1} in Step 1, the exponent $e - 1$ can be expanded as follows

$$e - 1 = \frac{p^m - 1}{p - 1} - 1 = p^{m-1} + p^{m-2} + \ldots + p^2 + p \ .$$

This exponentiation is computed by finding the powers A^{p^i}. The original Itoh-Tsujii algorithm proposes to use a normal basis representation over $GF(2)$ which turns the p^i-th power exponentiations into simple bitwise rotations. In [29] this technique was adapted by Guajardo and Paar to work efficiently in standard basis and it was shown that A^{e-1} can be computed by performing at most $\lfloor \log_2(m-1) \rfloor + HW(m-1) - 1$ multiplications and $\lfloor \log_2(m-1) \rfloor + HW(m-1)$ p^i-th power exponentiations in $GF(p^m)$, where $HW(m)$ denotes the hamming-weight of m. A^{p^i} is the i-th iterate of the *Frobenius map* where a single iterate is defined as $\sigma(A) = A^p$. Using the properties $\sigma(A + B) = \sigma(A) + \sigma(B)$ for any $A, B \in GF(p^m)$ and $\sigma(a) = a^p = a$ for any $a \in GF(p)$, the exponentiation $A^{p^i} = \sigma^i(A)$ can be simplified as

$$A^{p^i} = \left(\sum_{j=0}^{m-1} a_j x^j \right)^{p^i} = \sum_{j=0}^{m-1} (a_j x^j)^{p^i} = \sum_{j=0}^{m-1} a_j x^{jp^i} \ . \tag{2.4}$$

Theorem 2 shows that A^{p^i} can be computed by a simple scaled permutation of the coefficients in the polynomial representation of A.

Theorem 2 *[63, 67] For an irreducible binomial $f(x) = x^m - w$ defined over $GF(p)$, the following identity holds for an arbitrary positive integer i and $A \in GF(p^m)$,*

$$A^{p^i} = \left(\sum_{j=0}^{m-1} a_j x^j \right)^{p^i} = \sum_{j=0}^{m-1} (a_j \ c_{s_j}) x^{s_j}$$

where $s_j = jp^i \bmod m$ and $c_{s_j} = w^{\frac{jp^i - s_j}{m}}$. Furthermore, the s_j values are distinct for $0 \leq j \leq m - 1$.

Using the method in Theorem 2, exponentiations of degree p^i may be achieved with the help of a lookup table of precomputed c_{s_j} values, using not more than $m - 1$ constant coefficient multiplications. When m is prime, Corollary 1 [63, 67] further simplifies this computation

by showing that $s_j = jp^i \bmod m$ in Theorem 2 equals j and hence no permutations occur for the coefficients of A.

Corollary 1 *[63, 67] If $f(x) = x^m - w$ is irreducible over $GF(p)$, m is prime, $x^j \in GF(p)[x]$ and i is an arbitrary positive rational integer, then $(x^j)^{p^i} \equiv w^t x^j \pmod{f(x)}$, where $t = \frac{jp^i - j}{m}$.*

Proof of Corollary 1 We need to prove that $jp^i \bmod m = j$, or in other words $m | jp^i - j$. Since $m|(p-1)$ is a necessary condition for the existence of the irreducible binomial $f(x) = x^m - w$ over $GF(p)$ for a prime m (see the first condition in Theorem 1), m also divides $jp^i - j = j(p^i - 1) = j(p-1)(p^{i-1} + p^{i-2} + \cdots + p + 1)$. Hence, the proof is complete. □

2.3 Elliptic Curve Cryptography

In this section we present a brief overview on elliptic curves, with particular interest in curves over OEFs, and on *elliptic curve cryptography (ECC)*. For detailed information on ECC, the reader is referred to [15, 30].

2.3.1 Elliptic Curves

An elliptic curve E over a field F is defined by the following Weierstrass equation

$$E : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6 \ , \tag{2.5}$$

where $a_1, a_3, a_2, a_4, a_6 \in F$, and the discriminant of E defined as

$$\Delta = -d_2^2 d_8 - 8d_4^3 - 27d_6^2 + 9d_2 d_4 d_6 \ ,$$

where

$$\begin{aligned} d_2 &= a_1^2 + 4a_2 \ , \\ d_4 &= 2a_4 + a_1 a_3 \ , \\ d_6 &= a_3^2 + 4a_6 \ , \\ d_8 &= a_1^2 a_6 + 4a_2 a_6 - a_1 a_3 a_4 + a_2 a_3^2 - a_4^2 \ , \end{aligned}$$

and $\Delta \neq 0$. For any extension field K of F, the set of K-rational points on E is

$$E(K) = \{(x, y) \in K \times K : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} \cup \{\mathcal{O}\}$$

where $\mathcal{O}$ is the point at infinity. Note that E is defined over F and any extension of it. The condition $\Delta \neq 0$ guarantees that the curve is smooth, i.e., there is only a single tangent line at every point of the curve.

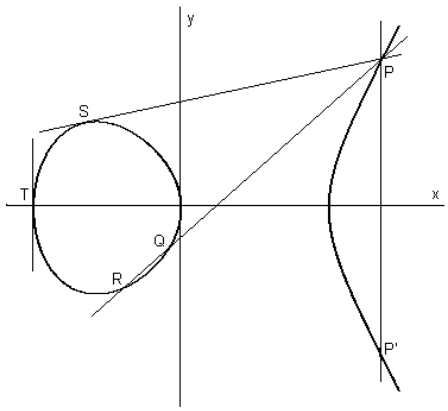

Figure 2.1: Point addition on the elliptic curve $E : y^2 = x^3 - 3x$.

2.3.2 Elliptic Curve Cryptography over $GF(p^m)$ where $p > 3$

When the characteristic p of a finite field F is different from 2 and 3, through change of variables the Weierstrass equation in (2.5), defining an elliptic curve over F, can be transformed into the curve

$$y^2 = x^3 + ax + b$$

where $a, b \in F$ and $\Delta = -16(4a^3 + 27b^2)$. In this case, for an extension field K of F, *E(K)* together with the point $\mathcal{O}$, serving as the identity element, forms an abelian group with the group law defined as follows.

1. *Identity Element:* For all $P \in E(K)$, $P + \mathcal{O} = \mathcal{O} + P = P$ and $\mathcal{O} = -\mathcal{O}$.

2. *Negative of an Element:* If $P = (x, y) \in E(K)$, $-P = (x, -y) \in E(K)$ and $(x, y) + (x, -y) = \mathcal{O}$.

3. *Point Addition and Doubling:* Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on the curve *E(K)*. $P_3 = (x_3, y_3)$, where $P_3 = P_1 + P_2$ and $P_1 \neq -P_2$, can be found as follows:

$$x_3 = \lambda^2 - x_1 - x_2 \tag{2.6}$$

$$y_3 = \lambda(x_1 - x_3) - y_1 \tag{2.7}$$

where

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1}, & if \quad P_1 \neq P_2 \\ \\ \frac{3x_1^2 + a}{2y_1}, & if \quad P_1 = P_2 \end{cases}$$

The elliptic curve group law can be interpreted geometrically as shown in Figure 2.1. Note here that a straight line crossing an elliptic curve intersects it at no more than three points and the point addition of these intersection points under the elliptic curve group law results in the identity element, i.e., the point at infinity $\mathcal{O}$. A straight line may intersect an elliptic curve in mainly four different ways and the group law may be interpreted accordingly as follows:

- If a straight line intersects an elliptic curve at a *single point* T, i.e., if it is *tangent*, the intersection point is counted twice and hence $2T = \mathcal{O}$.
- If a straight line *perpendicular to the x axis* intersects an elliptic curve at *two points* P and P', then $P + P' = \mathcal{O}$ and hence $-P = P'$.
- If a straight line intersects an elliptic curve at two points P and S, where S is a tangent, then $P + 2S = \mathcal{O}$ and hence $2S = -P$.
- If a straight line intersects an elliptic curve at *three points* P, Q and R, then $P + Q + R = \mathcal{O}$ and hence $Q + R = -P$ which is the mirror image of P over the x axis, i.e., P' in Figure 2.1.

Elliptic Curve Point Multiplication:

The main operation in an elliptic curve cryptosystem is the *point multiplication* or *scalar multiplication* which is the computation of $k \cdot P$ where P is a point on an elliptic curve E and

k is an integer. The security of an elliptic curve cryptosystem relies on the intractability of solving the additive discrete logarithm problem, i.e., finding the value of k for a given point $k \cdot P$, where the point P generates a sufficiently large subgroup over E.

Point multiplication is achieved through repeated point additions and doublings, and constitutes the majority of the computational workload in public key cryptosystems based on elliptic curves. For instance, the Diffie-Hellman key exchange algorithm using an elliptic curve can be achieved as follows. Let *Alice* and *Bob* be two parties who want to generate a common secret key s. They select a random common point P over E. They also choose their random private keys as positive integers r_A and r_B, where both r_A and r_B are less than the order of P in the additive elliptic curve group, and announce $p_A = r_A \cdot P$ and $p_B = r_B \cdot P$ as their public keys, respectively. In this case, each party can compute the common secret key by using his/her own private key and the other party's public key as

$$r_A \cdot p_B = r_A \cdot r_B \cdot P$$

and

$$r_B \cdot p_A = r_B \cdot r_A \cdot P$$

which result in the same point on the elliptic curve and both parties can use the x coordinate of this common point as their common secret key s.

2.4 Number Theoretic Transform

The *number theoretic transform (NTT) over a ring*, also known as *the discrete Fourier transform (DFT) over a finite field*, was introduced by Pollard [62]. The NTT computations over $GF(p)$ are defined by utilizing a d^{th} primitive root of unity, denoted by r, from $GF(p)$ or a finite extension of $GF(p)$.

Definition 1 *r is a primitive d^{th} root of unity modulo n if $r^d = 1 \pmod{n}$ and $r^{d/t} - 1 \neq 0 \pmod{n}$ for any prime divisor t of d.*

For a sequence (a) of length d whose entries are from $GF(p)$, the forward NTT of (a) over $GF(p)$, denoted by (A), can be computed as

$$A_j = \sum_{i=0}^{d-1} a_i r^{ij} \ , \ 0 \leq j \leq d-1 \ . \tag{2.8}$$

Here we refer to the elements of (a) and (A) by a_i and A_i, respectively, for $0 \leq i \leq d-1$. Likewise, the inverse NTT of (A) over $GF(p)$ can be computed as

$$a_i = \frac{1}{d} \cdot \sum_{j=0}^{d-1} A_j r^{-ij} \ , \ 0 \leq i \leq d-1 \ . \tag{2.9}$$

The sequences (a) and (A) are referred to as the *time and frequency domain representations*, respectively, of the same sequence.

Note that unlike the complex number $r = e^{j2\pi/d}$ generally used as the d-th primitive root of unity in the DFT computations, a finite field or ring element r can be utilized for the same purpose in an NTT. Choosing $r = \pm 2$ turns multiplications with powers of r into simple shift operations and thus enables very efficient NTT computations. However, we would like to caution the reader that in an NTT over $GF(p)$, the modulus p and the transform length d can not be chosen independently of each other. For an NTT of length d to exist over $GF(p)$, the condition $d|p-1$ should be satisfied[1]. Note that, in this case, the equality $\mathrm{GCD}(d, p) = 1$ holds for the *greatest common denominator* of d and p, and hence the inverse of d in $GF(p)$, which is needed for the inverse NTT computations, always exists. For further information on the NTT, we refer the reader to [52, 14, 39] .

2.4.1 Representing OEF Elements in the Frequency Domain

Using the NTT, one can convert an element of an OEF $GF(p^m)$, which is a polynomial of degree at most $(m-1)$ with coefficients in $GF(p)$, into its frequency domain sequence representation. For instance, for $a(x) \in GF(p^m)$ represented as

$$a(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-1} x^{m-1} \ ,$$

one can form a $d \geq m$ element sequence by using its ordered coefficients as

$$(a) = (a_0, a_1, a_2, \ldots, a_{m-1}, 0, 0, \ldots, 0) \ , \tag{2.10}$$

where $d-m$ zeros are appended to the right when $d > m$. By applying the NTT formula in (2.8) over the sequence (a), the frequency domain representation (A) of $a(x)$ can be obtained as the following sequence

[1]Likewise, if the NTT is performed over a finite ring $\mathbb{Z}_n$, i.e. if the transform modulus is a composite number of the form $n = p_1^{e_1} p_2^{e_2} p_3^{e_3} \ldots p_k^{e_k}$, where $p_1, p_2, p_3, \ldots, p_k$ are prime, then d must divide $\gcd(p_1 - 1, p_2 - 1, p_2 - 1, p_3 - 1, \ldots, p_k - 1)$ [4].

$$(A) = (A_0, A_1, A_2, \ldots, A_{m-1}, \ldots, A_{d-1}) \ . \tag{2.11}$$

It is possible to achieve finite field arithmetic in the frequency domain by using the frequency domain representations of OEF elements. In the following chapters, we will present efficient algorithms which realize OEF arithmetic operations such as multiplication and inversion in the frequency domain.

2.5 Conclusion

In this chapter we presented an overview on finite fields, OEFs and ECC. Furthermore, we mentioned about the NTT and the representation of OEF elements in the frequency domain using the NTT. In Chapter 3, we will prove the connection between the NTT and the residue number system. Furthermore, we will present a simple algorithm explaining frequency domain multiplication in $GF(p^m)$ using the NTT.

Chapter 3

Finite Field Multiplication Using the Number Theoretic Transform[1]

3.1 Introduction

We begin this chapter by presenting a straightforward algorithm for achieving multiplication in $GF(p^m)$ using the *number theoretic transform (NTT)*. Then we show the relationship between the NTT and the *residue number system (RNS)* [83, 78, 77] and prove that the frequency domain representation of a polynomial is equivalent to its RNS representation provided that the RNS is defined by a special group of modulus polynomials. Furthermore, we prove that the straightforward NTT based algorithm for multiplication in $GF(p^m)$ is computationally equivalent to an optimal case of multiplication in $GF(p^m)$ using the RNS.

3.2 Multiplication in $GF(p^m)$ Using the NTT

In Section 2.4, we learned about the NTT which is also known as the DFT over a finite field. A significant application of the DFT is convolution. Convolution of two d-element sequences (a) and (b) in the time domain results in another d-element sequence (c) and can be computed as follows:

$$c_i = \sum_{j=0}^{d-1} a_j b_{i-j \bmod d} \ , \quad 0 \leq i \leq d-1 \ . \tag{3.1}$$

[1] **The material presented in this chapter is included in [66].**

According to the convolution theorem, the above convolution operation in the time domain is equivalent to the following computation in the frequency domain:

$$C_i = A_i \cdot B_i \ , \ \ 0 \leq i \leq d-1 \ , \tag{3.2}$$

where (A), (B) and (C) denote the DFTs of (a), (b) and (c), respectively. Hence, convolution of two d-element sequences in the time domain, with complexity $O(d^2)$, is equivalent to simple pairwise multiplication of the DFTs of these sequences and has a surprisingly low $O(d)$ complexity. For details of the DFT, convolution theorem and their applications, the interested reader is referred to [18, 80, 60]. In this dissertation we are interested in the DFT in the context of finite fields, therefore we will use the terms DFT and NTT interchangeably.

Note that the summation in (3.1) is the *cyclic convolution* of the sequences (a) and (b). We have seen in (3.2) that this cyclic convolution can be computed very efficiently in the frequency domain by pairwise coefficient multiplications. Multiplication of two polynomials, on the other hand, is equivalent to the *acyclic (linear) convolution* of the polynomial coefficients. However, if we represent elements of $GF(p^m)$, which are at most $(m-1)^{st}$ degree polynomials with coefficients in $GF(p)$, with at least $d = (2m-1)$ element sequences by appending zeros at the end, then the cyclic convolution of two such sequences will be equivalent to their acyclic convolution and hence give us their polynomial multiplication.

Remember in (2.10) that one can form sequences by taking the ordered coefficients of polynomials. For instance,

$$a(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-1} x^{m-1} \ ,$$

an element of $GF(p^m)$ in polynomial representation, can be interpreted as the following $d \geq m$ element sequence after appending $d-m$ zeros to the right:

$$(a) = (a_0, a_1, a_2, \ldots, a_{m-1}, 0, 0, \ldots, 0) \ . \tag{3.3}$$

For $a(x), b(x) \in GF(p^m)$, and for $d \geq 2m-1$, the cyclic convolution of (a) and (b) yields a sequence (c) whose first $2m-1$ entries can be interpreted as the coefficients of a polynomial $c(x)$ such that $c(x) = a(x) \cdot b(x)$. The computation of this cyclic convolution can be performed by simple pairwise coefficient multiplications in the frequency domain. The below straightforward algorithm, presented as Algorithm 1, realizes the polynomial multiplication $c(x) = a(x) \cdot b(x)$ in the frequency domain. Note that Algorithm 1 computes the polynomial product in the frequency domain but the final reduction by the field generating polynomial

remains to be computed in the time domain.

Algorithm 1 Polynomial Multiplication by the Direct Application of the NTT

Input: $a(x), b(x) \in GF(p^m)$
Output: $c(x) = a(x) \cdot b(x)$
1: Interpret $a(x)$ and $b(x)$ as the sequences (a) and (b) with length $d \geq 2m-1$
2: Convert (a) and (b) into (A) and (B) using the NTT as in (2.8)
3: Multiply (A) with (B) to compute (C) as in (3.2)
4: Convert (C) to (c) using the inverse NTT as in (2.9)
5: Interpret the first $2m-1$ coefficients of (c) as the coefficients of $c(x) = a(x) \cdot b(x)$
6: Return $c(x)$

3.3 On the Relationship Between the NTT and RNS

In this section we describe the RNS and prove that the NTT is a special case of the RNS.

3.3.1 RNS and the Chinese Remainder Theorem

An RNS [83, 78, 77] can represent a large polynomial using a set of smaller polynomials. For a polynomial $a(x)$ and a modulus polynomial

$$P(x) = \prod_{i=0}^{d-1} p_i(x) \ ,$$

made up of relatively prime factors $p_i(x)$, for $0 \leq i \leq d-1$, $a(x)$ mod $P(x)$ can be uniquely represented by its residues modulo the polynomials $p_i(x)$, for $0 \leq i \leq d-1$, which define an RNS. Hence, provided that

$$\deg(a(x)) < \deg(P(x))$$

holds for the degrees of $a(x)$ and $P(x)$, $a(x)$ can be uniquely represented in this RNS by its residues given as

$$\left(\langle a(x)\rangle_{p_0(x)}, \langle a(x)\rangle_{p_1(x)}, \ldots, \langle a(x)\rangle_{p_{d-1}(x)}\right) \ , \tag{3.4}$$

where $\langle a(x)\rangle_{p_i(x)}$ denotes $a(x)$ mod $p_i(x)$.

Using the Chinese Remainder Theorem (CRT), the conversion from the RNS representation back to the normal polynomial representation can be achieved as

$$a(x) = \sum_{i=0}^{d-1} \langle a(x) \rangle_{p_i(x)} \cdot P_i(x) \ , \tag{3.5}$$

where

$$P_i(x) = \left(\frac{P(x)}{p_i(x)} \right) \cdot \left(\left(\frac{P(x)}{p_i(x)} \right)^{-1} \bmod p_i(x) \right), \quad 0 \leq i \leq d-1 \ . \tag{3.6}$$

3.3.2 On the Equivalence of the NTT and RNS

With Theorem 3, we show that the NTT is a special case of the RNS.

Theorem 3 *Computing the d-element NTT of the sequence* (a) *corresponding to* $a(x) \in GF(p^m)$*, as described with* (2.11) *where* $d \geq m$*, is equivalent to computing the RNS representation of* $a(x)$*, as described in* (3.4)*, in the RNS with the modulus polynomials* $p_i(x) = x - r^i$*, for* $0 \leq i \leq d-1$*, where* r *is the* d^{th} *primitive root of unity used for the NTT.*

Proof of Theorem 3 Let (A), with elements A_i for $0 \leq i \leq d-1$, denote the NTT of (a) corresponding to $a(x)$, and let $\left(\langle a(x) \rangle_{p_0(x)}, \langle a(x) \rangle_{p_1(x)}, \ldots, \langle a(x) \rangle_{p_{d-1}(x)} \right)$, with $p_i(x) = x - r^i$ for $0 \leq i \leq d-1$, denote the RNS representation of $a(x)$. We need to show that $A_i = \langle a(x) \rangle_{p_i(x)}$. Remember in (2.8) that the coefficients of (A) are computed as follows

$$A_i = \sum_{j=0}^{d-1} a_j r^{ji} \ , \ 0 \leq i \leq d-1 \ . \tag{3.7}$$

Likewise, for $a(x)$ represented as

$$a(x) = \sum_{j=0}^{m-1} a_j x^j$$

in standard basis, the i^{th} residue of $a(x)$, modulo $p_i(x) = x - r^i$, for $0 \leq i \leq d-1$, can be computed as

$$\begin{aligned} \langle a(x) \rangle_{p_i(x)} &= a(x) \bmod (x - r^i) \\ &= a(r^i) \\ &= \sum_{j=0}^{m-1} a_j r^{ji} \\ &= \sum_{j=0}^{d-1} a_j r^{ji} \ , \end{aligned} \tag{3.8}$$

where $a_j = 0$ for $m \leq j \leq d-1$. Since the summations in (3.7) and (3.8) are equivalent, the NTT and RNS representations of $a(x)$ are equivalent. □

3.3.3 Polynomial Multiplication Using the RNS

The RNS representation facilitates a divide-and-conquer method for polynomial multiplication. If the degree of a product polynomial $c(x) = a(x) \cdot b(x)$ is less than the degree of an RNS modulus $P(x)$, then the multiplication of $a(x)$ and $b(x)$ can be achieved in the RNS representation by simply multiplying their corresponding residues. The computation of $c(x) = a(x) \cdot b(x)$ can be conducted in the RNS representation as

$$\left(\langle c(x)\rangle_{p_0(x)}, \langle c(x)\rangle_{p_1(x)}, \ldots, \langle c(x)\rangle_{p_{k-1}(x)}\right) \ ,$$

where

$$\langle c(x)\rangle_{p_i(x)} = \langle \ \langle a(x)\rangle_{p_i(x)} \ \cdot \ \langle b(x)\rangle_{p_i(x)} \ \rangle_{p_i(x)} \ .$$

Thus, multiplication of two polynomials can be achieved in a completely parallel manner with minimal effort using the RNS representation. If the modulus polynomials $p_i(x)$, for $0 \leq i \leq d-1$, defining the RNS are all first degree binomials, then polynomial multiplication using the RNS representation can be achieved with the minimal number of coefficient multiplications which is only linear in the number of polynomial coefficients.

The following algorithm achieves polynomial multiplication of $a(x), b(x) \in GF(p^m)$ using the RNS representation. We would like to note that, similar to Algorithm 1, Algorithm 2 achieves only the polynomial multiplication of the finite field elements and the final modular reduction by the field generating polynomial remains to be computed.

Algorithm 2 RNS Polynomial Multiplication Using the CRT

Input: $a(x), b(x) \in GF(p^m)$
Output: $c(x) = a(x) \cdot b(x)$
1: Obtain $\langle a(x)\rangle_{p_i(x)}$ and $\langle b(x)\rangle_{p_i(x)}$, for $0 \leq i \leq d-1$, where $d \geq 2m-1$
2: Compute $\langle c(x)\rangle_{p_i(x)} = \langle a(x)\rangle_{p_i(x)} \cdot \langle b(x)\rangle_{p_i(x)}$, for $0 \leq i \leq d-1$
3: Obtain $c(x)$ from $\langle c(x)\rangle_{p_i(x)}$, for $0 \leq i \leq d-1$, using the CRT as described in (3.5)
4: Return $c(x)$

3.3.4 On the Equivalence of the Polynomial Multiplication Algorithms Using the NTT and RNS

In Section 3.3.2 we saw the relationship between the NTT and RNS. Now, we will prove with Theorem 5 that Algorithm 1 which utilizes the NTT using a d^{th} primitive root of unity r is equivalent to Algorithm 2, which performs multiplication in $GF(p^m)$ utilizing the RNS and CRT, when the modulus polynomials defining the RNS in Algorithm 2 are the first degree binomials $p_i(x) = x - r^i$, for $0 \leq i \leq d-1$. We first present the following theorem which will be used in the proof of Theorem 5.

Theorem 4 *For a d^{th} primitive root of unity r, the following equality holds:*

$$x^d - 1 = (x - r^0)(x - r^1)(x - r^2) \ \cdots \ (x - r^{d-1}) \tag{3.9}$$

Proof of Theorem 4 Any polynomial of degree d is uniquely identified by its d roots. Hence, it suffices to show that $x^d - 1$ and $(x - r^0)(x - r^1)(x - r^2) \ \cdots \ (x - r^{d-1})$ have the same roots in order to prove that they are equivalent. Clearly, since r is a d^{th} primitive root of unity, r^i is a distinct root of $(x-r^0)(x-r^1)(x-r^2) \ \cdots \ (x-r^{d-1})$ for $i = 0, 1, 2, \cdots, d-1$. Furthermore, again since r is a d^{th} primitive root of unity, $r^d = 1$ and $(r^i)^d - 1 = (r^d)^i - 1 = 1^i - 1 = 0$ for $i = 0, 1, 2, \cdots, d-1$. Hence, r^i is also a root of the polynomial $x^d - 1$ for $i = 0, 1, 2, \cdots, d-1$. Thus, since the two polynomials $x^d - 1$ and $(x - r^0)(x - r^1)(x - r^2) \ \cdots \ (x - r^{d-1})$ have the same roots, they are equivalent. □

Theorem 5 describes the relationship between Algorithm 2, which uses the RNS and CRT, and Algorithm 1 which uses the NTT.

Theorem 5 *Algorithm 1, which utilizes the NTT with a primitive root of unity r of order $d \geq 2m-1$ for multiplication in $GF(p^m)$, is equivalent to Algorithm 2 which realizes multiplication in $GF(p^m)$ utilizing the RNS defined by the relatively prime binomials $x-r^0$, $x-r^1$, $x-r^2$, ... , $x-r^{d-1}$.*

Proof of Theorem 5 Due to the convolution theorem [18, 60], pairwise multiplication of the elements of two d-element sequences in the frequency domain corresponds to the cyclic convolution of the two sequences in the time domain. Hence, by pairwise multiplying the

frequency domain representations of two input polynomials, Algorithm 1 yields a product which is the cyclic convolution of the input polynomials and thus equals $a(x) \cdot b(x) \bmod x^d - 1$. On the other hand, Algorithm 2 computes $a(x) \cdot b(x) \bmod P(x)$, where $P(x) = \prod_{0 \le i \le d-1} (x - r^i)$, which is equal to $a(x) \cdot b(x) \bmod x^d - 1$ due to Theorem 4. Hence, Algorithms 1 and 2 are equivalent. □

3.4 Conclusion

In this chapter, we established the relationship between the NTT and RNS, and proved that the NTT is a special case of the RNS. The NTT based method for finite field multiplication in $GF(p^m)$, presented with Algorithm 1, or the equivalent RNS based method presented with Algorithm 2, has only $O(m)$ complexity in terms of the required $GF(p)$ multiplications, ignoring the conversions which have $O(m^2)$ complexity. In the next chapter, we will provide efficient parameters for Algorithm 1 which facilitate extremely efficient conversions between the time and frequency domains, and thus help meet the minimal theoretical bound of $2m-1$ for the number of required multiplications in $GF(p)$ to achieve a multiplication in $GF(p^m)$ where p is an odd prime [83].

Chapter 4

Number Theoretic Transforms for Efficient Multiplication in $GF(p^m)$ for Elliptic Curve Cryptography[1]

4.1 Introduction

Frequency domain finite field multiplication, as described with Algorithm 1 in Chapter 3, can be sped up by using special parameters for the NTT computations. In this chapter, we present an overview of specialized NTTs and provide lists of efficient parameters, relevant to ECC, for their application to multiplication in $GF(p^m)$ using Algorithm 1.

4.2 Mersenne Transform

An NTT of special interest is *the Mersenne transform*, which is an NTT with arithmetic modulo a Mersenne number of the form $M_n = 2^n - 1$ [71]. The Mersenne transform allows for very efficient forward and inverse NTT operations for $r = \pm 2$. Multiplication of an n-bit number with integer powers of 2 modulo M_n can be achieved with a simple bitwise left rotation of the n-bit number, e.g. multiplication of an n-bit number with 2^i modulo M_n can be achieved with a simple bitwise left rotation by $i \bmod n$ bits. Similarly, multiplication of an n-bit number with integer powers of -2 modulo M_n can be achieved with a simple bitwise left rotation of the number, in addition to a negation if the power of -2 is odd. Also, note

[1] **The material presented in this chapter is included in [68, 66].**

that negation of an n-bit number modulo M_n can simply be achieved by flipping all n bits of the number. Hence, when $r = \pm 2$ all of the multiplications by powers of r in the forward and inverse NTT computations in a Mersenne transform can be achieved with simple bitwise rotations. In this case, for a transform length of d, the forward NTT computation can be achieved with only $(d-1)^2$ simple rotations and $d(d-1)$ additions/subtractions avoiding any multiplications. For the inverse NTT computation additional d constant multiplications with $1/d \bmod M_n$ are required. Hence, when p is a Mersenne prime, multiplication in $GF(p^m)$ using Algorithm 1 has only $O(m)$ complexity in terms of $GF(p)$ multiplications and $O(m^2)$ complexity in terms of $GF(p)$ additions/subtractions and rotations. For a more detailed complexity analysis and efficient implementation ideas for the Mersenne transform, we refer the interested reader to [71].

Remember that, as in all number theoretic transforms, in a Mersenne transform the values of the sequence length d and the d^{th} primitive root of unity r are dependent on each other and can not be chosen independently. In a Mersenne transform over $GF(p)$, where $p = M_n = 2^n - 1$ is a Mersenne prime, and for $r = \pm 2$, the following equalities hold determining the relationship between d and r:

$$d = \begin{cases} n\,, & r = 2 \\ 2n\,, & r = -2 \end{cases}$$

In Table 4.1, we provide a list of parameters for utilizing the Mersenne transform which may yield efficient multiplication in $GF(p^m)$ in the frequency domain, e.g. by using Algorithm 1 in Chapter 3, for operand sizes relevant to ECC. Note that when $p = M_n = 2^n - 1$, $r = 2$, $d = n$ and $m = (n+1)/2$, Algorithm 1 performs multiplication in $GF(p^m)$ meeting the minimal theoretical bound of $2m - 1$ for the number of required $GF(p)$ multiplications [83].

4.3 Pseudo-Mersenne Transform

Similar to the Mersenne transform achieved modulo a Mersenne number, an NTT modulo an integer submultiple of a Mersenne number, e.g., $M_n/t = (2^n - 1)/t$ for an integer $t > 1$, can also be performed efficiently and is called *the pseudo-Mersenne transform* [58]. In a pseudo-Mersenne transform all arithmetic operations can be achieved using Mersenne number arithmetic modulo the Mersenne number M_n and only the final result needs to be reduced modulo M_n/t. Hence, the pseudo-Mersenne transform, similar to the Mersenne

$\mathbf{p = M_n = 2^n - 1}$	**m**	**d**	**r**	**equivalent binary field size**
$2^{13}-1$	11	26	-2	$\sim 2^{143}$
$2^{13}-1$	12	26	-2	$\sim 2^{156}$
$2^{13}-1$	13	26	-2	$\sim 2^{169}$
$2^{17}-1$	9	17	2	$\sim 2^{153}$
$2^{17}-1$	11	34	-2	$\sim 2^{187}$
$2^{17}-1$	12	34	-2	$\sim 2^{204}$
$2^{17}-1$	13	34	-2	$\sim 2^{221}$
$2^{17}-1$	14	34	-2	$\sim 2^{238}$
$2^{17}-1$	15	34	-2	$\sim 2^{255}$
$2^{17}-1$	16	34	-2	$\sim 2^{272}$
$2^{17}-1$	17	34	-2	$\sim 2^{289}$
$2^{19}-1$	10	19	2	$\sim 2^{190}$
$2^{19}-1$	11	38	-2	$\sim 2^{209}$
$2^{19}-1$	12	38	-2	$\sim 2^{228}$
$2^{19}-1$	13	38	-2	$\sim 2^{247}$
$2^{19}-1$	14	38	-2	$\sim 2^{266}$
$2^{19}-1$	15	38	-2	$\sim 2^{285}$
$2^{19}-1$	16	38	-2	$\sim 2^{304}$
$2^{19}-1$	17	38	-2	$\sim 2^{323}$
$2^{19}-1$	18	38	-2	$\sim 2^{342}$
$2^{19}-1$	19	38	-2	$\sim 2^{361}$
$2^{31}-1$	11	31	2	$\sim 2^{341}$
$2^{31}-1$	12	31	2	$\sim 2^{372}$
$2^{31}-1$	13	31	2	$\sim 2^{403}$

Table 4.1: List of some $p = M_n$, m, d and $r = \pm 2$ values for efficient multiplication in $GF(p^m)$ in the frequency domain for ECC over finite fields of size 143 to 403 bits

transform, allows for very efficient forward and inverse NTT operations for $r = \pm 2$, since intermediary multiplications with integer powers of ± 2 can be achieved using arithmetic modulo M_n, rather than M_n/t, with a simple bitwise rotation, in addition to a negation if the power of $r = -2$ is odd. Also, remember that negation of an n-bit number modulo M_n can simply be achieved by flipping all n bits of the number. Thus, for a transform length of d, the forward NTT computation can be achieved with only $(d-1)^2$ simple rotations and $d(d-1)$ additions/subtractions avoiding any multiplications. For the inverse NTT computation additional d constant multiplications with $1/d \bmod M_n/t$ are required. Hence, when p is a pseudo-Mersenne prime, multiplication in $GF(p^m)$ using Algorithm 1 in Chapter 3 has only $O(m)$ complexity in terms of $GF(p)$ multiplications and $O(m^2)$ complexity in terms of $GF(p)$ additions/subtractions and rotations. We will see in Section 4.6 that for the cases when $p = M_n/t$ is a Fermat prime, with the use of the fast Fourier transform, this complexity can be further reduced to $O(m \log m)$ in terms of the number of required additions/subtractions and rotations. Although the pseudo-Mersenne transform increases the number of available transform lengths for the Mersenne transform, it has the downside of increasing the word size for the intermediary arithmetic operations from $n - \log_2 t$ to n for a pseudo-Mersenne transform modulo $M_n/t = (2^n - 1)/t$. In Table 4.2, we provide a list of parameters for utilizing the pseudo-Mersenne transform which may yield efficient multiplication in $GF(p^m)$ in the frequency domain, e.g. by using Algorithm 1 in Chapter 3, for operand sizes relevant to ECC.

4.4 Fermat Transform

A positive integer of the form $F_n = 2^{2^n} + 1$, where $n > 0$, is called a *Fermat number*. Fermat numbers which are prime are called *Fermat primes*. Similar to Mersenne primes, Fermat primes are popular choices as finite field characteristics since modular reductions by them can be achieved with simple addition/subtraction and shift operations. A number theoretic transform with arithmetic modulo a Fermat number is called *the Fermat Transform* [72, 71, 2, 3]. Fermat transforms were first defined and proposed for fast convolution and digital filtering by Agarwal and Burrus [2, 3]. In this work, we provide efficient parameters for their use in finite field multiplication, e.g. as with Algorithm 1 in Chapter 3, which may find applications in cryptography.

The Fermat transform allows for very efficient forward and inverse NTT computations for

$\mathbf{p = M_n/t = (2^n - 1)/t}$	**m**	**d**	**r**	**equivalent binary field size**
$(2^{15} - 1)/217$	15	30	-2	$\sim 2^{120}$
$(2^{23} - 1)/47$	9	23	2	$\sim 2^{162}$
$(2^{23} - 1)/47$	10	23	2	$\sim 2^{180}$
$(2^{23} - 1)/47$	11	23	2	$\sim 2^{198}$
$(2^{23} - 1)/47$	12	23	2	$\sim 2^{216}$
$(2^{23} - 1)/47$	17	46	-2	$\sim 2^{306}$
$(2^{23} - 1)/47$	18	46	-2	$\sim 2^{324}$
$(2^{23} - 1)/47$	19	46	-2	$\sim 2^{342}$
$(2^{23} - 1)/47$	20	46	-2	$\sim 2^{360}$
$(2^{23} - 1)/47$	21	46	-2	$\sim 2^{378}$
$(2^{23} - 1)/47$	22	46	-2	$\sim 2^{396}$
$(2^{23} - 1)/47$	23	46	-2	$\sim 2^{414}$
$(2^{27} - 1)/511$	11	27	2	$\sim 2^{209}$
$(2^{27} - 1)/511$	12	27	2	$\sim 2^{228}$
$(2^{27} - 1)/511$	13	27	2	$\sim 2^{247}$
$(2^{27} - 1)/511$	14	27	2	$\sim 2^{266}$
$(2^{32} - 1)/65535$	13	32	2	$\sim 2^{221}$
$(2^{32} - 1)/65535$	14	32	2	$\sim 2^{238}$
$(2^{32} - 1)/65535$	15	32	2	$\sim 2^{255}$
$(2^{32} - 1)/65535$	16	32	2	$\sim 2^{272}$
$(2^{33} - 1)/14329$	17	33	2	$\sim 2^{340}$
$(2^{37} - 1)/223$	11	37	2	$\sim 2^{330}$
$(2^{37} - 1)/223$	12	37	2	$\sim 2^{360}$
$(2^{37} - 1)/223$	13	37	2	$\sim 2^{390}$
$(2^{37} - 1)/223$	14	37	2	$\sim 2^{420}$
$(2^{37} - 1)/223$	15	37	2	$\sim 2^{450}$
$(2^{37} - 1)/223$	16	37	2	$\sim 2^{480}$
$(2^{37} - 1)/223$	17	37	2	$\sim 2^{510}$
$(2^{37} - 1)/223$	18	37	2	$\sim 2^{540}$
$(2^{37} - 1)/223$	19	37	2	$\sim 2^{570}$

Table 4.2: List of some $p = M_n/t$, m, d and $r = \pm 2$ values for efficient multiplication in $GF(p^m)$ in the frequency domain for ECC over finite fields of size 120 to 570 bits

$r = 2^{2^k}$, where k is a non-negative integer. For a Fermat prime $p = 2^{2^n}+1$, $2^{2^{n+1}} \equiv 1 \pmod{p}$ and $r = 2$ is a d^{th} primitive root of unity where $d = 2^{n+1}$. Likewise, $r = 2^{2^k}$, for $k \leq n+1$, is a d^{th} primitive root of unity where $d = 2^{n+1-k}$. Thus, since d is a power of 2 in this case, as we will see in Section 4.6 the fast Fourier transform [21] can be applied very efficiently for the NTT computations which significantly reduces the complexity of Algorithm 1 in Chapter 3. Also, all modular multiplications by powers of $r = 2^{2^k}$ can be achieved by simple shift and subtraction operations. In this case, a modular multiplication by a power of r is slightly more complex than that in the Mersenne transform where the same operation can be achieved with a mere bitwise rotation. However, since $F_n = 2^{2^n} + 1 = \frac{2^{2^{n+1}}-1}{2^{2^n}-1}$, the Fermat transform modulo $F_n = 2^{2^n} + 1$ is equivalent to the corresponding pseudo-Mersenne transform modulo $M_{2^{n+1}}/(2^{2^n} - 1) = \frac{2^{2^{n+1}}-1}{2^{2^n}-1}$ and can be achieved via Mersenne number arithmetic modulo the Mersenne number $2^{2^{n+1}} - 1$. Thus, one can achieve all multiplications by powers of $r = 2^{2^k}$ with simple bitwise rotations. The advantage of this approach compared with the Fermat transform using the Fermat number arithmetic is that modular multiplications with powers of r become mere rotations and can be achieved very easily, however it has the drawback of having almost the twice increase in the word size of the operands. Hence, when p is a Fermat number, the complexity of multiplication in $GF(p^m)$ using the Fermat transform is only $O(m)$ multiplications and $O(m \log m)$ additions/subtractions and shifts in terms of $GF(p)$ operations. In Table 4.3, we present a list of some Fermat primes and values for d and m that allow for application of the fast Fourier transform (see Section 4.6) and thus possibly efficient multiplication in $GF(p^m)$ in the frequency domain, e.g. as with Algorithm 1 in Chapter 3, for operand sizes relevant to ECC.

Efficient Fermat transform is also possible with a d^{th} primitive root of unity of the form $r = (\sqrt{2})^k$, for a positive integer k, thanks to the following relationship

$$\sqrt{2} \equiv 2^{e/4}(2^{e/2} - 1) \pmod{2^e + 1} .$$

Thus, when $r = (\sqrt{2})^k$ in a Fermat transform modulo $F_n = 2^{2^n} + 1$ or a pseudo-Fermat transform modulo $F_n/t = \frac{2^{2^n}+1}{t}$ (see Section 4.5), $r = (\sqrt{2})^{ki} = 2^a - 2^b$ for some positive integers a and b, and hence all multiplications by powers of $r = (\sqrt{2})^k$ can be carried out with at most two shifts/rotations and a subtraction.

$\mathbf{p = F_n = 2^{2^n} + 1}$	**m**	**d**	**r**	**equivalent binary field size**
$2^{2^3} + 1$	13	32	$\sqrt{2}$	117
$2^{2^3} + 1$	14	32	$\sqrt{2}$	126
$2^{2^3} + 1$	15	32	$\sqrt{2}$	135
$2^{2^3} + 1$	16	32	$\sqrt{2}$	144
$2^{2^4} + 1$	7	16	2^2	119
$2^{2^4} + 1$	8	16	2^2	136
$2^{2^4} + 1$	13	32	2	221
$2^{2^4} + 1$	14	32	2	238
$2^{2^4} + 1$	15	32	2	255
$2^{2^4} + 1$	16	32	2	272
$2^{2^4} + 1$	29	64	$\sqrt{2}$	478
$2^{2^4} + 1$	30	64	$\sqrt{2}$	495
$2^{2^4} + 1$	31	64	$\sqrt{2}$	512
$2^{2^4} + 1$	32	64	$\sqrt{2}$	544

Table 4.3: List of Fermat primes $p = F_n = 2^{2^n} + 1$ and d, r, m values for efficient multiplication in $GF(p^m)$ in the frequency domain for ECC over finite fields of size 117 to 544 bits

4.5 Pseudo-Fermat Transform

An NTT can also be efficiently performed modulo a pseudo-Fermat prime of the form $F_n/t = \frac{2^n+1}{t}$ for a positive integer t, and is called *the pseudo-Fermat transform* [59]. In a pseudo-Fermat transform, all intermediary arithmetic operations can be carried out using Fermat number arithmetic modulo $2^n + 1$ or Mersenne number arithmetic modulo the Mersenne number $M_{n+1} = 2^{n+1} - 1 = (2^n + 1) \cdot (2^n - 1)$ and only the final result needs to be reduced modulo $F_n/t = \frac{2^n+1}{t}$. Hence, the pseudo-Fermat transform retains some of the computational advantages of the Mersenne transform and introduces further variety in the available transform lengths in the expense of increasing the word size for the intermediary arithmetic operations from $n + 1 - \log_2 t$ to $n + 1$ for a pseudo-Fermat transform modulo $F_n/t = \frac{2^n+1}{t}$. In Table 4.4, we present a list of pseudo-Fermat primes and corresponding transform parameters suitable for finite field multiplication in $GF(p^m)$, e.g. as with Algorithm 1 of Chapter 3, relevant to ECC.

4.6 Fast Fourier Transform

An extremely efficient method for computing the DFT is the fast Fourier transform (FFT) [31, 21, 82]. The FFT algorithm works by exploiting the symmetry of the DFT computation and the periodicity of the d^{th} primitive root of unity r when the sequence length d is a composite number. For instance, if a sequence is of even length, i.e. if its length d is divisible by two, then by applying the FFT the DFT computation of this d-element sequence is basically reduced to the DFT computations of two $(d/2)$-element sequences, namely the sequence comprising only the even indexed elements of the original sequence and the sequence comprising only the odd indexed elements of the original sequence. The DFT of a d-element sequence (a), where d is divisible by 2, can be expressed as follows:

$$\begin{aligned} A_j &= \sum_{i=0}^{d-1} a_i r^{ij} && , \quad 0 \le j \le d-1 \\ &= \sum_{i=0}^{\frac{d}{2}-1} a_{2i} r^{2ij} + \sum_{i=0}^{\frac{d}{2}-1} a_{2i+1} r^{(2i+1)j} && , \quad 0 \le j \le d-1 \\ &= \sum_{i=0}^{\frac{d}{2}-1} a_{2i} (r^2)^{ij} + r^j \sum_{i=0}^{\frac{d}{2}-1} a_{2i+1} (r^2)^{ij} && , \quad 0 \le j \le d-1 \ . \end{aligned} \tag{4.1}$$

$\mathbf{p = F_n/t = (2^n + 1)/t}$	**m**	**d**	**r**	**equivalent binary field size**
$(2^{13}+1)/3$	13	26	2	$\sim 2^{156}$
$(2^{15}+1)/99$	15	30	2	$\sim 2^{135}$
$(2^{17}+1)/3$	17	34	2	$\sim 2^{272}$
$(2^{19}+1)/3$	19	38	2	$\sim 2^{342}$
$(2^{19}+1)/3$	10	19	2^2	$\sim 2^{180}$
$(2^{20}+1)/17$	8	16	$(\sqrt{2})^5$	$\sim 2^{128}$
$(2^{20}+1)/17$	20	40	2	$\sim 2^{320}$
$(2^{20}+1)/17$	10	20	2^2	$\sim 2^{160}$
$(2^{21}+1)/387$	21	42	2	$\sim 2^{273}$
$(2^{22}+1)/1985$	22	44	2	$\sim 2^{264}$
$(2^{22}+1)/1985$	11	22	2^2	$\sim 2^{132}$
$(2^{23}+1)/3$	23	46	2	$\sim 2^{506}$
$(2^{27}+1)/1539$	27	54	2	$\sim 2^{459}$
$(2^{27}+1)/1539$	9	18	2^3	$\sim 2^{153}$
$(2^{28}+1)/17$	14	28	2^2	$\sim 2^{336}$
$(2^{28}+1)/17$	7	14	2^4	$\sim 2^{168}$
$(2^{32}+1)/641$	16	32	2^2	$\sim 2^{368}$
$(2^{32}+1)/641$	8	16	2^4	$\sim 2^{184}$

Table 4.4: List of some $p = F_n/t$, m, d and $r = \pm 2$ values for efficient multiplication in $GF(p^m)$ in the frequency domain for ECC over finite fields of size 128 to 506 bits

Note that when r is a d^{th} primitive root of unity, r^2 is a $(\frac{d}{2})^{th}$ primitive root of unity. Hence, the above d-element DFT computation of A_j, for $0 \leq j \leq d-1$, can be performed with two $(\frac{d}{2})$-element DFTs which are the DFTs of the $(\frac{d}{2})$-element sequences consisting of the even indexed elements and the odd indexed elements of (a). In (4.1), the first and the second summations correspond to the $(\frac{d}{2})$-element DFTs of the sequences comprising the even and odd indexed elements of (a), respectively. Here, A_j needs to be computed for $0 \leq j \leq d-1$, not for $0 \leq j \leq \frac{d}{2}-1$. However, $(r^2)^j$ is periodic with $\frac{d}{2}$ for a d^{th} primitive root of unity r and d even, and hence $r^{j+\frac{d}{2}} = -r^j$. Thus, the equalities

$$\sum_{i=0}^{\frac{d}{2}-1} a_{2i}(r^2)^{i(j+\frac{d}{2})} = \sum_{i=0}^{\frac{d}{2}-1} a_{2i}(r^2)^{ij}$$

and

$$r^{j+\frac{d}{2}} \sum_{i=0}^{\frac{d}{2}-1} a_{2i+1}(r^2)^{i(j+\frac{d}{2})} = -r^j \sum_{i=0}^{\frac{d}{2}-1} a_{2i+1}(r^2)^{ij}$$

hold. Therefore, once A_j is computed for $0 \leq j \leq \frac{d}{2}-1$ as in (4.1) by performing two $(\frac{d}{2})$-element DFTs, $\frac{d}{2}-1$ multiplications for multiplications of the second summations by r^j (for $j=0$ no multiplication is necessary for a multiplication by r^j) and $\frac{d}{2}$ additions for merging the two summations together, we can compute A_j for $\frac{d}{2} \leq j \leq d-1$ immediately by using the same already computed summations and with only additional $\frac{d}{2}$ subtractions for merging the two summations. The inverse FFT can be computed in a similar manner as the forward FFT with the exception of the minus signs in front of the powers of r and d additional constant multiplications due to the multiplications with d^{-1}. When d is a power of two, the same approach can easily be applied recursively, surprisingly reducing the $O(d^2)$ complexity of the DFT computation to $O(d \log_2 d)$. Likewise, if d is a power of three, a similar approach could be be applied recursively to reduce the $O(d^2)$ complexity of the DFT computation to $O(d \log_3 d)$. As shown in [82], the symmetry of the DFT and the periodicity of r can similarly be exploited recursively whenever d is a composite number.

Unfortunately, the full recursive application of the FFT to NTT, for multiplication in $GF(p^m)$ using Algorithm 1, would find limited application since there are only a limited number of cases in Tables 4.1, 4.2, 4.3 and 4.4 where the available sequence length d is highly composite, e.g. a power of 2 or another small prime number. Application of the FFT to NTT, and thus to Algorithm 1, is most suited when the field characteristic is a Fermat prime or a pseudo-Fermat prime, as for the cases presented with Tables 4.3 and 4.4.

However, in applications of Algorithm 1 utilizing the Mersenne transform, where arithmetic operations can be performed more efficiently modulo a Mersenne prime $M_n = 2^n - 1$, the allowable sequence length d is either the prime number n (for $r = 2$) or $2n$ (for $r = -2$). Hence, either $d = n$ is prime and the FFT algorithm can not be applied at all or $d = 2n$ and only a single level of recursion is allowed in the FFT operations which would have limited computational advantage.

In the next chapter, we introduce the *DFT modular multiplication* algorithm which achieves both multiplication and modular reduction in the frequency domain and could be more efficient than Algorithm 1 of Chapter 3 or other efficient methods, especially for multiplication in Mersenne fields.

4.7 Conclusion

In this chapter, we presented special parameters which speed up the NTT computations, and thus multiplication in $GF(p^m)$ using Algorithm 1, for operand sizes relevant to ECC. Many of the presented parameters help Algorithm 1 meet the minimal theoretical bound of $2m - 1$ for the number of required $GF(p)$ multiplications to achieve a multiplication in $GF(p^m)$ when p is an odd prime [83]. In Chapter 5, we will present a non-trivial algorithm, named *DFT modular multiplication*, for multiplication in $GF(p^m)$ which utilizes the same parameters presented in this chapter and may achieve multiplication more efficiently than Algorithm 1.

Chapter 5

Modular Multiplication in the Frequency Domain[1]

5.1 Introduction

In many finite field applications, a chain of arithmetic operations need to be performed, rather than a solitary one. For example, in elliptic curve cryptography (ECC) a scalar point product is computed by applying a chain of finite field additions, subtractions, multiplications, squarings and inversions on the input point coordinates [15, 30]. The Montgomery residue representation has proven to be useful in this computation [56, 36]. In using this method, first the operands are converted to their respective Montgomery residue representations, then utilizing Montgomery arithmetic the desired computation is implemented, and finally the result is converted back to the normal integer or polynomial representation. If there are a large number of operations performed in the Montgomery domain, due to the efficiency of the intervening computations, the forward and backward conversion operations become affordable. We introduce the same notion for frequency domain arithmetic. We present an arithmetic operation in the frequency domain that is equivalent to Montgomery multiplication in the time domain. Due to the *linearity* property of the discrete Fourier transform (DFT) [18, 60], operations in the time domain such as addition/subtraction and multiplication by a scalar directly map to the frequency domain, i.e., for any two sequences (a) and (b) representing elements of $GF(p^m)$ in the time domain and for any two scalars

[1] **The material presented in this chapter is included in [69, 66].**

$y, z \in GF(p)$,

$$\text{DFT}(\ y \cdot (a)\ \pm\ z \cdot (b)\ )\ =\ y \cdot \text{DFT}(\ (a)\ )\ \pm\ z \cdot \text{DFT}(\ (b)\ )\ .$$

Hence, if a modular multiplication algorithm in the frequency domain can also be utilized, then all finite field operations such as addition/subtraction and multiplication can be performed in the frequency domain, and thus a finite field application such as ECC can be achieved completely in the frequency domain, assuming also that for inversion a Fermat-like inversion algorithm consisting of multiplications is utilized and/or projective coordinates are used to avoid inversions. In the remainder of this section, we introduce the *DFT modular multiplication* algorithm which allows for both polynomial multiplication and Montgomery modular reduction operations in the frequency domain.

5.2 Mathematical Notation

The DFT modular multiplication algorithm runs in the frequency domain, and therefore the parameters used in the algorithm are represented in their frequency domain sequence representations using the NTT as explained in Section 2.4.1. These parameters are the irreducible field generating polynomial $f(x)$, the normalized irreducible field generating polynomial $f_N(x) = f(x)/f(0)$, the sequence length d, the indeterminate x, the input operands $a(x), b(x) \in GF(p^m)$, and the result $c(x) = a(x) \cdot b(x) \cdot x^{-(m-1)} \in GF(p^m)$. The time domain sequence representations of the polynomial parameters are $(f), (f_N), (x), (a), (b)$ and (c), respectively, and their frequency domain sequence representations, i.e. the DFTs of the time domain sequence representations, are $(F), (F_N), (X), (A), (B)$ and (C). We will denote the elements of a sequence with the name of the sequence and a subscript for showing the location of the particular element in the sequence, e.g. for the indeterminate x represented as the following d-element sequence in the time domain

$$(x) = (0, 1, 0, 0, \cdots, 0)\ ,$$

the NTT of (x) is computed as the following d-element sequence

$$(X) = (1, r, r^2, r^3, r^4, r^5, \ldots, r^{d-1})$$

whose first and last elements are denoted as $X_0 = 1$ and $X_{d-1} = r^{d-1}$, respectively. Remember that r, defined as in Section 2.4, is the d^{th} primitive root of unity needed for the NTT

computations and $d \geq 2m-1$ is a required condition for successfully achieving polynomial multiplication in the frequency domain, as explained in Section 3.2.

5.3 The DFT Modular Multiplication Algorithm

DFT modular multiplication presented with Algorithm 3 consists of two parts: multiplication (Steps $1-3$) and Montgomery reduction (Steps $4-13$). Multiplication is performed by simple pairwise multiplication of the coefficients of the frequency domain sequence representations of the input operands. Reduction is more complex and performed by Montgomery reduction in the frequency domain. In the reduction process the normalized field generating polynomial $f_N(x) = f(x)/f_0 \bmod p$ is used. Hence, $f_N(x)$ is equivalent to $f(x)$ but normalized to have $f_N(0) = 1$ and thus $f_{N_i} = f_i/f_0 \bmod p$ and $F_{N_i} = F_i/f_0 \bmod p$, for $0 \leq i \leq d-1$.

Algorithm 3 DFT modular multiplication algorithm for $GF(p^m)$

Input: $d \geq 2m-1$, $(F_N) \equiv f_N(x)$, $(X) \equiv x$, $(A) \equiv a(x) \in GF(p^m)$, $(B) \equiv b(x) \in GF(p^m)$
Output: $(C) \equiv a(x) \cdot b(x) \cdot x^{-(m-1)} \in GF(p^m)$

1: **for** $i = 0$ to $d-1$ **do**
2: $\quad C_i \leftarrow A_i \cdot B_i$
3: **end for**
4: **for** $j = 0$ to $m-2$ **do**
5: $\quad S \leftarrow 0$
6: $\quad$ **for** $i = 0$ to $d-1$ **do**
7: $\quad\quad S \leftarrow S + C_i$
8: $\quad$ **end for**
9: $\quad S \leftarrow -S/d$
10: $\quad$ **for** $i = 0$ to $d-1$ **do**
11: $\quad\quad C_i \leftarrow (C_i + F_{N_i} \cdot S) \cdot X_i^{-1}$
12: $\quad$ **end for**
13: **end for**
14: Return (C)

Proof of Correctness:

DFT modular multiplication is a direct adaptation of Montgomery multiplication for the frequency domain. Polynomial multiplication part of the algorithm (Steps $1-3$) is performed via simple pairwise multiplications. As a result, the polynomial $c(x) = a(x) \cdot b(x)$ is obtained. In the modular reduction part (Steps $4-13$), S is computed such that $(c(x) + f_N(x) \cdot S)$ is a multiple of x. This is accomplished by computing $-c_0$, the negative of the first coefficient in

the time domain sequence (c), and then by making c_0 zero by adding $(F_N) \cdot S$ to (C) in the frequency domain. Then again in the frequency domain, $(c(x) + f_N(x) \cdot S)$ is divided by x and the result, which is congruent to $c(x) \cdot x^{-1}$ modulo $f(x)$ in the time domain, is obtained. This division of $(c(x) + f_N(x) \cdot S)$ by x is accomplished in the frequency domain by dividing $(C) + (F_N) \cdot S$ by (X) (in Step 11). By repeating Steps $5 - 12$ of the algorithm $m - 1$ times the final result (C), which represents $a(x) \cdot b(x) \cdot x^{-(m-1)} \in GF(p^m)$ in the frequency domain, is obtained.

□

The inputs of the DFT modular multiplication algorithm presented with Algorithm 3 are the sequence length $d \geq 2m - 1$ and the DFTs (A) and (B) of the d-element sequences (a) and (b) which represent $a(x), b(x) \in GF(p^m)$, respectively. The output of the algorithm is (C), the DFT of the sequence (c), where (c) represents the d-element sequence for $c(x) = a(x) \cdot b(x) \cdot x^{-(m-1)} \in GF(p^m)$. The extra $x^{-(m-1)}$ factor shows that the DFT modular multiplication algorithm actually computes the Montgomery product of input polynomials in the frequency domain. Hence, the input polynomials $a(x)$ and $b(x)$ may be viewed as the Montgomery residue representations of two polynomials $u(x)$ and $v(x)$ such that

$$a(x) = u(x) \cdot x^{m-1} \in GF(p^m)$$

and

$$b(x) = v(x) \cdot x^{m-1} \in GF(p^m) \ .$$

With DFT modular multiplication the residue representation is kept intact, i.e.,

$$a(x) \cdot b(x) \cdot x^{-(m-1)} = (u(x) \cdot v(x)) \cdot x^{m-1}$$

which allows for further computations in the frequency domain.

For $d \approx 2m$, modular multiplication in $GF(p^m)$ with the DFT modular multiplication algorithm requires only $2m$ multiplications in addition to $4m^2 - 3m - 1$ constant multiplications and $4m^2 - 5m + 1$ additions in the base field $GF(p)$, while the classical schoolbook method, given in (2.1), requires m^2 multiplications and $(m-1)^2$ additions ignoring the cost of modular reduction given in (2.2). In the next section, we will see that this complexity of DFT modular multiplication may be improved dramatically by using special values for p, r, d and the irreducible field generating polynomial $f(x)$.

5.4 Utilizing Efficient Parameters to Speed up DFT Modular Multiplication

Using the smallest possible sequence length d, satisfying $d \geq 2m-1$, will lead to the smallest number of arithmetic operations in the computation of DFT modular multiplication. Optimally, $d = 2m-1$ will lead to the least number of arithmetic operations.

As mentioned in Section 4.2, using a Mersenne prime as the modulus and selection of $r = \pm 2$ will allow for extremely efficient modular multiplications with $r^i = \pm 2^i$ for integer values of i. This modular multiplication can be achieved with a simple bitwise rotation (in addition to a negation when $r = -2$ and i is odd) which is inexpensive. In the DFT modular multiplication algorithm, this property may be exploited if the field characteristic p is chosen as a Mersenne prime and r is chosen as $r = \pm 2$. In that case, in Step 11 of the algorithm, multiplications with $X_i^{-1} = r^{-i} = (\pm 2)^{-i} = (\pm 1)^i \cdot 2^{-i \bmod d}$ become simple $(-i \bmod d)$-bit left-rotations (in addition to a negation when $r = -2$ and i is odd), which have negligible cost compared to a regular multiplication. Also, note that when p is a Mersenne prime, negation of an element of $GF(p)$ can be achieved by simply flipping its bits. Multiplications with F_{N_i} in Step 11 can also be avoided in a similar fashion for special $f(x)$. For instance, for the binomial $f(x) = x^m \pm r^{s_0}$ with s_0 an integer, $F_{N_i} = \pm r^{mi - s_0 \bmod d} + 1$ and hence for $r = \pm 2$ multiplications with F_{N_i} can be achieved with only one bitwise rotation and one addition/subtraction. Likewise, for the trinomial $f(x) = x^m \pm r^{s_{m'}} x^{m'} \pm r^{s_0}$ or $f(x) = x^m \mp r^{s_{m'}} x^{m'} \pm r^{s_0}$, where s'_m and s_0 are integers, $F_{N_i} = \pm r^{mi - s_0 \bmod d} + r^{m'i + s_{m'} - s_0 \bmod d} + 1$ or $F_{N_i} = \pm r^{mi - s_0 \bmod d} - r^{m'i + s_{m'} - s_0 \bmod d} + 1$, respectively, and hence multiplications with F_{N_i} can be achieved with only two bitwise rotations and two additions/subtractions. Finally, we would like to caution the reader that all these parameters p, d, r and $f(x)$ are dependent on each other and can not be chosen independently.

5.5 Existence of Efficient Parameters

In Tables 4.1, 4.2, 4.3 and 4.4, we gave lists of parameters that would yield efficient multiplication in $GF(p^m)$ using Algorithm 1 in Chapter 3 for operand sizes relevant to ECC. These same parameters can also be used for efficient multiplication in $GF(p^m)$ using the DFT modular multiplication algorithm. For each parameter listed in these tables, one can verify that there exist many special irreducible binomials of the form $x^m \pm 2^s$, or trinomials

of the form $x^m \pm r^{s_1}x_1 \pm r^{s_0}$ or $x^m \pm r^{s_1}x_1 \mp r^{s_0}$, as field generating polynomials that would allow for efficient DFT modular multiplication. For some of these cases there exist efficient irreducible binomials which we present with Theorem 7 and with Tables 8.1, 8.2 and 8.4 in Appendix, whereas for other cases we were not able to find such binomials and included lists of efficient irreducible trinomials instead (see Tables 8.1, 8.2, 8.3 and 8.4 in Appendix). However, as shown in Theorem 7, for the computationally more desirable cases of $d = 2m$, $m = n$ and $p = 2^n - 1$, where $r = 2$, there always exist efficient irreducible binomials for finite fields practically relevant to ECC. We would like to first present Theorem 6 and Definition 2, which will be used in the proof of Theorem 7.

Theorem 6 *[53] Let $\alpha, \beta \in GF(p)$ and $\alpha = \beta^i$. The orders of α and β are related as*

$$\mathrm{ord}(\alpha) = \frac{\mathrm{ord}(\beta)}{\gcd(i, \mathrm{ord}(\beta))} ,$$

where $\mathrm{ord}(a)$ *denotes the order of field element a and* $\gcd(a, b)$ *denotes the greatest common denominator of a and b.*

Definition 2 *A Wieferich prime is an odd prime p which satisfies $2^{p-1} = 1 \bmod p^2$.*

Theorem 7 *For a Mersenne prime $p = 2^n - 1$ and for $m = n$, a binomial of the form $x^m \pm 2^s$, where s is an integer not congruent to 0 modulo n, is irreducible in $GF(p)[x]$ if m is not a Wieferich prime.*

Proof of Theorem 7 For a binomial of the form $x^m \pm 2^s$ to be irreducible in $GF(p)[x]$, it needs to satisfy all three conditions of Theorem 1. The third condition is satisfied since $m = n$ is a prime number and the condition $m = 0 \bmod 4$ never holds. For the first and second conditions, we consider the cases $x^m - 2^s$ and $x^m + 2^s$ separately.

Let us first consider the binomials $x^m - 2^s$. When p is a Mersenne prime of the form $p = 2^n - 1$ the order of 2 in $GF(p)$ is $n = m$ since $2^n = 1 \pmod p$ and $2^i \neq 1 \pmod p$ for $i < n$. Due to Theorem 6 the order of 2^s in $GF(p)$ is $\frac{\mathrm{ord}(2)}{\gcd(s,\mathrm{ord}(2))} = \frac{m}{\gcd(s,m)} = m$. The only prime factor of m, which is itself, divides m and hence the first condition of Theorem 1 is satisfied. The second condition of Theorem 1 is satisfied since $m | \frac{2^m-2}{m}$ never holds unless m is a Wieferich prime.

Now, let us consider the binomials $x^m + 2^s = x^m - (-2^s)$. Let us first find the order of (-2^s) in $GF(p)$, i.e., the smallest positive integer k such that $(-2^s)^k = 1 \pmod p$. The equality $(-2^s)^k = (-1)^k \cdot (2^s)^k = 1 \pmod p$ can hold true in only two cases:

- Case 1: The integer k is odd, thus $(-1)^k = -1 \pmod{p}$, and $(2^s)^k = -1 \pmod{p}$
- Case 2: The integer k is even, thus $(-1)^k = 1 \pmod{p}$, and $(2^s)^k = 1 \pmod{p}$

We have $-1 \bmod p = 2^n - 2 = 2(2^{n-1} - 1)$ and $(2^s)^k \bmod p = 2^{sk \bmod n}$. The equality $2(2^{n-1}-1) = 2^{sk \bmod n}$ never holds for any positive integer k. Hence, the equality "$(2^s)^k = -1 \pmod{p}$" in Case 1 can not hold and Case 1 is not possible. So, the integer k, which is the order of (-2^s) in $GF(p)$, satisfies Case 2, i.e., it is even and the smallest positive integer that satisfies $(2^s)^k = 1 \pmod{p}$. Since in $GF(p)$ $\text{ord}(2^s) = m$ and m is odd, the smallest even k which satisfies $(2^s)^k = 1 \pmod{p}$ is $2m$. Hence $2m$ is the order of (-2^s) in $GF(p)$. The first condition of Theorem 1 is satisfied for the irreducible binomials of the form $x^m + 2^s = x^m - (-2^s)$ since the only prime factor of m, which is itself, divides $2m$, the order of (-2^s) in $GF(p)$. The second condition of Theorem 1 is also satisfied since $m|\frac{2^m-2}{2m}$ never holds unless m is a Wieferich prime. □

The only known Wieferich primes are 1093 and 3511. It is also known that there are no other Wieferich primes less than 4×10^{12} [22]. Hence, for the more efficient cases in Table 4.1 where the field characteristic $p = M_n = 2^n - 1$ is a Mersenne prime and $m = n$, m^{th} degree irreducible binomials of the form $x^m \pm 2^s$, for a nonzero integer s, always exist.

5.6 Complexity Analysis

In this section, we present the complexity of DFT modular multiplication for a practical set of parameters relevant to ECC and compare it with the classical schoolbook method, given with (2.1) and (2.3), and the NTT based method presented with Algorithm 1. In our complexity analysis we assume the use of a Mersenne prime as the finite field characteristic p, an irreducible field generating binomial of the form $f(x) = x^m \pm 2^{s_0}$, a d-th primitive root of unity $r = \pm 2$ and a sequence length as $d \approx 2m$. The field parameters we use, such as the low Hamming weight field generating polynomial and Mersenne prime field characteristic, lead to efficient implementation of multiplication for all methods. Therefore, for the selected parameters, we can safely assume that our comparisons are fair. In Table 5.1, we present the complexities of multiplication in $GF(p^m)$ in terms of $GF(p)$ operations for the classical school book method, given with (2.1) and (2.3), Algorithm 1 and Algorithm 3 when such ideal parameters are used. Note that the astonishingly low $O(m)$ complexity of Algorithms 1

and 3 in terms of the number of $GF(p)$ multiplications is achieved under the ideal conditions with the efficient field parameters together with the choice of $r = \pm 2$.

	Schoolbook	**Algorithm 1**	**Algorithm 3**
# Multiplications	m^2	$\approx 2m$	$\approx 2m$
# Constant Multiplications	–	$\approx 2m - 1$	$m - 1$
# Additions/Subtractions	$m^2 - m$	$\approx 8m^2 - 7m$	$\approx 6m^2 - 7m + 1$
# Rotations	$m - 1$	$\approx 8m^2 - 11m + 3$	$\approx 4m^2 - 4m$

Table 5.1: Complexity of multiplication in $GF(p^m)$ in terms of the number of $GF(p)$ operations when $f(x) = x^m \pm 2^{s_0}$, p is a Mersenne prime and $d \approx 2m$

One could argue that for a Mersenne transform modulo a Mersenne prime $p = 2^n - 1$, where $r = -2$ and $d = 2n$ is composite, it is possible to utilize the FFT [21] for one level and obtain faster computations of the forward and inverse NTT in Algorithm 1 for multiplication in $GF(p^m)$. We present the equivalent single level FFT computation for the forward NTT operation, as used in Step 2 of Algorithm 1 for obtaining the frequency domain representations of $a(x), b(x) \in GF(p^m)$, with (5.1) and (5.2) exemplarily for $a(x)$.

$$A_j = \sum_{i=0}^{\frac{m-1}{2}} a_{2i} r^{2ij} + r^j \sum_{i=0}^{\frac{m-3}{2}} a_{2i+1} r^{2ij} \ , \ 0 \le j \le m-1 \tag{5.1}$$

$$A_{j+m} = \sum_{i=0}^{\frac{m-1}{2}} a_{2i} r^{2ij} - r^j \sum_{i=0}^{\frac{m-3}{2}} a_{2i+1} r^{2ij} \ , \ 0 \le j \le m-1 \tag{5.2}$$

Similarly, for the inverse NTT operation, as used in Step 4 of Algorithm 1 for obtaining the time domain representation of $c(x) = a(x) \cdot b(x)$, the equivalent single level inverse FFT computation is presented with (5.3), (5.4) and (5.5).

$$c_i = \frac{1}{d} \cdot \left(\sum_{j=0}^{m-1} C_{2j} r^{-2ij} + r^{-i} \sum_{j=0}^{m-1} C_{2j+1} r^{-2ij} \right) \ , \ 0 \le i \le m-2 \tag{5.3}$$

$$c_{i+m} = \frac{1}{d} \cdot \left(\sum_{j=0}^{m-1} C_{2j} r^{-2ij} - r^{-i} \sum_{j=0}^{m-1} C_{2j+1} r^{-2ij} \right) \ , \ 0 \le i \le m-2 \tag{5.4}$$

$$c_{m-1} = \frac{2}{d} \sum_{j=0}^{m-1} C_{2j} r^{-2(m-1)j} \tag{5.5}$$

Note that the equations (5.1) and (5.2) in the forward FFT computation, and similarly (5.3) and (5.4) in the inverse FFT computation, are the same except for the sign between the two summations. Using (5.1) and (5.2), the frequency domain representation of each one of $a(x)$ and $b(x)$, as in Step 2 of Algorithm 1, can be obtained with only around m^2 additions/subtractions and $m^2 - 2m + 1$ bitwise rotations in $GF(p)$. Similarly, using (5.3), (5.4) and (5.5), the time domain sequence representation (c) of the product $c(x) = a(x) \cdot b(x)$, as in Step 4 of Algorithm 1, can be obtained with only around $2m-1$ constant multiplications by d^{-1}, $2m^2 - m - 1$ additions/subtractions and $2m^2 - 4m + 2$ bitwise rotations in $GF(p)$. Thus, the complexity of Algorithm 1 is effectively reduced to $2m$ multiplications, $2m - 1$ constant multiplications, $4m^2 - m - 1$ additions and $4m^2 - 8m + 4$ rotations in $GF(p)$.

On the other hand, similar improvements also exist for DFT modular multiplication. For instance, for DFT modular multiplication in $GF(p^m)$ where $p = 2^n - 1$, $m = n$ and $r = -2$, $f(x) = x^m - 2$ is always irreducible (see Theorem 7) and could be used as the field generating polynomial. In this case, the normalized irreducible polynomial would be $f_N(x) = -\frac{1}{2} \cdot x^m + 1$ and the following equality would hold in $GF(p)$:

$$F_{N_i} \;=\; -\frac{1}{2} \cdot (-2)^{mi} + 1 = \begin{cases} -\frac{1}{2} + 1 = \frac{1}{2}\,, & i \;\; even \\ \\ \frac{1}{2} + 1\,, & i \;\; odd \end{cases} \tag{5.6}$$

since

$$(-2)^{mi} \equiv (-2)^{ni} \equiv (-1)^{ni} \cdot (2^n)^i \equiv (-1)^{ni} \pmod{p}\,.$$

Note in (5.6) that F_{N_i} has only two distinct values, namely $\frac{1}{2}$ and $\frac{1}{2} + 1$. Hence, $F_{N_i} \cdot S$ in Step 11 of Algorithm 3 can attain only two values for any distinct value of S and these values can be precomputed outside the loop avoiding all such computations inside the loop. The precomputations can be achieved very efficiently with only one bitwise rotation and one addition. With the suggested improvement, both the number of base field additions/subtractions and the number of base field bitwise rotations required to perform an extension field multiplication are reduced by $(d-1) \cdot (m-1) = (2m-1) \cdot (m-1) = 2m^2 - 3m + 1$.

In Table 5.2, we present the new complexities for multiplication in $GF(p^m)$ when the single level FFT is used for Algorithm 1 and the above mentioned improvement is utilized for the DFT modular multiplication algorithm presented with Algorithm 3.

	Schoolbook	**Algorithm 1 (with FFT)**	**Algorithm 3 (improved)**
# Multiplications	m^2	$2m$	$2m$
# Constant Multiplications	–	$2m-1$	$m-1$
# Additions/Subtractions	m^2-m	$4m^2-m-1$	$4m^2-4m$
# Rotations	$m-1$	$4m^2-8m+4$	$2m^2-m-1$

Table 5.2: Complexity of multiplication in $GF(p^m)$ in terms of the number of $GF(p)$ operations when $f(x) = x^m - 2$, $p = 2^m - 1$ is a Mersenne prime and $d = 2m$

Clearly, the complexity of DFT modular multiplication (Algorithm 3) is an improvement upon the straightforward NTT based approach (Algorithm 1). Moreover, since it requires a significantly less number of complex operations such as multiplication and constant multiplication, its overall performance is better than the classical schoolbook method (given in (2.1) and (2.3)) especially for computationally constrained platforms where multiplication is significantly more expensive compared to simpler operations such as addition, subtraction or bitwise rotation.

Multiplication operation is inherently more complex than addition, subtraction or bitwise rotation and usually takes more clock cycles to run in hardware. In many modern microprocessors, in order to achieve higher clock rates, deeper pipelines are designed in the processor microarchitectures which results in significant differences in the number of clock cycles needed for different instructions. For instance, in the processor microarchitecture of Pentium 4 the latency is only half a clock cycle for a simple 16-bit integer addition, 1 clock cycle for a 32-bit integer addition and 14 clock cycles for a 32-bit integer multiplication [32]. As shown in Table 5.2, there is a tradeoff between Algorithm 3 and the schoolbook method in terms of the numbers of complex and simpler operations. Algorithm 3 requires computation of only a linear number of base field multiplications while in the classical schoolbook method a quadratic number of base field multiplications are performed. On the other hand, the number of simpler base field operations such as additions and rotations are significantly higher in Algorithm 3. Therefore, it may be more desirable to use Algorithm 3 in computational environments where multiplication is expensive compared with other operations such as addition and bitwise rotation.

In specialized hardware implementations, a multiplier circuit either runs much slower

than an adder or it is designed significantly larger in area to run as fast. In extremely constrained environments, such as wireless sensor network nodes which may be running using the constrained energy harnessed from the environment, the execution time may not be critical, but the available power may be tightly constrained. In this case, it may be desirable to have a simple low power/small area implementation of an n-bit multiplier which achieves an n-bit multiplication via n additions and n shift operations. Therefore, in a simple serialized hardware implementation the complexity of an n-bit multiplication may be assumed to be roughly equal to the complexity of n additions and n shift operations. Under these assumptions, Table 5.3 presents the complexities of modular multiplication in $GF(p^{13})$ for both Algorithm 3 and the classical schoolbook method when $p = 2^{13} - 1$ and $GF(p^{13})$ is constructed using the irreducible binomial $f(x) = x^{13} - 2$. The table also includes the total number of clock cycles for a single multiplication in $GF(p^m)$ with each method, assuming addition/subtraction and shift/rotation operations take a single clock cycle. Note that this finite field has size $\sim 2^{169}$ and is chosen to be representative for ECC. It is clear in Table 5.3 that Algorithm 3 would perform better in this scenario for the given parameters.

	Schoolbook	**Algorithm 3 (improved)**
#Multiplications	169	26
#Constant Multiplications	–	12
#Additions/Subtractions	156	624
#Rotations	12	324
#Total Clock Cycles	4562	1936

Table 5.3: Complexity of multiplication in $GF(p^{13})$ where $f(x) = x^{13} - 2$, $p = 2^{13} - 1$, $d = 26$ and $r = -2$

In order to see the crossover points between the performances of Algorithm 3 and the classical schoolbook method for different multiplication/addition latency ratios k and different field extension degrees m, in Figure 5.1 we graph the total number of clock cycles it takes to achieve multiplication with both methods assuming a base field addition/subtraction or bitwise-rotation operation takes only 1 clock cycle to complete and a base field multiplication operation takes k clock cycles. Here we assume the use of optimal parameters as mentioned earlier such as a Mersenne prime field characteristic p, $f(x) = x^m - 2$ and $d = 2m$. As we can see in Figure 5.1, for very small multiplication/addition latency ratios the schoolbook method performs better, however the crossover point is around $k = 4$ and for $k \geq 4$

Algorithm 3 performs clearly better.

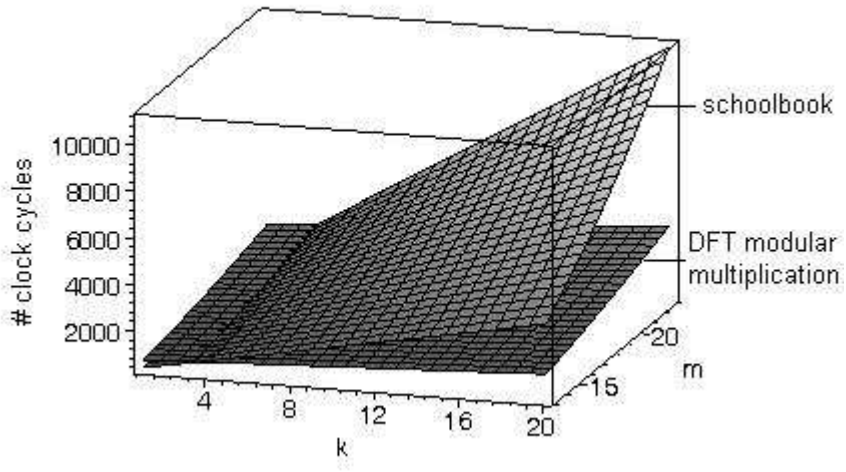

Figure 5.1: Number of required clock cycles for multiplication in $GF(p^m)$, where $p = 2^m - 1$, with Algorithm 3 and the classical schoolbook method assuming an addition or a bitwise-rotation in $GF(p)$ takes a single clock cycle and a $GF(p)$ multiplication takes k clock cycles

5.7 Conclusion

We introduced the DFT modular multiplication algorithm which performs modular multiplication in the frequency domain using Montgomery reduction. By allowing for modular reductions in the frequency domain, the costly overhead of back and forth conversions between the frequency and time domains is avoided, and thus efficient finite field multiplication is made possible for cryptographic operand sizes. We have shown that with the utilization of *DFT modular multiplication*, especially in computationally constrained platforms, finite field multiplication could be achieved more efficiently in the frequency domain than in the time domain for even small finite fields, e.g. ~ 160 bits in length, relevant to ECC.

Chapter 6

Implementing Elliptic Curve Cryptography in the Frequency Domain[1]

Acknowledgement: The material presented in this chapter is the result of a joint work with Sandeep Kumar and Prof. Christof Paar at the Communication Security Group, Ruhr-University Bochum, Germany.

6.1 Introduction

Elliptic curve cryptosystems [35, 55] are favorable choices for asymmetric data encryption compared to other popular algorithms such as RSA [73] mainly due to their requirement for smaller key sizes. According to the current best security estimates, the same level of security provided by a 1024-bit key in RSA can be achieved with only a 160-bit key in elliptic curve cryptography (ECC) [44, 45]. The key size determines the size of the operands over which finite field arithmetic operations are performed and consequently the efficiency of the cryptosystem [15, 54]. A comprehensive overview for hardware implementations of RSA and ECC are provided in [11].

Efficiency of an elliptic curve cryptosystem is highly dependent on the underlying finite field arithmetic. Multiplication in $GF(p^m)$ can be achieved with a quadratic number of multiplications and additions in the base field $GF(p)$ using the classical schoolbook method

[1] **The material presented in this chapter is included in [64].**

of polynomial multiplication. In the base field, the multiplication operation is inherently much more complex than other operations such as addition, therefore it is desirable that one performs as small a number of base field multiplications as possible for achieving an extension field multiplication. *Discrete Fourier transform (DFT) modular multiplication*, introduced in Chapter 5, achieves multiplication in $GF(p^m)$ in the frequency domain with only a linear number of base field $GF(p)$ multiplications in addition to a quadratic number of simpler base field operations such as additions/subtractions and bitwise rotations. In this chapter, we prove with our hardware implementation results that by using the DFT modular multiplication algorithm one can achieve efficient finite field multiplication in the frequency domain for small operands, e.g. as small as 160 bits in length, relevant to ECC.

In an ECC processor the multiplier unit usually consumes the most area on the chip, therefore it is crucial that one uses an area/time efficient multiplier, particularly in constrained environments, such as smart cards, wireless sensor network nodes or radio frequency identification tags, where resources are precious. In this work we address this issue by proposing an area/time efficient ECC processor architecture utilizing the DFT modular multiplication algorithm in a class of optimal extension fields (OEF) [8, 9] with the Mersenne prime field characteristic $p = 2^n - 1$ and the extension degree $m = n$. The proposed ECC processor architecture utilizes a hardware-optimized version of the DFT modular multiplication algorithm and requires an area ranging between 25k to 50k equivalent gates for implementations over OEFs of size 169, 289 and 361 bits.

In Section 6.2, we briefly review the DFT modular multiplication algorithm. In Section 6.3, we present an efficient elliptic curve cryptographic processor design which utilizes an optimized DFT modular multiplier architecture over $GF((2^{13}-1)^{13})$, $GF((2^{17}-1)^{17})$ and $GF((2^{19}-1)^{19})$ for an ASIC implementation using AMI Semiconductor $0.35\mu m$ CMOS technology. Finally, in Section 6.4 we present our implementation results.

6.2 DFT Modular Multiplication

Remember in Chapter 5 that for the frequency domain representations of the inputs $a(x) \cdot x^{m-1}$ and $b(x) \cdot x^{m-1}$, both in $GF(p^m)$, the DFT modular multiplication algorithm, presented with Algorithm 3, computes the frequency domain representation of $a(x) \cdot b(x) \cdot x^{m-1} \in$

$GF(p^m)$. Hence, it keeps the Montgomery residue representation intact and allows for consecutive multiplications in the frequency domain using the same algorithm.

n	**p = $2^n - 1$**	**m**	**d**	**r**	**equivalent binary field size**
13	8191	13	26	-2	$\sim 2^{169}$
17	131071	17	34	-2	$\sim 2^{289}$
19	524287	19	38	-2	$\sim 2^{361}$

Table 6.1: List of parameters suitable for optimized DFT modular multiplication

As shown in Table 5.2 of Section 5.6, when $r = -2$, $p = 2^n - 1$, the field generating polynomial is $f(x) = x^m - 2$, m is odd and $m = n$, the complexity of Algorithm 3 can be improved by simple precomputations. We present the optimized algorithm for this special case with Algorithm 4 and in Table 6.1 we suggest a list of parameters for implementation of the optimized algorithm over finite fields of different sizes. Note that the listed parameters are perfectly suited for ECC. In Section 6.4, we will provide hardware implementation results for all the finite fields listed in Table 6.1 and thus show the relevance of the optimized algorithm for area/time efficient hardware implementation of ECC in constrained environments.

6.3 Implementation of an ECC Processor Utilizing DFT Modular Multiplication

In this section we present a hardware implementation of an ECC processor which uses the DFT modular multiplication algorithm introduced with Algorithm 3 in Chapter 5 and optimized with Algorithm 4 in this chapter. The DFT modular multiplication algorithm trades off computationally expensive modular multiplication operations for simple bitwise rotations which can be achieved practically for free in hardware by proper rewiring. We exemplarily use the field $GF((2^{13} - 1)^{13})$ to explain our design, although the design is easily

Algorithm 4 Optimized DFT modular multiplication in $GF(p^m)$ for $r = -2$, $d = 2m$, $p = 2^n - 1$, m odd, $m = n$ and $f(x) = x^m - 2$

Input: $d = 2m$, $(A) \equiv a(x) \in GF(p^m)$, $(B) \equiv b(x) \in GF(p^m)$
Output: $(C) \equiv a(x) \cdot b(x) \cdot x^{-(m-1)} \in GF(p^m)$

1: **for** $i = 0$ to $d - 1$ **do**
2: $C_i \leftarrow A_i \cdot B_i$
3: **end for**
4: **for** $j = 0$ to $m - 2$ **do**
5: $S \leftarrow 0$
6: **for** $i = 0$ to $d - 1$ **do**
7: $S \leftarrow S + C_i$
8: **end for**
9: $S \leftarrow -S/d$
10: $S_{half} \leftarrow S/2$
11: $S_{even} \leftarrow S_{half}$
12: $S_{odd} \leftarrow S + S_{half}$
13: **for** $i = 0$ to $d - 1$ **do**
14: **if** $i \bmod 2 = 0$ **then**
15: $C_i \leftarrow C_i + S_{even}$
16: **else**
17: $C_i \leftarrow -(C_i + S_{odd})$
18: **end if**
19: $C_i \leftarrow C_i/2^i$
20: **end for**
21: **end for**
22: Return (C)

extendable for the other parameter sizes mentioned in Table 6.1 and the implementation results for all the parameter sizes are given in Section 6.4.

We first describe the implementation of the base field arithmetic in $GF(2^{13}-1)$ and then make parameter decisions based on Algorithm 4 to implement an efficient DFT modular multiplier. Then we present the overall processor design to compute the ECC scalar point multiplication operation.

6.3.1 Base Field Arithmetic

Base field arithmetic consists of addition, subtraction (addition with a negation) and multiplication in $GF(2^{13}-1)$. We design our arithmetic architectures here to ensure area/time efficiency.

Base Field Addition

Addition in the base field is implemented using a ripple carry adder. Reduction with the Mersenne prime $p = 2^{13}-1$ is just an extra addition of the carry generated. This additional addition is always hard wired, independent of the value of the carry, to avoid any timing related attacks. Figure 6.1 shows the design of the base field adder built using half adders and full adders.

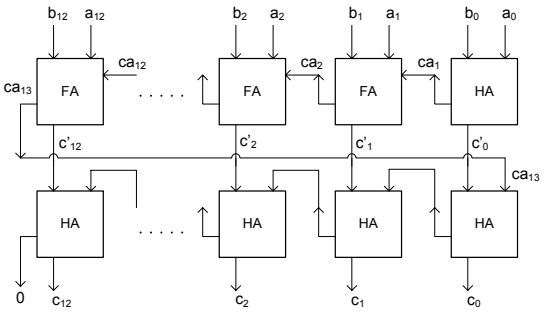

Figure 6.1: Base Field Addition Architecture

Base Field Negation

Negation in the base field is extremely simple to implement with a Mersenne prime p as the field characteristic. Negation of $B \in GF(p)$ is normally computed as $B' = p - B$. However, when p is a Mersenne prime, it is easy to see from the binary representation of $p = 2^{13} - 1$ (which is all 1's) that this subtraction is equivalent to flipping (NOT) of the bits of B. Hence, subtraction in this architecture can be implemented by using the adder architecture with an additional bitwise NOT operation on the subtrahend.

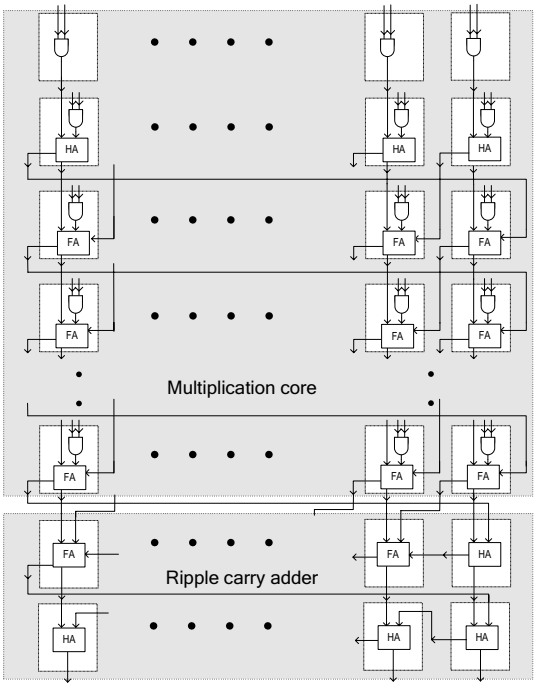

Figure 6.2: Base Field Multiplication with Interleaved Reduction

Base Field Multiplication

Base field multiplication is a 13×13-bit integer multiplication followed by a modular reduction with $p = 2^{13} - 1$. Since p is a Mersenne prime, an efficient way to implement this operation is to do an integer multiplication with interleaved reduction. Figure 6.2 shows the design of our base field multiplier architecture. It consists of the multiplication core, which performs integer multiplication with interleaved reduction of the carry, and a reduction unit

for the final reduction of the result which is performed using a ripple carry adder.

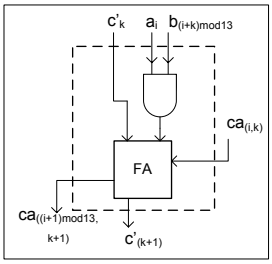

Figure 6.3: Processing Cell for the Base Field Multiplier Core

The processing cell of the multiplier core, which is shown in Figure 6.3, is built with a full adder. Here, a_i and b_i represent the inputs, ca_i represents the carry, and i and k are the column and row numbers, respectively, in Figure 6.2.

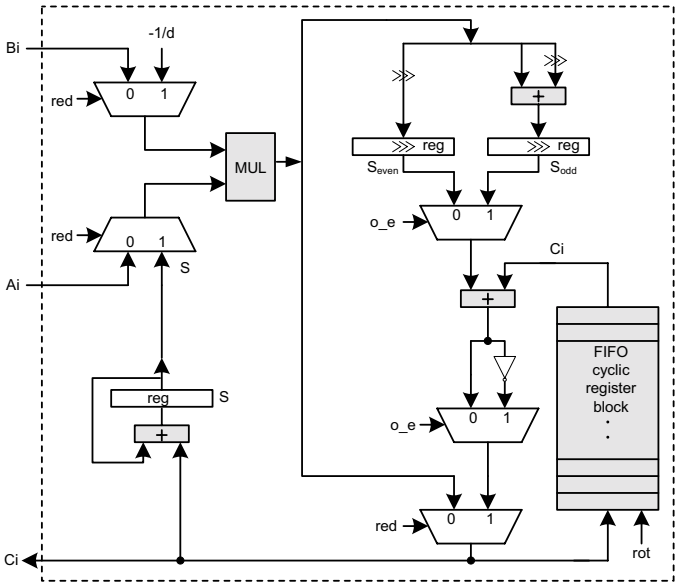

Figure 6.4: DFT Modular Multiplier Architecture

6.3.2 Polynomial Multiplier

Finite field multiplication of polynomials in an extension field, with coefficients in the base field, is computed using a polynomial multiplier. Using an extension field $GF(p^m)$, one can

reduce the area of a finite field multiplier in a very natural way, since in this case only a smaller base field multiplier is required. For instance, for performing multiplication in $GF((2^{13}-1)^{13})$, only a 13×13-bit base field multiplier is needed. However, an implementation in the time domain has the disadvantage of having a quadratic time complexity, because a total number of $13 \times 13 = 169$ base field multiplications need to be computed. In our design, we save most of these base field multiplications by utilizing *DFT modular multiplication* which requires performing only a linear number of 26 base field multiplications (see Steps $1-3$ in Algorithm 4).

Algorithm 5 Pseudo-code for hardware implementation of DFT modular multiplication

Input: $d = 2m$, $(A) \equiv a(x) \in GF(p^m)$, $(B) \equiv b(x) \in GF(p^m)$
Output: $(C) \equiv a(x) \cdot b(x) \cdot x^{-(m-1)} \in GF(p^m)$

1: $S \leftarrow 0$
2: **for** $i = 0$ to $d-1$ **do**
3: $\quad C_i \leftarrow A_i \cdot B_i$
4: $\quad S \leftarrow S + C_i$
5: **end for**
6: **for** $j = 0$ to $m-2$ **do**
7: $\quad S \leftarrow -S/d$
8: $\quad S_{even} \leftarrow S/2$
9: $\quad S_{odd} \leftarrow S + S/2$
10: $\quad S \leftarrow 0$
11: $\quad$ **for** $i = 0$ to $d-1$ **do**
12: $\quad\quad$ **if** $i \bmod 2 = 0$ **then**
13: $\quad\quad\quad C_i \leftarrow C_i + S_{even}$
14: $\quad\quad$ **else**
15: $\quad\quad\quad C_i \leftarrow -(C_i + S_{odd})$
16: $\quad\quad$ **end if**
17: $\quad\quad S_{even} \leftarrow S_{even}/2$
18: $\quad\quad S_{odd} \leftarrow S_{odd}/2$
19: $\quad\quad$ **for** $k = i+1$ to $d-1$ **do**
20: $\quad\quad\quad C_k \leftarrow C_k/2$
21: $\quad\quad$ **end for**
22: $\quad\quad S \leftarrow S + C_i$
23: $\quad$ **end for**
24: **end for**
25: Return (C)

DFT Modular Multiplication

For the application of DFT modular multiplication, Algorithm 4 is modified for an optimized hardware implementation. The main design decision here is to use a single base field multiplier to perform Step 2 (for pairwise coefficient multiplications) and Step 9 (for multiplications with the constant $-1/d$). Next, the two loops Steps $6-8$ (for accumulating C_i's) and Steps $13-20$ (for computing C_i's) were decided to be performed in parallel simultaneously. The final design that emerged is as shown in Figure 6.4 and the functionality is represented by the pseudo-code in Algorithm 5. During Steps $2-5$ (Algorithm 5) the multiplexer select signal *red* is set to 0 and later it is set to 1 for the remaining steps. This allows *MUL* (the base field multiplier) to be used for the initial multiplication, with the proper results accumulated in the register S. The registers S_{even} and S_{odd} cyclically rotate their contents every clock cycle for performing Steps 17 and 18 (Algorithm 5), respectively.

Step 19 in Algorithm 4, which involves different amounts of cyclic rotations for different C_i, would normally be inefficient to implement in hardware. In this work, this problem is solved in a *unique* way with the *FIFO (First-In First-Out) cyclic register block*. This temporary memory location for storing C_i values pushes in values till it is completely full. In the next loop, as the values are moved out of the *FIFO*, each of them is cyclically rotated at each clock cycle as they move up. Hence the different C_i values are cyclically rotated by different number of bits with no extra cost. The pseudo-code which shows the functionality of the memory block is given with Steps $19-21$ of Algorithm 5. Steps $14-18$ in Algorithm 4 are implemented using the two multiplexers with the select signal *o_e*.

Thus, based on an iterative study of different architectures we investigated, in Algorithm 5 we show the steps of an optimized version of the original DFT modular multiplication algorithm presented in Chapter 5 with Algorithm 3. Algorithm 5 is a hardware optimized and reordered version of Algorithm 3, which is fine tuned to generate a hardware efficient architecture. Due to its regular design, the proposed DFT modular multiplier architecture is easy to layout. The area is optimized by reusing the various components. Also, since all the bus signals are 13-bits wide, signal routing is made extremely easy in the design.

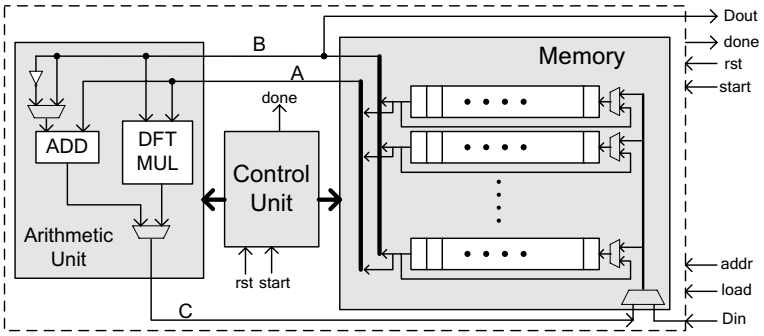

Figure 6.5: Top Level ECC Processor Architecture

6.3.3 Point Arithmetic

The overall architecture of the ECC processor is shown in Figure 6.5. The *Arithmetic Unit* consists of the *DFT modular multiplier* unit, and the base field adder unit which has a negation unit on one of its inputs for performing also the subtraction. All the necessary point variables are stored in the *Memory* component. We use FIFO registers here, because DFT modular multiplication and addition/subtraction operate only on 13 bits of the data at each clock cycle. This enables our processor to use 13-bit wide buses throughout the design, resulting in easy routing with reduced power consumption. To avoid losing the contents of the memory when being read out, they are looped back in the FIFO block, if new values are not being written in.

The *Control Unit* is the most important component which performs point arithmetic by writing the required variables onto the A and B busses, performing the required operations on them and storing the result back to the proper memory register. The *Control Unit* is also responsible for interacting with the external world by reading in inputs and writing out the results. The instruction set of the *Control Unit* is given in Table 6.2.

In our design, ECC point arithmetic is performed using mixed Jacobian-affine coordinates [54] and thus inversions are avoided. Here, we assume the utilized elliptic curve is of the form $y^2 = x^3 - 3x + b$. We use the binary NAF method (Algorithm 3.31 in [54]) with mixed coordinates to perform point multiplication. Point arithmetic is performed in such a way that the least amount of temporary storage is required. Since the point multiplication algorithm allows overwriting of the inputs while performing point doubling and addition, it requires only three extra temporary memory locations. The point doubling operation

is performed in Jacobian coordinates and requires 8 DFT modular multiplications and 12 sequence additions (or subtractions). Point addition is performed in mixed Jacobian-affine coordinates and requires 11 DFT modular multiplications and 7 additions. Point subtraction (for the binary NAF method) is easily implemented in exactly the same way as point addition with the exception of flipping the bits of y_2, the y-coordinate of the point to be subtracted. The *Memory* unit therefore consists of eight FIFO register blocks, one for each sequence, and has the total size of $26 \times 13 \times 8 = 2704$ bits.

Command	**Action**
LOAD [$addr$]	Load data into the register [$addr$]
READ [$addr$]	Read data from the register [$addr$]
DFT_MULT [$addr_A$] [$addr_B$] [$addr_C$]	Perform DFT modular multiplication on the sequences in [$addr_A$] and [$addr_B$], and store the result in [$addr_C$]
ADD_SEQ [$addr_A$] [$addr_B$] [$addr_C$]	Perform base field addition on the sequences in [$addr_A$] and [$addr_B$], and store the result in [$addr_C$]
SUB_SEQ [$addr_A$] [$addr_B$] [$addr_C$]	Perform base field subtraction on the sequences in [$addr_A$] and [$addr_B$], and store the result in [$addr_C$]
MOVE_SEQ [$addr_A$] [$addr_B$]	Move data from the register [$addr_A$] to the register [$addr_B$]
DFT_SEQ [$addr$]	Convert the data sequence in [$addr$] to the frequency domain
TIME_SEQ [$addr$]	Convert the data sequence in [$addr$] to the time domain

Table 6.2: Controller Commands of the ECC Processor

The inversion operation required for the final conversion from projective to affine coordinates is performed using Fermat's Little Theorem. The conversion from the time to the frequency domain, and vice-versa, is achieved by simple rotations which are performed using the *FIFO cyclic register block* inside the DFT modular multiplier unit.

Field	**Arithmetic Unit**	**Control Unit**	**Memory**	**Total Area**
$GF((2^{13}-1)^{13})$	5537.06	351.26	18768.66	24754.62
$GF((2^{17}-1)^{17})$	6978.95	362.56	31794.52	39243.00
$GF((2^{19}-1)^{19})$	10898.82	362.89	39586.72	50959.02

Table 6.3: Areas (in equivalent gate counts) for the presented ECC processor

6.4 Performance Analysis

In this section, we present the implementation results for our ECC processor design for three different finite fields: $GF((2^{13}-1)^{13})$, $GF((2^{17}-1)^{17})$ and $GF((2^{19}-1)^{19})$. For our performance measurements, we synthesized for a custom ASIC design using AMI Semiconductor $0.35\mu m$ CMOS technology using the Synopsys Design Compiler tools. Timing measurements were performed using the Modelsim simulator against test vectors generated with Maple. Table 6.4 presents the number of clock cycles required for DFT modular multiplication and ECC point arithmetic. It also shows the maximum clock frequency of the processor and the total time required to perform a point multiplication. Table 6.3 shows the area requirements for the ECC processor in terms of the equivalent number of NAND gates. The areas required for each of the three main components of the processor are also shown individually.

Although there are numerous ECC hardware implementations which are openly available in the literature, in Table 6.5 we attempt to compare our results only to VLSI implementations of ECC oriented towards similar application scenarios requiring small area with moderate speed. To the best of our knowledge, the only OEF implementation in hardware is presented by Lee et al. [43]. The authors present an FPGA implementation of ECC over $GF(p^m)$, where $p = 2^{31}-1$ and $m = 7$, which is comparable to our implementation over $GF((2^{17}-1)^{17})$. The best design mentioned here has a gate count of 228k and performs a

Field	DFT Multiplication	Point Double	Point Addition	Maximum Frequency (MHz)	Point Multiplication (avg. ms)
$GF((2^{13}-1)^{13})$	354	3180	4097	238.7	3.47
$GF((2^{17}-1)^{17})$	598	5228	6837	226.8	10.33
$GF((2^{19}-1)^{19})$	744	6444	8471	221.7	16.34

Table 6.4: Timing measurements (in clock cycles) for the presented ECC processor

Implementation	Field Size (equiv. binary)	Area (equiv. kgates)	Maximum Frequency (MHz)	Point Multiplication (avg. ms)
Lee et. al. [43]	$\sim 2^{217}$	228	26	11
Öztürk et. al. [61]	$\sim 2^{165}$	30	100	6.3
Satoh and Takano [74]	$\sim 2^{160}$	28	364	7.5
Ours	$\sim 2^{169}$	24.8	238.7	3.47
	$\sim 2^{289}$	39.2	226.8	10.33
	$\sim 2^{361}$	50.9	221.7	16.34

Table 6.5: Comparisons with other ECC processors for similar application scenarios

scalar point multiplication in 11 ms at a maximum possible clock frequency of 26 MHz. The huge area is due to the inversion unit that is used as an alternative to projective coordinates used in our design. This leads to our design being a factor of 5.8 smaller than the only known OEF hardware implementation but still being able to provide the same timing performance.

We would like to compare our results also to ECC implementations over other fields for similar applications. An implementation of ECC over the prime field $GF((2^{167}+1)/3)$, which has a comparable key length with our implementation over $GF((2^{13}-1)^{13})$, is presented in [61]. This design occupies an area of around 30k gates and achieves a point multiplication in 6.3 ms at 100 MHz clock frequency. Our design is 20% smaller than this design and still efficient in performance. Finally, we compare our design to the scalable dual-field based implementation presented by Satoh and Takano [74]. For the 160-bit field size, this

implementation has an area of 28k gates and achieves a point multiplication in 7.5 ms at the maximum clock frequency of 364 MHz. Our implementation over $GF((2^{13}-1)^{13})$, with the same field size, is more efficient in terms of both area and time. Based on these observations, we can easily confirm that the proposed implementation is area efficient without compromising on speed.

6.5 Conclusion

In this chapter, we presented the first hardware implementation of a frequency domain multiplier suitable for ECC and the first hardware implementation of ECC in the frequency domain. We proposed a novel area/time efficient ECC processor architecture which performs all finite field arithmetic operations in the frequency domain. The proposed architecture utilizes *the DFT modular multiplication algorithm* in a class of OEFs $GF(p^m)$ where the field characteristic is a Mersenne prime $p = 2^n - 1$ and $m = n$. The main advantage of our architecture is that it achieves extension field modular multiplication in the frequency domain with only a *linear* number of base field $GF(p)$ multiplications in addition to a quadratic number of simpler operations such as addition and bitwise rotation. We synthesized our architecture for custom VLSI CMOS technology to estimate the area and time performance, and showed that the proposed ECC processor is time/area efficient and would be useful in resource constrained applications.

Chapter 7

Inversion in the Frequency Domain[1]

7.1 Introduction

In Chapter 5, we introduced an efficient method, named *DFT modular multiplication*, for computing the Montgomery product of finite field elements in the frequency domain. With this method, we showed that multiplication in $GF(p^m)$ can be achieved with only a linear number of base field $GF(p)$ multiplications in addition to a quadratic number of simpler base field operations such as addition and fixed bitwise rotation for practical values of p and m relevant to *elliptic curve cryptography (ECC)*. In Chapter 6, we introduced an efficient, low-area implementation of an ECC processor architecture which utilizes the DFT modular multiplication algorithm and operates in the frequency domain. The introduced architecture performed all finite field arithmetic operations in the frequency domain, however it avoided inversions through the use of the *projective coordinates*. Even though the proposed architecture proved efficient for hardware implementations of ECC, the memory required for storing the projective point coordinates constituted a large amount of the circuit area. Projective coordinate representation requires three coordinate values to represent a point, while affine coordinate representation requires only two, and this may be a drawback for projective coordinate implementations of ECC in tightly constrained devices. Therefore, it is important to have a frequency domain inversion algorithm in order to realize ECC in affine coordinates potentially yielding lower storage requirement and power consumption. In this chapter, we introduce an adaptation of the Itoh-Tsujii inversion algorithm, described in Chapter 2, to the frequency domain for a class of OEFs $GF(p^m)$ where the field characteristic is a Mersenne

[1] **The material presented in this chapter is included in [70].**

prime $p = 2^n - 1$ or a Mersenne prime divisor $p = (2^n - 1)/t$ for a positive integer t and $m = n$. Our algorithm achieves an extension field $GF(p^m)$ inversion with only a single inversion, $O(m \log m)$ multiplications and constant multiplications, $O(m^2 \log m)$ additions and $O(m^2 \log m)$ fixed bitwise rotations in the base field $GF(p)$.

7.2 Itoh-Tsujii Inversion in the Frequency Domain

We propose a direct adaptation of the Itoh-Tsujii algorithm to the frequency domain for inversion in OEFs. As described in Section 2.2.1 of Chapter 2, Itoh-Tsujii inversion involves a chain of multiplications and Frobenius map computations in $GF(p^m)$ in addition to a single inversion in the base field $GF(p)$. For the required $GF(p^m)$ multiplications, we propose using DFT modular multiplication introduced in Chapter 5. Since Frobenius map computations can be achieved very easily in the time domain with simple pairwise multiplications, we propose performing the required Frobenius map computations in the time domain by applying the inverse NTT. Hence, back and forth conversions are needed between the frequency and time domains for the Frobenius map computations.

For efficient computation of the required DFT modular multiplications and Frobenius map computations in $GF(p^m)$, we propose using efficient parameters such as the irreducible field generating binomial $f(x) = x^m - 2$ for constructing $GF(p^m)$, the d^{th} primitive root of unity as $r = -2$, and the field characteristic as the Mersenne prime divisor $p = (2^n - 1)/t$ for a positive integer t, where n is odd and equals the field extension degree m. In this case, $r = -2 \in GF(p)$ is a primitive root of unity of order $2m$ and hence $d = 2m$. Theorem 7 in Chapter 5 proved that for $p = (2^n - 1)$ and $m = n$, $f(x) = x^m - 2$ is irreducible over $GF(p)$ for all practical values of p relevant to ECC. Furthermore, in Appendix a list of relevant binomials of the form $f(x) = x^m - 2$ are presented and shown to be irreducible over $GF(p)$ for many values of $p = (2^n - 1)/t$.

As noted in Chapters 4 and 5, when $r = -2$ and $p = (2^n - 1)/t$ for a positive integer t, a modular multiplication in $GF(p)$ with a power of r can be achieved very efficiently with a simple bitwise rotation in addition to a negation if the power is odd. It was also shown in Chapter 5 that for $r = -2$, m odd and $n = m$, the DFT modular multiplication algorithm can be optimized by precomputing some intermediary values in the algorithm. Remember that when the field generating polynomial is $f(x) = x^m - 2$, $r = -2$, $p = (2^n - 1)/t$ for a positive integer t, and m is odd and equal to n, i.e. when the bit length of the field

characteristic $p = 2^n - 1$ is equal to the field extension degree, the complexity of the DFT modular multiplication algorithm is only $2m$ multiplications, $m-1$ constant multiplications, $4m^2 - 4m$ additions/subtractions and $2m^2 - m - 1$ bitwise rotations in terms of $GF(p)$ operations, as presented in Table 7.2. A list of such efficient parameters suited for ECC is given in Table 7.1.

n	**p** $= (\mathbf{2^n - 1})/\mathbf{t}$	**m**	**d**	**r**	**equivalent binary field size**
13	8191/1	13	26	-2	$\sim 2^{169}$
17	131071/1	17	34	-2	$\sim 2^{289}$
19	524287/1	19	38	-2	$\sim 2^{361}$
23	8388607/47	23	46	-2	$\sim 2^{401}$

Table 7.1: Short list of efficient parameters for inversion in $GF(p^m)$ in the frequency domain

In Algorithm 6, we present the frequency domain Itoh-Tsujii algorithm exemplarily for the finite field $GF(p^m)$ with $p = 2^{13} - 1$ and $m = 13$. Note in Algorithm 6 that, for $A, B \in GF(p^m)$ and a positive integer i, *FrobeniusMap(A,i)* denotes the i^{th} Frobenius map of A and equals A^{p^i}, and *DFTmul(A,B)* denotes the DFT modular multiplication of A and B. A^{e-1} is computed in Steps $2-10$ of the algorithm with four DFT modular multiplications and five p^i-th power exponentiations in $GF(p^m)$, by using two temporary variables. However, there is a trade-off between the amount of temporary storage requirement and the required number of multiplications and Frobenius map computations. In the computation of A^{e-1}, one can always minimize the number of required temporary variables to one by using an alternating chain of p-th power exponentiations and multiplications with A, e.g., in Algorithm 6 A^{e-1} can be computed with the following chain of computations $T1 = A^{(10)_p}$, $T1 = A^{(11)_p}$, $T1 = A^{(110)_p}$, $T1 = A^{(111)_p}$, $T1 = A^{(1110)_p}$, $T1 = A^{(1111)_p}$, $T1 = A^{(11110)_p}$, $T1 = A^{(11111)_p}$, $T1 = A^{(111110)_p}$, $T1 = A^{(111111)_p}$, $T1 = A^{(1111110)_p}$, $T1 = A^{(1111111)_p}$, $T1 = A^{(11111110)_p}$, $T1 = A^{(11111111)_p}$, $T1 = A^{(111111110)_p}$, $T1 = A^{(111111111)_p}$, $T1 = A^{(1111111110)_p}$, $T1 = A^{(1111111111)_p}$, $T1 = A^{(11111111110)_p}$, $T1 = A^{(11111111111)_p}$, $T1 = A^{(111111111110)_p}$, $T1 = A^{(111111111111)_p}$ and $T1 = A^{(1111111111110)_p}$ by performing eleven multiplications with A and twelve p-th power exponentiations in $GF(p^m)$. We would like to note here that DFT modular multiplications in Algorithm 6 keep the Montgomery residue representation intact, but each Frobenius

Algorithm 6 Itoh-Tsujii inversion in $GF(p^m)$ in the frequency domain where $p = 2^n - 1$, $n = 13$ and $m = n$ (for $A, B \in GF(p^m)$ and a positive integer i, FrobeniusMap(A, i) denotes $A^{p^i} \in GF(p^m)$ and DFTmul(A, B) denotes the result of the DFT modular multiplication of A and B)

Input: $(A) \equiv a(x) \cdot x^{m-1} \in GF(p^m)$
Output: $(B) \equiv a(x)^{-1} \cdot x^{m-1} \in GF(p^m)$

// Compute $M \cdot a(x)^{e-1} \cdot x^{m-1} \in GF(p^m)$, where $e = p^{m-1} + p^{m-2} + p^{m-3} + \cdots + p + 1$
$T1 \leftarrow$ FrobeniusMap$(A, 1)$ // $A^{(10)_p}$
$T1 \leftarrow$ DFTmul$(T1, A)$ // $A^{(11)_p}$
$T2 \leftarrow$ FrobeniusMap$(T1, 2)$ // $A^{(1100)_p}$
$T1 \leftarrow$ DFTmul$(T1, T2)$ // $A^{(1111)_p}$
$T2 \leftarrow$ FrobeniusMap$(T1, 4)$ // $A^{(11110000)_p}$
$T1 \leftarrow$ DFTmul$(T1, T2)$ // $A^{(11111111)_p}$
$T2 \leftarrow$ FrobeniusMap$(T1, 4)$ // $A^{(111111110000)_p}$
$T1 \leftarrow$ DFTmul$(T2, T1)$ // $A^{(111111111111)_p}$
$T2 \leftarrow$ FrobeniusMap$(T1, 1)$ // $A^{(1111111111110)_p}$
// Compute $M \cdot a(x)^e \cdot x^{m-1} \in GF(p^m)$
$T1 \leftarrow$ DFTmul$(T2, A)$
// Compute $M^{-1} \cdot (a(x)^e)^{-1} \in GF(p)$
$A^{-e} \leftarrow T1_0^{-1}$
// Compute $a(x)^{-1} \cdot x^{m-1} \in GF(p^m)$
for $i = 0$ to $d - 1$ **do**
    $B_i \leftarrow A^{-e} \cdot T2_i$
end for
Return (B)

map computation adds an additional factor to the result. However, we will see in detail in in the remainder of this section that these additional factors cancel out within the algorithm.

Frobenius Map Computations:

We have seen in Section 2.2.1 of Chapter 2 that, when the field extension degree m is prime and the field generating polynomial $f(x)$ is a binomial, a Frobenius map computation in $GF(p^m)$ in the time domain is a simple fixed pairwise multiplication of the polynomial coefficients. Therefore, in our frequency domain Itoh-Tsujii algorithm we propose to perform the Frobenius map computations in the time domain. In order to do so, we will convert a frequency domain sequence to the time domain before computing its Frobenius endomorphism and come back to the frequency domain afterwards as shown in Algorithm 7. For $d = 2m$, since the time domain sequences have zeros as their higher ordered m elements, the NTT computations in Algorithm 7 can be simplified. Furthermore, since $d = 2m$ is composite, the performance of the NTT can be improved by utilizing the *fast Fourier transform (FFT)* [21] for a single level. We present the equivalent single level FFT computation for the inverse NTT operation with (7.1), and for the forward NTT computation with (7.2) and (7.3). Note that (7.2) and (7.3) are equivalent, except for the sign between the two summations.

$$a_i = \frac{2}{d} \cdot \sum_{j=0}^{m-1} A_{2j} r^{-2ij} \ , \ 0 \leq i \leq m-1 \tag{7.1}$$

$$A_j = \sum_{i=0}^{\frac{m-1}{2}} a_{2i} r^{2ij} + r^j \sum_{i=0}^{\frac{m-3}{2}} a_{2i+1} r^{2ij} \ , \ 0 \leq j \leq m-1 \tag{7.2}$$

$$A_{j+m} = \sum_{i=0}^{\frac{m-1}{2}} a_{2i} r^{2ij} - r^j \sum_{i=0}^{\frac{m-3}{2}} a_{2i+1} r^{2ij} \ , \ 0 \leq j \leq m-1 \tag{7.3}$$

As described with Theorem 2 and Corollary 1 in Chapter 2, when m is prime and $f(x) = x^m - w$ is irreducible over $GF(p)$, the equality $(x^j)^{p^i} = w^t x^j \pmod{f(x)}$, where $t = \frac{jp^i - j}{m}$, holds. Hence, the Frobenius coefficients do not need to be permuted. Furthermore, when $p = (2^n - 1)/t$, $m = n$ is prime and $f(x) = x^m - 2$, the following equality holds for the j^{th}

coefficient of the i^{th} iterate of the Frobenius map

$$w^t = 2^{\frac{jp^i - j}{m}} = 2^{\frac{j(p^i-1)}{m}} = 2^{j(p^{i-1}+p^{i-2}+\cdots+p+1)\frac{p-1}{m}} .$$

Due to the first condition of Theorem 1 (in Chapter 2), since $f(x) = x^m - 2$ is irreducible in $GF(p)$, $m|\text{ord}(2)$ and hence $m|(p-1)$. Thus, the above Frobenius map coefficients are all powers of 2 and multiplications by these coefficients can be achieved with $m-1$ simple bitwise rotations as shown in Step 5 of Algorithm 7. In Algorithm 7, FrobeniusMapCoefficient(i, j) equals $\frac{j(p^i-1)}{m}$ mod n and denotes the amount of bitwise left-rotations to be performed on the j^{th} coefficient of the time domain sequence to achieve the i^{th} iterate of the Frobenius map. With all the above mentioned optimizations utilized, the complexity of Algorithm 7 in terms of $GF(p)$ operations is m constant multiplications, $m^2 - 2m + 1$ fixed bitwise rotations and $m^2 - m$ additions for the inverse NTT computation, $m^2 - 2m + 1$ fixed bitwise rotations and m^2 additions/subtractions for the forward NTT computation and $m - 1$ fixed bitwise rotations for the Frobenius map computation in the time domain, totaling m constant multiplications, $2m^2 - 3m + 1$ fixed bitwise rotations and $2m^2 - m$ additions/subtractions, as given in Table 7.2.

Algorithm 7 Frobenius map computation in $GF(p^m)$ in the frequency domain when $p = (2^n - 1)/t$, and the irreducible field generating polynomial is $f(x) = x^m - 2$ (FrobeniusMapCoefficient$(i, j) = \frac{j(p^i-1)}{m}$ mod n)

Input: i, $(A) \equiv a(x) \cdot x^{m-1} \in GF(p^m)$
Output: $(B) \equiv (a(x) \cdot x^{m-1})^{p^i} \in GF(p^m)$

1: // Compute the time domain representation (a) of (A) using the inverse NTT
2: $(a) \leftarrow$ InverseNTT$((A))$
3: // Perform pairwise multiplications through simple bitwise rotations
4: **for** $j = 1$ to $m - 1$ **do**
5: $\quad a_j \leftarrow a_j <<$ FrobeniusMapCoefficient(i, j) // left rotate the bits of a_j
6: **end for**
7: // Compute the frequency domain representation (A) of (a) using the NTT
8: $(A) \leftarrow$ NTT$((a))$
9: Return $((A))$

Note that, in Algorithm 6, DFTmul(A, B) function which computes the DFT modular multiplication of (A) and (B) keeps the Montgomery representation with the multiplicative factor x^{m-1} intact, however FrobeniusMap(A, i) function which computes the i^{th} iterate of the Frobenius endomorphism on (A) adds an additional term to the multiplicative factor

x^{m-1}. Remember in Corollary 1 (Chapter 2) that when m is prime and $f(x) = x^m - 2$, the i^{th} iterate of the Frobenius endomorphism on x^{m-1} results in $(x^{m-1})^{p^i} = 2^t x^{m-1}$ where $t = \frac{(m-1)p^i-(m-1)}{m}$. Through the Frobenius map computations in Algorithm 6, the additional multiplicative factors 2^t accumulate to some value M until the computation of A^{e-1} in Step 10. Thus, in Step 12, the computed value $T1$ corresponds to some time domain value $M \cdot a(x)^e \cdot x^{m-1}$. Note that the i^{th} coefficient of the NTT of $M \cdot a(x)^e \cdot x^{m-1}$ is equal to $T1_i = M \cdot a(x)^e \cdot r^{i(m-1)}$ and thus $T1_0 = M \cdot a(x)^e$. Hence, $M \cdot a(x)^e \in GF(p)$ can be obtained by looking at the 0^{th} coefficient of $T1$. In Step 14, by taking the inverse of $T1_0$, $T1_0^{-1} = M^{-1} \cdot a(x)^{-e}$, rather than the desired value $a(x)^{-e}$, is obtained. However, the M^{-1} factor cancels out in the last step, i.e. in Step 17, when this false value of A^{-e} corresponding to $M^{-1} \cdot a(x)^{-e}$ is multiplied with the false value of A^{e-1} in $T2$ corresponding to $M \cdot a(x)^{e-1} \cdot x^{m-1}$ to give us the expected correct result which is the frequency domain representation of $a(x)^{-1} \cdot x^{m-1} \in GF(p^m)$.

Inversion in $GF(p)$:

We propose using Fermat inversion for performing the single inversion in $GF(p)$ required in Step 14 of Algorithm 2. For an n-bit prime p, this inversion can be conducted by taking the $(p-2)^{nd}$ power of the operand through a square-and-multiply chain with no more than $n-1$ multiply and $n-1$ square operations in $GF(p)$.

Complexity of Itoh-Tsujii Inversion in the Frequency Domain:

As described with Algorithm 6 for the exemplary finite field $GF(p^m)$ with $p = 2^{13} - 1$ and $m = 13$, Itoh-Tsujii algorithm achieves inversion utilizing a chain of multiplications and Frobenius map computations. Remember in Section 2.2.1 of Chapter 2 that in order to compute A^{e-1}, e.g. in Steps $1-10$ of Algorithm 6, one needs to perform $\Delta - 1$ multiplications and Δ Frobenius map computations in $GF(p^m)$, where $\Delta = \lfloor \log_2(m-1) \rfloor + HW(m-1)$. For inversion in the frequency domain, we propose to use the DFT modular multiplication algorithm (Algorithm 3) for the required multiplications and Algorithm 7 for the required Frobenius map computations. A^e can be computed with an additional DFT modular multiplication in the frequency domain, as in Step 12 of Algorithm 6. The single $GF(p)$ inversion

A^{-e}, as in Step 14 of Algorithm 6, can be computed using Fermat inversion with no more than $n-1$ multiplications and $n-1$ squarings in $GF(p)$ for $p=(2^n-1)/t$. Finally, A^{-1} is computed, as in Steps $16-18$ of Algorithm 6, with $2m$ multiplications in $GF(p)$. We have seen that when the field generating polynomial is $f(x)=x^m-2$, $p=(2^m-1)/t$, $d=2m$ and $r=-2$ is used as the d^{th} primitive root of unity, DFT modular multiplication (Algorithm 3) and Frobenius endomorphism (Algorithm 7) operations can be achieved extremely efficiently with the complexities given in Table 7.2. Thus, the total complexity of Itoh-Tsujii inversion in the frequency domain for such parameters is $2m\Delta+4m-2$ multiplications, $2m\Delta-\Delta$ constant multiplications, $6m^2\Delta-5m\Delta$ additions and $4m^2\Delta-4m\Delta$ bitwise rotations in $GF(p)$ as given in Table 7.2.

	$\#\mathcal{M}$	$\#\mathcal{CM}$	$\#\mathcal{A}/\mathcal{S}$	$\#\mathcal{R}$
Algorithm 3	$2m$	$m-1$	$4m^2-4m$	$2m^2-m-1$
ITI (frequency)	$2m\Delta+4m-2$	$2m\Delta-\Delta$	$6m^2\Delta-5m\Delta$	$4m^2\Delta-4m\Delta$
Algorithm 7	$-$	m	$2m^2-m$	$2m^2-3m+1$
ITI (time)	$m^2\Delta-m^2$ $+4m-2$	$-$	$m^2\Delta-m^2$ $-m\Delta+2m-1$	$2m\Delta-m$ $-2\Delta+2$

Table 7.2: Complexities of Algorithm 3, Algorithm 7, and time and frequency domain Itoh-Tsujii inversion (ITI) in $GF(p^m)$ in terms of the number of required $GF(p)$ multiplications, constant multiplications, additions/subtractions and rotations, when $f(x)=x^m-2$, $p=(2^n-1)/t$, $m=n$ is odd and $d=2m$, $(\Delta=\lfloor\log_2(m-1)\rfloor+HW(m-1))$

In the time domain the same Itoh-Tsujii inversion is achieved with slight differences. In this case, for multiplications in $GF(p^m)$ the classical schoolbook method, as described with (2.1) and (2.3), is used with the complexity of m^2 multiplications, m^2-m additions and $m-1$ bitwise rotations in $GF(p)$. Furthermore, Frobenius map computations in $GF(p^m)$ are achieved with the complexity of only $m-1$ bitwise rotations in $GF(p)$. Since $a(x)^e$ is known to be in $GF(p)$, the multiplication $a(x)^{e-1}\cdot a(x)$ can be achieved by finding only the first coefficient of the product with only m multiplications, $m-1$ additions and 1 bitwise rotation

in $GF(p)$. The single $GF(p)$ inversion, for computing $(a(x)^e)^{-1}$, can be achieved with $2(m-1)$ multiplications and the product $a(x)^{e-1} \cdot a(x)^{-e}$ can be computed with m multiplications in $GF(p)$. Thus, the total complexity of Itoh-Tsujii inversion in the time domain, for the same parameters, is $m^2\Delta - m^2 + 4m - 2$ multiplications, $m^2\Delta - m^2 - m\Delta + 2m - 1$ additions and $2m\Delta - m - 2\Delta + 2$ rotations in $GF(p)$. The complexity of Itoh-Tsujii inversion both in the frequency and time domains are presented in Table 7.2.

	Frequency Domain	**Time Domain**
#Multiplications	180	726
#Constant Multiplications	125	–
#Additions/Subtractions	4745	636
#Fixed Rotations	3120	109

Table 7.3: Complexities of Itoh-Tsujii inversion in $GF(p^{13})$ in the time and frequency domains in terms of the number of $GF(p)$ operations for $f(x) = x^{13} - 2$ and $p = 2^{13} - 1$

Multiplication operation is inherently more complex and usually takes more clock cycles to run in hardware. In many modern microprocessors, in order to achieve higher clock rates, deeper pipelines are designed in the processor microarchitectures which results in significant differences in the number of clock cycles needed for different instructions. For instance, in the processor microarchitecture of Pentium 4 the latency is only half a clock cycle for a simple 16-bit integer addition, 1 clock cycle for a 32-bit integer addition and 14 clock cycles for a 32-bit integer multiplication [32]. As shown in Table 7.3 for the exemplary finite field $GF((2^{13}-1)^{13})$, Itoh-Tsujii algorithm requires a dramatically less number of base field multiplications in the frequency domain than in the time domain. Therefore, it may be desirable to utilize frequency domain inversion in computational environments where multiplication is expensive compared with other operations such as addition and bitwise rotation.

In order to see the crossover points between the performances of time and frequency domain Itoh-Tsujii algorithms for different multiplication/addition latency ratios k and different field extension degrees m, in Figure 7.1 we present the total number of clock cycles it takes to achieve inversion with both methods assuming a base field addition/subtraction or

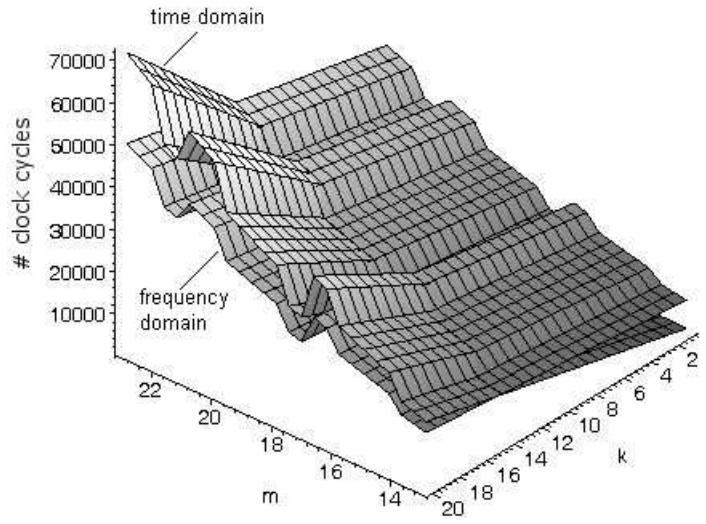

Figure 7.1: Number of required clock cycles for inversion in $GF(p^m)$ in the time and frequency domains, for $p = (2^n - 1)/t$, $m = n$ and $f(x) = x^m - 2$, assuming a $GF(p)$ addition or bitwise-rotation takes a single clock cycle while a $GF(p)$ multiplication takes k clock cycles

bitwise-rotation operation takes only 1 clock cycle to complete and a base field multiplication operation takes k clock cycles. As we can see in the graph, for small multiplication/addition latency ratios inversion in the time domain performs clearly better. For the field extension degree of $m = 13$, the crossover point is at around $k = 14$, and hence the latency ratio k should be at least 14 for Itoh-Tsujii inversion in $GF(p^{13})$ to perform better in the frequency domain. As the field extension degree m gets larger, frequency domain inversion starts performing better at smaller latency ratios.

7.3 Conclusion

In this chapter, the known first frequency domain finite field inversion algorithm is proposed for ECC. The proposed algorithm is an adaptation of the *Itoh-Tsujii algorithm* for OEFs to the frequency domain, and can achieve inversion in the extension field $GF(p^m)$ with only

a single inversion, $O(m \log m)$ multiplications and constant multiplications, $O(m^2 \log m)$ additions and $O(m^2 \log m)$ fixed bitwise rotations in the base $GF(p)$ for a class of OEFs where the field characteristic p is a prime number of the form $p = (2^n - 1)/t$, for a positive integer t, and $m = n$. With its low computational complexity, i.e. $O(m \log m)$ in terms of the required base field multiplications in addition to $O(m^2 \log m)$ base field additions and fixed bitwise rotations, the proposed algorithm would be well suited especially for efficient low-power constrained hardware implementation of ECC using the affine coordinates.

Chapter 8

Conclusion

8.1 Summary and Conclusions

We investigated the application of the *number theoretic transform (NTT)* based multiplication, which found many applications in digital signal processing for multiplication of very long sequences, e.g. with at least a few thousand elements, to finite fields with an emphasis on *elliptic curve cryptography (ECC)*, and presented practical parameters for its efficient implementation. We showed the relationship between the NTT and the *residue number system (RNS)* and proved that the frequency domain representation of a polynomial using the NTT is equivalent to the RNS representation of the same polynomial provided certain conditions are satisfied on the modulus polynomials defining the RNS. Hence we showed that multiplication in $GF(p^m)$ using the NTT is equivalent to an optimal case of multiplication in $GF(p^m)$ using the RNS in terms of the number of required $GF(p)$ multiplications.

Furthermore, we introduced the *DFT modular multiplication* algorithm which performs modular multiplication in the frequency domain using Montgomery reduction. By allowing for modular reductions in the frequency domain the costly overhead of back and forth conversions between the frequency and time domains is avoided, and thus more efficient finite field multiplication is made possible for cryptographic operand sizes. We showed that with the utilization of the NTT in general, and with the *DFT modular multiplication* method in particular, especially in computationally constrained platforms finite field multiplication can be achieved more efficiently in the frequency domain than in the time domain for even small finite fields, e.g. ~ 160 bits in length, relevant to ECC.

We proposed a novel area/time efficient ECC processor architecture which utilizes DFT

modular multiplication and performs all finite field arithmetic operations in the frequency domain using the projective coordinates. We implemented our architecture in hardware and synthesized it for custom VLSI CMOS technology to estimate the area and time performance. We showed that the proposed ECC processor is time/area efficient and useful in resource constrained environments such as wireless sensor networks. This work presented the first hardware implementation of a frequency domain multiplier suitable for ECC and the first hardware implementation of ECC in the frequency domain.

Without an efficient inversion algorithm, it is more preferable to implement an elliptic curve cryptosystem using the projective coordinates, even though this may require more storage and possibly result in degraded performance. We proposed an adaptation of the Itoh-Tsujii inversion algorithm to the frequency domain which will make affine coordinate implementation of ECC feasible in the frequency domain, and potentially result in less storage requirement and improved performance. To the best of our knowledge, this is the first time a frequency domain inversion algorithm is proposed for the implementation of ECC in the frequency domain using affine coordinates.

8.2 Directions for Future Research

1. We identify the investigation of hardware architectures for efficient implementation of the frequency domain inversion algorithm proposed in Chapter 7, and thus efficient hardware implementation of ECC in the frequency domain using affine coordinates, as a subject for further investigation. Research in this direction may result in ECC processor architectures which may require less storage and yield better efficiency by avoiding the use of projective coordinates.

2. A new normal form for elliptic curves, called Edwards curves, was introduced recently by Harold M. Edwards in [26]. Edwards curves are shown to simplify the elliptic curve group law significantly and hence improve the performance of an elliptic curve cryptosystem [13, 12]. We believe using Edwards curves will improve the performance of an elliptic curve cryptosystem operating in the frequency domain.

3. In this work, efficient finite field arithmetic operations using the NTT are investigated mainly for efficient hardware implementation of ECC using OEFs. Arithmetic algorithms similar to those introduced in this work may be investigated for normal bases,

binary fields, i.e. $GF(2^m)$, or prime fields as well which will be also valuable in enabling the utilization of frequency domain finite field arithmetic for other cyptographic algorithms such as RSA [73].

4. While widely used cryptographic algorithms are known to be strong against the computational resources of an attacker, the real threat to cryptographic systems come from the fault attacks [7, 16] which try to extract the cryptographic key by deliberately injecting faults into a cryptosystem or side-channel attacks which utilize the side-channel information such as timing [37], power dissipation [38, 19], thermal noise or electromagnetic emanation [5], where the attacker utilizes the flaws in the implementation of a cryptographic algorithm rather than the algorithm itself. We identify the investigation of tamper resistance characteristics and possible strengths and/or weaknesses of the algorithms and architectures presented in this dissertation, and of frequency domain finite field arithmetic in general, against side-channel cryptanalysis as a direction for further investigation.

5. We introduced frequency domain finite field arithmetic algorithms for ECC and presented a hardware implementation of ECC in the frequency domain. Efficient implementation of frequency domain finite field arithmetic for ECC in general purpose microprocessors is a direction for further study.

Appendix

Finite Field	**f(x)**		
$GF((2^{13}-1)^{11})$	$x^{11}+2^2x^3+1,$	$x^{11}+2^3x^2+1,$	$x^{11}+2^6x^5+1$
$GF((2^{13}-1)^{12})$	$x^{12}+2^5x^5+1,$	$x^{12}+2^8x^5+1,$	$x^{12}+2^5x^7+1$
$GF((2^{13}-1)^{13})$	$x^{12}\pm 2^{s_0},$	for $1 \le s_0 \le 12$	
$GF((2^{17}-1)^{9})$	$x^9+2^6x+1,$	$x^9+2^9x+1,$	$x^9+2^{11}x^2+1$
$GF((2^{17}-1)^{11})$	$x^{11}+2^8x^3+1,$	$x^{11}+2^{13}x^3+1,$	$x^{11}+2^5x^4+1$
$GF((2^{17}-1)^{12})$	$x^{12}+x+1,$	$x^{12}+x^{11}+1,$	$x^{12}+2^9x^5+1$
$GF((2^{17}-1)^{13})$	$x^{13}+2x+1,$	$x^{13}+2^2x+1,$	$x^{13}+2x^2+1$
$GF((2^{17}-1)^{14})$	$x^{14}+x^2+1,$	$x^{14}+x^6+1,$	$x^{14}+x^8+1$
$GF((2^{17}-1)^{15})$	$x^{15}+2^4x+1,$	$x^{15}+2^5x^2+1,$	$x^{15}+2^6x^4+1$
$GF((2^{17}-1)^{16})$	$x^{16}+2^5x^5+1,$	$x^{16}+2^5x^{11}+1,$	$x^{16}+2^{16}x^5+2^{16}$
$GF((2^{17}-1)^{17})$	$x^{17}\pm 2^{s_0},$	for $1 \le s_0 \le 16$	
$GF((2^{19}-1)^{10})$	$x^{10}\pm 2^{s_0},$	for $1 \le s_0 \le 18$	
$GF((2^{19}-1)^{11})$	$x^{11}+2^9x+1,$	$x^{11}+2^{13}x+1,$	$x^{11}+2^2x^2+1$
$GF((2^{19}-1)^{12})$	$x^{12}+2^{11}x+1,$	$x^{12}+2^{14}x+1,$	$x^{12}+2^{10}x^5+1$
$GF((2^{19}-1)^{13})$	$x^{13}+2^4x+1,$	$x^{13}+2^4x^2+1,$	$x^{13}+2^6x^2+1$
$GF((2^{19}-1)^{14})$	$x^{14}+2^3x+1,$	$x^{14}+2^{17}x+1,$	$x^{14}+2^3x^2+1$
$GF((2^{19}-1)^{15})$	$x^{15}+2^9x+2^0,$	$x^{15}+2^{18}x+1,$	$x^{15}+x^2+1$
$GF((2^{19}-1)^{16})$	$x^{16}+2^{18}x^3+1,$	$x^{16}+2^{15}x^7+1,$	$x^{16}+2^{15}x^9+1$
$GF((2^{19}-1)^{17})$	$x^{17}+x^3+1,$	$x^{17}+x^{14}+1,$	$x^{17}+2^5x+1$
$GF((2^{19}-1)^{18})$	$x^{18}+x^8+1,$	$x^{18}+x^{10}+1,$	$x^{18}+2^2x+1$
$GF((2^{19}-1)^{19})$	$x^{19}\pm 2^{s_0},$	for $1 \le s_0 \le 18$	
$GF((2^{31}-1)^{11})$	$x^{11}+2^{20}x+1,$	$x^{11}+2^{29}x+1,$	$x^{11}+2^5x^4+1$
$GF((2^{31}-1)^{12})$	$x^{12}+2^{21}x+1,$	$x^{12}+2^{23}x+1,$	$x^{12}+2^{30}x+1$
$GF((2^{31}-1)^{13})$	$x^{13}+2^8x+1,$	$x^{13}+2^{12}x+1,$	$x^{13}+2^{15}x+1$

Table 8.1: Short list of efficient irreducible polynomials for the construction of the finite fields listed in Table 4.1

Finite Field	**f(x)**		
$GF((\frac{2^{15}-1}{217})^{15})$	$x^{15}+2^2x^3+2^2$,	$x^{15}+2^3x^2+2^3$,	$x^{15}+2^4x^3+2^4$
$GF((\frac{2^{23}-1}{47})^{9})$	x^9+2^2x+1,	x^9+2^6x+1,	x^9+2^9x+1
$GF((\frac{2^{23}-1}{47})^{10})$	$x^{10}+2^8x+1$,	$x^{10}+2^{11}x^3+1$,	$x^{10}+2^{18}x^3+1$
$GF((\frac{2^{23}-1}{47})^{11})$	$x^{11}+x^5+1$,	$x^{11}+x^6+1$,	$x^{11}+2^4x+1$
$GF((\frac{2^{23}-1}{47})^{12})$	$x^{12}+2^2x+1$,	$x^{12}+2^3x^3+1$,	$x^{12}+2^7x^3+1$
$GF((\frac{2^{23}-1}{47})^{17})$	$x^{17}+2^{15}x+1$,	$x^{17}+2^{17}x^3+1$,	$x^{17}+2^5x^4+1$
$GF((\frac{2^{23}-1}{47})^{18})$	$x^{18}+2^9x+1$,	$x^{18}+2^{15}x+1$,	$x^{18}+2^7x^3+1$
$GF((\frac{2^{23}-1}{47})^{19})$	$x^{19}+2^{16}x^4+1$,	$x^{19}+2^2x^6+1$,	$x^{19}+2^{15}x^8+1$
$GF((\frac{2^{23}-1}{47})^{20})$	$x^{20}+2^{11}x^7+1$,	$x^{20}+2^{17}x^9+1$,	$x^{20}+2^{17}x^{11}+1$
$GF((\frac{2^{23}-1}{47})^{21})$	$x^{21}+2^9x+1$,	$x^{21}+2^4x^2+1$,	$x^{21}+2^6x^7+1$
$GF((\frac{2^{23}-1}{47})^{22})$	$x^{22}+2^{17}x^3+1$,	$x^{22}+2^{15}x^5+1$,	$x^{22}+2^5x^7+1$
$GF((\frac{2^{23}-1}{47})^{23})$	$x^{23}\pm 2^{s_0}$,	for $1 \le s_0 \le 22$	
$GF((\frac{2^{27}-1}{511})^{11})$	$x^{11}+x^3+1$,	$x^{11}+x^8+1$,	$x^{11}+2^{25}x+1$
$GF((\frac{2^{27}-1}{511})^{12})$	$x^{12}+2^4x+1$,	$x^{12}+2^{13}x+1$,	$x^{12}+2^{22}x+1$
$GF((\frac{2^{27}-1}{511})^{13})$	$x^{13}+2^5x^3+1$,	$x^{13}+2^7x^3+1$,	$x^{13}+2x^5+1$
$GF((\frac{2^{27}-1}{511})^{14})$	$x^{14}+2^{25}x+1$,	$x^{14}+2^{14}x^5+1$,	$x^{14}+2^{14}x^9+1$
$GF((\frac{2^{32}-1}{65535})^{13})$	$x^{13}+2^5x^3+1$,	$x^{13}+2^{11}x^3+1$,	$x^{13}+2^{15}x^5+1$
$GF((\frac{2^{32}-1}{65535})^{14})$	$x^{14}+2^2x^3+1$,	$x^{14}+2^4x^3+1$,	$x^{14}+2^{11}x^5+1$
$GF((\frac{2^{32}-1}{65535})^{15})$	$x^{15}+2^9x^4+1$,	$x^{15}+2^5x^7+1$,	$x^{15}+2^5x^8+1$
$GF((\frac{2^{32}-1}{65535})^{16})$	$x^{16}+2^{12}x^{11}+2^{12}$,	$x^{16}+x^3+2^3$,	$x^{16}+x^{11}+2^3$
$GF((\frac{2^{33}-1}{14329})^{17})$	$x^{17}+2^8x+1$,	$x^{17}+2^9x^2+1$,	$x^{17}+2^{15}x^2+1$
$GF((\frac{2^{37}-1}{223})^{11})$	$x^{11}+2^{19}x+1$,	$x^{11}+2^{31}x+1$,	$x^{11}+2^{32}x+1$
$GF((\frac{2^{37}-1}{223})^{12})$	$x^{12}+2^{17}x+1$,	$x^{12}+2^{19}x+1$,	$x^{12}+2^{25}x+1$
$GF((\frac{2^{37}-1}{223})^{13})$	$x^{13}+2^2x+1$,	$x^{13}+2^{24}x+1$,	$x^{13}+2^{28}x+1$
$GF((\frac{2^{37}-1}{223})^{14})$	$x^{14}+2^5x+1$,	$x^{14}+2^{21}x^3+1$,	$x^{14}+2^{30}x^3+1$
$GF((\frac{2^{37}-1}{223})^{15})$	$x^{15}+2^7x+1$,	$x^{15}+2^{22}x+1$,	$x^{15}+2^{27}x+1$
$GF((\frac{2^{37}-1}{223})^{16})$	$x^{16}+2^{22}x+1$,	$x^{16}+2^3x^3+1$,	$x^{16}+2^{19}x^3+1$
$GF((\frac{2^{37}-1}{223})^{17})$	$x^{17}+2^7x+1$,	$x^{17}+2^{11}x^2+1$,	$x^{17}+2^{12}x^2+1$
$GF((\frac{2^{37}-1}{223})^{18})$	$x^{18}+x+1$,	$x^{18}+x^{17}+1$,	$x^{18}+2^{15}x+1$
$GF((\frac{2^{37}-1}{223})^{19})$	$x^{19}+x^6+1$,	$x^{19}+x^{13}+1$,	$x^{19}+2x^3+1$

Table 8.2: Short list of efficient irreducible polynomials for the construction of the finite fields listed in Table 4.2

Finite Field	**f(x)**		
$GF((2^{2^3}+1)^{13})$	$x^{13}+2^8x+1,$	$x^{13}+2^{15}x+1,$	$x^{13}+2^3x^3+1$
$GF((2^{2^3}+1)^{14})$	$x^{14}+2^6x+1,$	$x^{14}+2^{14}x+1,$	$x^{14}+2^{15}x+1$
$GF((2^{2^3}+1)^{15})$	$x^{15}+2^{12}x+1,$	$x^{15}+2^{13}x+1,$	$x^{15}+2^4x^7+1$
$GF((2^{2^3}+1)^{16})$	$x^{16}+2^5x^7+2^5,$	$x^{16}+2^{11}x^9+2^{11},$	$x^{16}-2^5x^7+2^5$
$GF((2^{2^4}+1)^{7})$	$x^7+2^5x+1,$	$x^7+2^{15}x+1,$	$x^7+2^{16}x+1$
$GF((2^{2^4}+1)^{8})$	$x^8+x^3+1,$	$x^8+x^5+1,$	$x^8+2^4x^3+1$
$GF((2^{2^4}+1)^{13})$	$x^{13}+2^{29}x+1,$	$x^{13}+2^5x^3+1,$	$x^{13}+2^{11}x^3+1$
$GF((2^{2^4}+1)^{14})$	$x^{14}+2^2x^3+1,$	$x^{14}+2^4x^3+1,$	$x^{14}+2^{20}x^3+1$
$GF((2^{2^4}+1)^{15})$	$x^{15}+2^9x^4+1,$	$x^{15}+2^{29}x^2+1,$	$x^{15}+2^{30}x^7+1$
$GF((2^{2^4}+1)^{16})$	$x^{16}+2^{27}x^7+2^{27},$	$x^{16}+2^{29}x^5+2^{29},$	$x^{16}+2^{29}x^{13}+2^{29}$
$GF((2^{2^4}+1)^{29})$	$x^{29}+x^{11}+1,$	$x^{29}+x^{18}+1,$	$x^{29}+2^8x^2+1$
$GF((2^{2^4}+1)^{30})$	$x^{30}+2^7x^7+1,$	$x^{30}+2^{23}x^7+1,$	$x^{30}+2^{30}x^9+1$
$GF((2^{2^4}+1)^{31})$	$x^{31}+2^{30}x^3+1,$	$x^{31}+2^{31}x^3+1,$	$x^{31}+2^{23}x+1$
$GF((2^{2^4}+1)^{32})$	$x^{32}+2^{10}x+2^{10},$	$x^{32}+2^{25}x^5+2^{25},$	$x^{32}+2^{27}x^5+2^{27}$

Table 8.3: Short list of efficient irreducible polynomials for the construction of the finite fields listed in Table 4.3

Finite Field	**f(x)**		
$GF((\frac{2^{13}+1}{3})^{13})$	$x^{13} \pm 2^{s_0}$,	for $1 \leq s_0 \leq 12$	
$GF((\frac{2^{15}+1}{99})^{15})$	$x^{15} + 2$,	$x^{15} + 2^2$,	$x^{15} + 2^4$
$GF((\frac{2^{17}+1}{3})^{17})$	$x^{17} \pm 2^{s_0}$,	for $1 \leq s_0 \leq 16$	
$GF((\frac{2^{19}+1}{3})^{19})$	$x^{19} \pm 2^{s_0}$,	for $1 \leq s_0 \leq 18$	
$GF((\frac{2^{19}+1}{3})^{10})$	$x^{10} + 2^5x + 1$,	$x^{10} + 2^{24}x + 1$,	$x^{10} + 2x^2 + 1$
$GF((\frac{2^{20}+1}{17})^{20})$	$x^{20} + 2^2x + 2^2$,	$x^{20} + 2^{16}x + 2^{16}$,	$x^{20} + 2^2x^5 + 2^2$
$GF((\frac{2^{20}+1}{17})^{10})$	$x^{10} + 2^7x + 2^7$,	$x^{10} + 2^{15}x + 2^{15}$,	$x^{10} + 2x^3 + 2$
$GF((\frac{2^{21}+1}{387})^{21})$	$x^{21} + 2^3x + 2^3$,	$x^{21} + 2x^7 + 2$,	$x^{21} + 2^3x + 2^3$
$GF((\frac{2^{22}+1}{1985})^{22})$	$x^{22} + 2^6x + 2^6$,	$x^{22} + 2^{24}x + 2^{24}$,	$x^{22} + 2^{23}x^3 + 2^{23}$
$GF((\frac{2^{22}+1}{1985})^{11})$	$x^{11} \pm 2^{s_0}$,	for $1 \leq s_0 \leq 21$	
$GF((\frac{2^{23}+1}{3})^{23})$	$x^{23} \pm 2^{s_0}$,	for $1 \leq s_0 \leq 22$	
$GF((\frac{2^{27}+1}{1539})^{27})$	$x^{27} \pm 2^{s_0}$,	for $1 \leq s_0 \leq 8$	
$GF((\frac{2^{27}+1}{1539})^{9})$	$x^{9} \pm 2^{s_0}$,	for $\gcd(s_0, 3) = 1$	
$GF((\frac{2^{28}+1}{17})^{14})$	$x^{14} + 2^6x + 2^6$,	$x^{14} + 2^{11}x + 2^{11}$,	$x^{14} + 2^{16}x + 2^{16}$
$GF((\frac{2^{28}+1}{17})^{7})$	$x^{7} \pm 2^{s_0}$,	for $\gcd(s_0, 7) = 1$	
$GF((\frac{2^{32}+1}{641})^{16})$	$x^{16} + 2^2x + 2^2$,	$x^{16} + 2^{11}x + 2^{11}$,	$x^{16} + 2^{35}x + 2^{35}$
$GF((\frac{2^{32}+1}{641})^{8})$	$x^{8} + 2^5x + 2^5$,	$x^{8} + 2^6x + 2^6$,	$x^{8} + 2^{18}x + 2^{18}$

Table 8.4: Short list of efficient irreducible polynomials for the construction of the finite fields listed in Table 4.4

Bibliography

[1] R. Agarwal and J. Cooley. New Algorithms for Digital Convolution. *Acoustics, Speech, and Signal Processing, IEEE Transactions on*, 25(5):392–410, Oct 1977.

[2] R. C. Agarwal and C. S. Burrus. Fast Digital Convolution Using Fermat Transforms. In *Southwest IEEE Conf. Rec.*, pages 538–543, Houston, Texas, USA, April 1973.

[3] R. C. Agarwal and C. S. Burrus. Fast Convolutions Using Fermat Number Transforms with Applications to Digital Filtering. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-22(2):87–97, April 1974.

[4] R. C. Agarwal and C. S. Burrus. Number Theoretic Transforms to Implement Fast Digital Convolution. *Proceedings of the IEEE*, 63(4):550–560, April 1975.

[5] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM Side-Channel(s). In *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science (LNCS)*, pages 29–45. Springer, 2003.

[6] D. Agrawal, S. Baktır, D. Karakoyunlu, P. Rohatgi, and B. Sunar. Trojan detection using ic fingerprinting. *Security and Privacy, 2007. SP '07. IEEE Symposium on*, pages 296–310, 20-23 May 2007.

[7] R. Anderson and M. Kuhn. Tamper Resistance: A Cautionary Note. In *WOEC'96: Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 1–1, Berkeley, CA, USA, 1996. USENIX Association.

[8] D. V. Bailey and C. Paar. Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms. In H. Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, volume LNCS 1462, pages 472–485, Berlin, Germany, 1998. Springer-Verlag.

[9] D. V. Bailey and C. Paar. Efficient Arithmetic in Finite Field Extensions with Application in Elliptic Curve Cryptography. *Journal of Cryptology*, 14(3):153–176, 2001.

[10] A. Baraniecka and G. Jullien. Hardware Implementation of Convolution Using Number Theoretic Transforms. volume 4, pages 490–493, Apr 1979.

[11] L. Batina, S. B. Örs, B. Preneel, and J. Vandewalle. Hardware Architectures for Public Key Cryptography. *INTEGRATION, the VLSI journal*, 34(6):1–64, 2003.

[12] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted Edwards Curves. In *Progress in Cryptology - AFRICACRYPT 2008*, volume 5023 of *Lecture Notes in Computer Science (LNCS)*, pages 389–405. Springer, 2008.

[13] D. J. Bernstein and T. Lange. Faster Addition and Doubling on Elliptic Curves. In *Advances in Cryptology ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science (LNCS)*, pages 29–50. Springer, 2007.

[14] R. E. Blahut. *Fast Algorithms for Digital Signal Processing.* Addison-Wesley, Reading, Massachusetts, USA, 1985.

[15] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, London Mathematical Society Lecture Notes Series 265, 1999.

[16] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *Advances in Cryptology EUROCRYPT 97*, volume 1233 of *Lecture Notes in Computer Science (LNCS)*, pages 37–51. Springer, 1997.

[17] S. Boussakta, A. Y. M. Shakaff, F. Marir, and A. G. J. Holt. Number Theoretic Transforms of Periodic Structures and Their Applications. *IEE Proceedings on Circuits, Devices and Systems*, 135(2):83–96, Apr 1988.

[18] C. S. Burrus and T. W. Parks. *DFT/FFT and Convolution Algorithms.* John Wiley & Sons, 1985.

[19] S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science (LNCS)*, pages 51–62. Springer, 2003.

[20] P. Chevillat and F. Closs. Signal Processing with Number Theoretic Transforms and Limited Word Lengths. volume 3, pages 619–623, Apr 1978.

[21] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19:297–301, 1965.

[22] R. Crandall, K. Dilcher, and C. Pomerance. A Search for Wieferich and Wilson Primes. *Mathematics of Computation*, 66(217):433–449, 1997.

[23] R. Crandall and C. Pomerance. *Prime Numbers*. Springer-Verlag, New York, NY, USA, 2001.

[24] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

[25] V. S. Dimitrov, T. V. Cosklev, and B. Bonevsky. Number Theoretic Transforms over the Golden Section Quadratic Field. *IEEE Transactions on Signal Processing*, 43(8):1790–1797, Aug 1995.

[26] H. M. Edwards. A Normal Form for Elliptic Curves. *Bulletin (New Series) of the American Mathematical Society*, 44(3):393–422, July 2007.

[27] T. ElGamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.

[28] Martin Fürer. Faster integer multiplication. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 57–66, New York, NY, USA, 2007. ACM.

[29] J. Guajardo and C. Paar. Itoh-Tsujii Inversion in Standard Basis and Its Application in Cryptography. *Design, Codes, and Cryptography*, (25):207–216, 2002.

[30] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[31] M. Heideman, D. Johnson, and C. Burrus. Gauss and the History of the Fast Fourier Transform. *ASSP Magazine, IEEE*, 1(4):14–21, Oct 1984.

[32] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel. The Microarchitecture of the Pentium 4 Processor. *Intel Technology Journal*, Q1 2001.

[33] T. Itoh and S. Tsujii. A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases. *Information and Computation*, 78:171–177, 1988.

[34] A. Karatsuba and Y. Ofman. Multiplication of Multidigit Numbers on Automata. *Sov. Phys. Dokl. (English translation)*, 7(7):595–596, 1963.

[35] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[36] Ç. K. Koç and T. Acar. Montgomery Multplication in $GF(2^k)$. *Design, Codes, and Cryptography*, 14(1):57–69, 1998.

[37] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology CRYPTO 96*, volume 1109 of *Lecture Notes in Computer Science (LNCS)*, pages 104–113. Springer, 1996.

[38] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Advances in Cryptology CRYPTO 99*, volume 1666 of *Lecture Notes in Computer Science (LNCS)*, pages 388–397. Springer, 1999.

[39] H. Krishna, B. Krishna, K. Lin, and J. Sun. *Computational Number Theory and Digital Signal Processing: Fast Algorithms and Error Control Techniques.* Macmilian, 1967.

[40] T. Kriz and D. Bachman. A Number Theoretic Transform Approach to Image Rotation in Parallel Array Processors. volume 5, pages 430–433, Apr 1980.

[41] S. Kumar, M. Girimondo, A. Weimerskirch, C. Paar, A. Patel, and A. S. Wander. Embedded End-to-end Wireless Security with ECDH Key Exchange. *Circuits and Systems, 2003. MWSCAS '03. Proceedings of the 46th IEEE International Midwest Symposium on*, 2:786–789 Vol. 2, 27-30 Dec. 2003.

[42] B. Lawrence. Application of the Fast Fourier Number Theoretic Transform to Radar. *Radar Conference, 1991., Proceedings of the 1991 IEEE National*, pages 137–141, 12-13 Mar 1991.

[43] M-K. Lee, K. T. Kim, H. Kim, and D. K. Kim. Efficient Hardware Implementation of Elliptic Curve Cryptography over $GF(p^m)$. In *Proceedings of the 6th International Workshop on Information Security Applications (WISA 2005)*, volume 3786 of *Lecture Notes in Computer Science (LNCS)*, pages 207–217. Springer, 2006.

[44] A. K. Lenstra and E. R. Verheul. Selecting Cryptographic Key Sizes. In *Public Key Cryptography PKC 2000*, volume 1751 of *Lecture Notes in Computer Science (LNCS)*, pages 446–465. Springer, 2000.

[45] A. K. Lenstra and E. R. Verheul. Selecting Cryptographic Key Sizes. *Journal of Cryptology*, 14:255–293, 2001.

[46] W. Li. The Modified Fermat Number Transform and Its Application. pages 2365–2368 vol.3, 1-3 May 1990.

[47] R. Lidl and H. Niederreiter. *Finite Fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley, Reading, Massachusetts, USA, 1983.

[48] K. Y. Liu, I. S. Reed, and T. K. Truong. Fast Number-theoretic Transforms for Digital Filtering. *Electronics Letters*, 12(24):644–646, November 25 1976.

[49] G. Madre, E. H. Baghious, S. Azou, and G. Burel. Design of a Variable Rate Algorithm for CS-ACELP Coder. volume 1, pages 600–604 vol.1, 11-15 May 2003.

[50] G. Madre, E. H. Baghious, S. Azou, and G. Burel. Fast Pitch Modelling for CS-ACELP Coder Using Fermat Number Transforms. pages 765–768, 14-17 Dec. 2003.

[51] G. Madre, E. H. Baghious, S. Azou, and G. Burel. Linear Predictive Speech Coding Using Fermat Number Transform. volume 2, pages 607–612 vol.2, 2-5 July 2003.

[52] J. H. McClellan and C. M. Rader. *Number Theory in Digital Signal Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1979.

[53] R. J. McEliece. *Finite Fields for Computer Scientists and Engineers*. Kluwer Academic Publishers, 2nd edition, 1989.

[54] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, Florida, USA, 1997.

[55] V. Miller. Uses of Elliptic Curves in Cryptography. In H. C. WIlliams, editor, *Advances in Cryptology — CRYPTO '85*, volume LNCS 218, pages 417–426, Berlin, Germany, 1986. Springer-Verlag.

[56] P. L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, April 1985.

[57] Y. Morikawa and H. Hamada. Implementation for Two-dimensional FIR Filters Using the Number Theoretic Transform. volume 8, pages 1248–1251, Apr 1983.

[58] H. J. Nussbaumer. Digital Filtering Using Complex Mersenne Transforms. *IBM Journal of Research and Development*, 20(5), September 1976.

[59] H. J. Nussbaumer. Digital Filtering Using Pseudo Fermat Number Transforms. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-25:79–83, February 1977.

[60] A. V. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete-Time Signal Processing*. Prentice-Hall, Upper Saddle River, New Jersey, USA, 2nd edition, 1999.

[61] E. Öztürk, B. Sunar, and E. Savas. Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic. In *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)*, volume 3156 of *Lecture Notes in Computer Science (LNCS)*, pages 92–106. Springer, 2004.

[62] J. M. Pollard. The Fast Fourier Transform in a Finite Field. *Mathematics of Computation*, 25:365–374, 1971.

[63] S. Baktır. Efficient Algorithms for Finite Fields, with Applications in Elliptic Curve Cryptography. Master's thesis, ECE Department, Worcester Polytechnic Institute, Worcester, Massachusetts, USA, April 2003.

[64] S. Baktır, S. Kumar, C. Paar, and B. Sunar. A State-of-the-art Elliptic Curve Cryptographic Processor Operating in the Frequency Domain. *Mobile Networks and Applications (MONET)*, 12(4):259–270, September 2007.

[65] S. Baktır, J. Pelzl, T. Wollinger, B. Sunar, and C. Paar. Optimal tower fields for hyperelliptic curve cryptosystems. *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference on*, 1:522–526 Vol.1, 7-10 Nov. 2004.

[66] S. Baktır and B. Sunar. Frequency Domain Finite Field Arithmetic for Elliptic Curve Cryptography. *To be Published*.

[67] S. Baktır and B. Sunar. Optimal tower fields. *IEEE Transactions on Computers*, 53(10):1231–1243, 2004.

[68] S. Baktır and B. Sunar. Achieving Efficient Polynomial Multiplication in Fermat Fields Using the Fast Fourier Transform. In *ACM-SE 44: Proceedings of the 44th annual Southeast regional conference*, pages 549–554, New York, NY, USA, 2006. ACM.

[69] S. Baktır and B. Sunar. Finite Field Polynomial Multiplication in the Frequency Domain with Application to Elliptic Curve Cryptography. In *Computer and Information Sciences ISCIS 2006*, volume 4263 of *Lecture Notes in Computer Science (LNCS)*, pages 991–1001. Springer, 2006.

[70] S. Baktır and B. Sunar. Optimal Extension Field Inversion in the Frequency Domain. In *International Workshop on the Arithmetic of Finite Fields WAIFI 2008*, Lecture Notes in Computer Science (LNCS). Springer, 2008.

[71] C. M. Rader. Discrete Convolutions via Mersenne Transforms. *IEEE Transactions on Computers*, C-21(12):1269–1273, December 1972.

[72] C. M. Rader. The Number Theoretic DFT and Exact Discrete Convolution. In *IEEE Arden House Workshop on Digital Signal Processing*, Harriman, NewYork, January 1972.

[73] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[74] A. Satoh and K. Takano. A scalable dual-field elliptic curve cryptographic processor. *Computers, IEEE Transactions on*, 52(4):449–460, April 2003.

[75] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.

[76] A. Y. Md. Shakaff, A. Pajayakrit, and A. G. J. Holt. Practical Implementations of Block-mode Image Filters Using the Fermat Number Transform on a Microprocessor-based System. *IEE Proceedings Circuits, Devices and Systems*, 135(4):141–154, Aug 1988.

[77] A. Skavantzos and F. J. Taylor. On the Polynomial Residue Number System. *IEEE Transactions on Signal Processing*, 39(2):376–382, February 1991.

[78] F. J. Taylor. Residue Arithmetic A Tutorial with Examples. *IEEE Computer*, 17(5):50–62, May 1984.

[79] T. Toivonen and J. Heikkila. Video Filtering with Fermat Number Theoretic Transforms Using Residue Number System. *Circuits and Systems for Video Technology, IEEE Transactions on*, 16(1):92–101, Jan. 2006.

[80] R. Tolimieri, M. An, and C. Lu. *Algorithms for Discrete Fourier Transform and Convolution.* Springer-Verlag, 1989.

[81] P. K. S. Wah and T. Siegenthaler. Practical Transform Techniques for Error and Erasure Correction. *Electronics Letters*, 18(10):432–434, May 13 1982.

[82] S. Winograd. On Computing the Discrete Fourier Transform. *Proc. Nat. Acad. Sci. USA*, 73(4):1005–1006, April 1976.

[83] S. Winograd. Some Bilinear Forms whose Multiplicative Complexity Depends on the Field of Constants. *Mathematical Systems Theory*, 10:169–180, 1977.

[84] A. Woodbury, D. V. Bailey, and C. Paar. Elliptic Curve Cryptography on Smart Cards without Coprocessors. In *IFIP CARDIS 2000, Fourth Smart Card Research and Advanced Application Conference*, Bristol, UK, September 20–22 2000. Kluwer.

[85] S. Xu, L. Dai, and S. C. Lee. Autocorrelation Analysis of Speech Signals Using Fermat Number Transform (FNT). *IEEE Transactions on Signal Processing*, 40(8):1910–1914, Aug 1992.