

Finite Field Polynomial Multiplication in the Frequency Domain with Application to Elliptic Curve Cryptography

Selçuk Baktır, Berk Sunar

Worcester Polytechnic Institute
100 Institute Road, Worcester, MA 01609, USA
{selcuk, sunar}@wpi.edu

Abstract. We introduce an efficient method for computing Montgomery products of polynomials in the *frequency domain*. The discrete Fourier transform (DFT) based method originally proposed for integer multiplication provides an extremely efficient method with the best asymptotic complexity, i.e. $O(m \log m \log \log m)$, for multiplication of m -bit integers or $(m - 1)^{st}$ degree polynomials. However, the original DFT method bears significant overhead due to the conversions between the time and the frequency domains which makes it impractical for short operands as used in many applications. In this work, we introduce *DFT modular multiplication* which performs the entire modular multiplication (including the reduction step) in the frequency domain, and thus eliminates costly back and forth conversions. 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 elliptic curve cryptography (ECC). To the best of our knowledge, this is the first work that proposes the use of frequency domain arithmetic for ECC and shows that it can be efficient.

Key Words: Finite field multiplication, DFT, elliptic curve cryptography.

1 Introduction

Finite fields have many applications in coding theory [2, 1], cryptography [11, 4, 13], and digital signal processing [3]. Hence efficient implementation of finite field arithmetic operations is crucial. Multiplication of finite field elements is commonly implemented in terms of modular multiplication of polynomials realized in two steps: ordinary polynomial multiplication and modular reduction of the result by the field generating polynomial. Unlike modular reduction which has only linear complexity, i.e. $O(m)$, for polynomials of degree $m - 1$ when a fixed special modulus is chosen, polynomial multiplication with the classical *schoolbook method* has quadratic complexity, i.e. $O(m^2)$, given in terms of the number of coefficient multiplications and additions. This complexity may be improved to $O(m^{\log_2 3})$ using the Karatsuba algorithm [8]. However, despite the significant improvement gained by the Karatsuba algorithm, the complexity is

still not optimal. Furthermore, the implementation of the Karatsuba algorithm is more burdensome due to its recursive nature. The known fastest multiplication algorithm, introduced by Schönhage and Strassen [16], performs multiplication in the frequency domain using the fast Fourier transform (FFT) [6] with complexity $O(m \log m \log \log m)$ for multiplication of m -bit integers or m -coefficient polynomials [7]. Unfortunately, the FFT based algorithm becomes efficient and useful in practice only for very large operands, e.g. larger than 1000 bits in size, due to the overhead associated with the forward and inverse Fourier transform operations.

When the transformation computations between the time and frequency domains are not considered, frequency domain polynomial multiplication (without modular reduction) has surprisingly low *linear*, i.e. $O(m)$, complexity. Sadly, no efficient method for performing modular reduction is known to exist in the frequency domain, and therefore one needs to convert the result of a polynomial multiplication operation back to the time domain to perform modular reduction bearing significant overhead. In this paper we propose an algorithm for achieving modular reduction in the frequency domain. Thus, the entire finite field multiplication, including modular reduction, can be carried out in the frequency domain. In many finite field applications a chain of arithmetic operations is performed rather than a solitary one. Using our method, after an initial conversion step all intermediary operations may be computed in the frequency domain. Therefore, there will be no need for conversion before and after every single finite field multiplication except before the very first and after the very last ones.

2 Background

Number Theoretic Transform (NTT)

Number theoretic transform over a ring, also known as *the discrete Fourier transform over a finite field*, was introduced by Pollard [14]. For a finite field $GF(q)$ and a generic sequence (a) of length d whose entries are from $GF(q)$, the forward NTT of (a) over $GF(q)$, denoted by (A) , can be computed as

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

Likewise, the inverse NTT of (A) over $GF(q)$ can be computed as

$$a_i = \frac{1}{d} \cdot \sum_{j=0}^{d-1} A_j r^{-ij} \quad , \quad 0 \leq i \leq d-1 . \quad (2)$$

Here we refer to the elements of (a) and (A) by a_i and A_i , respectively, for $0 \leq i \leq d-1$. Also, we will refer to (a) and (A) as the *time and frequency domain* sequences, respectively. The above NTT computations over $GF(q)$ are defined by utilizing a d -th primitive root of unity, denoted by r , from $GF(q)$ or a

finite extension of it. Note that unlike the complex number $r = e^{j2\pi/d}$ generally used as the d -th primitive root of unity in the discrete Fourier transform (DFT) computations, a finite field element r can be utilized for the same purpose in an NTT. A number theoretic transform of special interest is *Mersenne transform*, which is an NTT with arithmetic modulo a Mersenne number $M_n = 2^n - 1$ [15]. Hence, if q is a Mersenne prime, then the NTT over $GF(q)$ is a Mersenne transform.

Mersenne transform allows for very efficient forward and inverse DFT operations for $r = \pm 2^k$, where k is an integer. Multiplication of an n -bit number with integer powers of 2^k 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 $(2^k)^i$ modulo M_n can be achieved with a simple bitwise left rotation by $ki \bmod n$ bits. Similarly, multiplication of an n -bit number with integer powers of -2^k 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^k is odd. Hence, when $r = \pm 2^k$ all of the multiplications by powers of r in the forward and inverse DFT computations in a Mersenne transform can be achieved with simple bitwise rotations. In this case, for a transform length of d , the forward DFT computation can be achieved with only $(d-1)^2$ simple rotations and $d(d-1)$ additions/subtractions avoiding any multiplications. For the inverse DFT computation additional d constant multiplications with $1/d$ are performed. For a more detailed complexity analysis of Mersenne transform, we refer the interested reader to [15].

Note that 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 $GF(q)$, where $q = M_n = 2^n - 1$ is a Mersenne prime, $r = \pm 2^k$ can take the values 2, -2 and 2^2 . Hence, the following equalities hold determining the relationship between d and r :

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

An extremely efficient method for computing the DFT is the fast Fourier transform (FFT) [6]. 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 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. 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 $O(d \log_2 d)$. However, we would like to note that for the case of Mersenne transform modulo a Mersenne prime $M_n = 2^n - 1$, and when $r = \pm 2^k$, the allowable sequence length d is either the prime number n (for $r = 2$ or $r = 2^2$) 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 computation which would have limited computational advantage.

Multiplication in $GF(q^m)$ Using NTT

Cyclic convolution of any two d -element sequences (a) and (b) in the time domain results in another d -element sequence (c) and can be computed as follows:

$$c_i = \sum_{j=0}^{d-1} a_j b_{i-j \bmod d}, \quad 0 \leq i \leq d-1. \quad (3)$$

The above convolution operation in the time domain is equivalent to the following computation in the frequency domain:

$$C_i = A_i \cdot B_i, \quad 0 \leq i \leq d-1. \quad (4)$$

Thus, convolution of two d -element sequences in the time domain, with complexity $O(d^2)$, has a surprisingly low $O(d)$ complexity in the frequency domain.

Multiplication of two polynomials is basically the same as the *acyclic (linear) convolution* of the polynomial coefficients. We have seen that cyclic convolution can be performed very efficiently in the frequency domain by pairwise coefficient multiplications, hence it will be wise to represent an element of $GF(q^m)$, in polynomial representation an $(m-1)^{st}$ degree polynomial with coefficients in $GF(q)$, with at least a $d = (2m-1)$ element sequence by appending zeros at the end, so that the cyclic convolution of two such sequences will be equivalent to their acyclic convolution and give us their polynomial multiplication. We can form sequences by taking the ordered coefficients of polynomials. For instance, $a(x) = a_0 + a_1x + a_2x^2 + \dots + a_{m-1}x^{m-1}$, an element of $GF(q^m)$ in polynomial representation, can be interpreted as the sequence $(a) = (a_0, a_1, a_2, \dots, a_{m-1}, 0, 0, \dots, 0)$ after appending $d-m$ zeros to the right. For $a(x), b(x) \in GF(q^m)$ the cyclic convolution of (a) and (b) yields a sequence (c) whose first $2m-1$ entries can be interpreted as the coefficients of $c(x) = a(x) \cdot b(x)$. The following straightforward algorithm (Algorithm 1) realizes the polynomial multiplication $c(x) = a(x) \cdot b(x)$.

Algorithm 1 Polynomial Multiplication by Direct Application of DFT

Input: $a(x), b(x) \in GF(q^m)$

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 (1)
 - 3: Multiply (A) with (B) to compute (C) as in (4)
 - 4: Convert (C) to (c) using the inverse NTT as in (2)
 - 5: Interpret the first $2m-1$ coefficients of (c) as the coefficients of $c(x) = a(x) \cdot b(x)$
 - 6: Return $c(x)$
-

Note that with Algorithm 1 the polynomial product $c(x) = a(x) \cdot b(x)$ is computed in the frequency domain but the final reduction by the field generating polynomial remains to be computed. One needs to convert (C) back to the time domain to do the modular reduction, so that further multiplications can be performed on it using the same method. In the next section, we introduce *DFT modular multiplication* which performs both polynomial multiplication and modular reduction in the frequency domain and thus avoids conversions between the time and frequency domains.

3 Modular Multiplication in the Frequency Domain

In many finite field applications, a chain of arithmetic operations needs to be performed. For example, in elliptic curve cryptography a scalar point product is computed by applying a chain of finite field additions, subtractions, multiplications, squarings and inversions on the input point coordinates [5]. The Montgomery residue representation has proven to be useful in this computation [12], [10], [9]. 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 is 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 the frequency domain. We find an arithmetic operation in the frequency domain that is equivalent to Montgomery multiplication in the time domain. In the remainder of this section, we introduce the *DFT modular multiplication* algorithm which allows for both polynomial multiplication and modular reduction operations in the frequency domain.

3.1 The DFT Modular Multiplication Algorithm

We use (f) and (x) to represent the d -element sequences for the irreducible field generating polynomial $f(x)$ and the constant polynomial $x \in GF(q^m)$, respectively, in the time domain. We denote the NTTs of (f) and (x) with (F) and (X) , respectively, and let F_i and X_i , for $0 \leq i \leq d-1$, denote their elements. For instance, the constant polynomial x is represented in the time domain with the d -element sequence $(x) = (0, 1, 0, 0, \dots, 0)$ and its frequency domain representation is $(X) = (1, r, r^2, r^3, r^4, r^5, \dots, r^{d-1})$.

DFT modular multiplication presented with Algorithm 2 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

$f'(x) = f(x)/f_0 \bmod q$ is used. Hence, $f'(x)$ is equivalent to $f(x)$ but normalized to have $f'(0) = 1$. We let (f') and (F') denote the d -element time and frequency domain sequence representations for $f'(x)$, where $f'_i = f_i/f_0 \bmod q$ and $F'_i = F_i/f_0 \bmod q$, respectively, for $0 \leq i \leq d-1$.

Algorithm 2 DFT Modular Multiplication

Input: $(A), (B)$ which represent $a(x), b(x) \in GF(q^m)$ in the frequency domain

Output: (C) which represents $c(x) = a(x) \cdot b(x) \cdot x^{-(m-1)} \bmod f(x)$ in the frequency domain

```

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:  for  $i = 0$  to  $d - 1$  do
11:     $C_i \leftarrow (C_i + F'_i \cdot S) \cdot X_i^{-1}$ 
12:  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 $(c(x) + S \cdot f'(x))$ 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 $S \cdot (F')$ to (C) in the frequency domain. Then again in the frequency domain, $(c(x) + S \cdot f'(x))$ 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 $(c(x) + S \cdot f'(x))$ by x is accomplished in the frequency domain by dividing $(C) + (F') \cdot S$ by (X) (Step 11). By repeating steps 5 – 12 of the algorithm $m - 1$ times the final result (C) , which represents $a(x) \cdot b(x) \cdot x^{-(m-1)} \bmod f(x)$ in the frequency domain, is obtained. \square

For $d \approx 2m$, modular multiplication in $GF(q^m)$ with the DFT modular multiplication algorithm requires $4m^2 - m - 1$ multiplications and $4m^2 - 4m$ additions in the ground field $GF(q)$, while the schoolbook method requires only m^2 multiplications and $(m-1)^2$ additions ignoring the cost of modular reduction.

Nevertheless, this complexity may be improved dramatically by using special values for q , r , d and the irreducible field generating polynomial $f(x)$ as we will see in the next section.

3.2 Utilizing Efficient Parameters for 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 2, using a Mersenne prime as the modulus and selection of $r = \pm 2^k$, where k is an integer, will allow for extremely efficient modular multiplication with $r^i = (\pm 2^k)^i$. This modular multiplication can be achieved with a simple bitwise rotation (in addition to a negation when $r = -2^k$ and i is odd) which is inexpensive. In the DFT modular multiplication algorithm, this property may be exploited if q is chosen as a Mersenne prime and r is chosen as $r = \pm 2^k$. In that case, in step 11 of the algorithm multiplications with $X_i^{-1} = r^{-i} = (\pm 2^k)^{-i} = (\pm 1)^i \cdot 2^{-ki \bmod d}$ become simple $(-ki \bmod d)$ -bit left-rotations (in addition to a negation when $r = -2^k$ and i is odd), which have negligible cost compared to a regular multiplication. Also, note that when q is a Mersenne prime, negation of an element of $GF(q)$ can be achieved by simply flipping its bits. Multiplications with F'_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, $F'_i = \pm r^{mi - s_0 \bmod d} + 1$ and hence for $r = \pm 2^k$ multiplications with F'_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, $F'_i = \pm r^{mi - s_0} + r^{m'i + s'_m - s_0} + 1$ or $F'_i = \pm r^{mi - s_0} - r^{m'i + s'_m - s_0} + 1$, respectively, and hence multiplications with F'_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 q , d , r and $f(x)$ are dependent on each other and can not be chosen independently.

3.3 Existence of Efficient Parameters

In Table 1 we give a list of parameters that would yield efficient multiplication in $GF(q^m)$ using the DFT modular multiplication algorithm for operand sizes relevant to elliptic curve cryptography. For every case listed in Table 1, 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.

4 Complexity of DFT Modular Multiplication

In this section, we present the complexity of DFT modular multiplication for a practical set of parameters relevant to ECC and compare it with two other

$q = M_n = 2^n - 1$	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	2^2 or 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	2^2 or 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	2^2 or 2	$\sim 2^{341}$
$2^{31} - 1$	12	31	2^2 or 2	$\sim 2^{372}$
$2^{31} - 1$	13	31	2^2 or 2	$\sim 2^{403}$

Table 1. List of some $q = M_n$, m , d and $r = \pm 2^k$ values for efficient DFT modular multiplication in $GF(q^m)$ for ECC over finite fields of size 143 to 403 bits.

efficient methods, namely the direct application of DFT multiplication with time domain modular reduction (Algorithm 1) and the Karatsuba algorithm [8]. In our complexity analysis we assume the use of Mersenne prime finite field characteristics as q , irreducible field generating binomials of the form $f(x) = x^m \pm 2^{s_0}$, d -th primitive root of unity $r = \pm 2^k$ and sequence length $d \approx 2m$. The field parameters we use, such as low Hamming weight field generating polynomial and Mersenne prime field characteristics, lead to efficient implementation of multiplication for all three methods. Therefore, for the selected parameters, we can safely assume that our comparisons are fair. In Table 2, we present the complexities of multiplication in $GF(q^m)$ in terms of $GF(q)$ operations for all three approaches when such ideal parameters are used. Note that the astonishingly low $O(m)$ complexity of Algorithms 1 and 2 in terms of the number of $GF(q)$ multiplications is achieved under the ideal conditions with these efficient field parameters together with the choice of $r = \pm 2^k$. The complexity presented for the Karatsuba algorithm is only an approximate one for the case when m is a power of two. For the best complexities of the Karatsuba algorithm for different extension degrees, the reader is referred to [17].

Clearly, the complexity of DFT modular multiplication (Algorithm 2) is an improvement upon the direct DFT approach (Algorithm 1). Moreover, since DFT modular multiplication requires significantly less number of complex operations such as multiplication and constant multiplication, its overall performance appears to be better than the Karatsuba algorithm, for computationally constrained platforms where multiplication is significantly more expensive compared to simpler operations such as addition, subtraction or bitwise rotation.

	Algorithm 1	Karatsuba	Algorithm 2
#Multiplication	$\approx 2m$	$\approx m^{\log_2 3}$	$\approx 2m$
#Const. Mult.	$\approx 2m$	–	$m - 1$
#Add./Subtr.	$\approx 8m^2 - 4m$	$\approx 6m^{\log_2 3} - 7m + 1$	$\approx 6m^2 - 6m$
#Rotation	$\approx 8m^2 - 8m + 2$	$m - 1$	$\approx 4m^2 - 4m$

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

Multiplication operation is inherently more complex than addition, subtraction or bitwise rotation. Therefore a multiplier circuit either runs much slower or it is designed significantly larger in area to run as fast. A straightforward low power/small area implementation of an n -bit multiplication can be achieved via n additions and n shift operations. Therefore, in 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. Using the same approach, we may assume that multiplication by a constant n -bit number can be achieved with $n/2$ shifts and $n/2$ additions on average. Under these assumptions, Table 3 gives the complexities of modular multiplication in $GF(q^{13})$ for all three methods when $q = 2^{13} - 1$ and $GF(q^{13})$ is constructed using the irreducible binomial $f(x) = x^{13} - 2$. The table also includes the total number of clock cycles that a single multiplication with these methods takes, assuming addition/subtraction and rotation operations all take a single clock cycle. Note that this finite field has size $\sim 2^{169}$ and is chosen to be representative for elliptic curve cryptography. We would like to note that all values in Table 3 are obtained directly from Table 2, except for the exact complexity of Karatsuba multiplication for $GF(q^{13})$ which we draw from [17].

Finally, we would like to note that for composite values of d , the FFT algorithm would apply to both Algorithm 1 and Algorithm 2 similarly for efficient computations. However, since in our case d does not take highly composite values (see Table 1), the speed up gained by possible application of FFT would be limited. Hence, for the generality and simplicity of our analysis, we did not consider the effect of using FFT in our complexity derivations.

	Algorithm 1	Karatsuba	Algorithm 2
#Multiplication	26	91	26
#Const. Mult.	26	–	12
#Add./Subtr.	1300	390	936
#Rotation	1250	12	624
#Total Clock Cycles	3564	2768	2392

Table 3. Complexity of multiplication in $GF(q^{13})$ where $f(x) = x^{13} - 2$, $q = 2^{13} - 1$, $d = 26$ and $r = -2$.

5 Future Work

The DFT modular multiplication algorithm presented in this paper may efficiently apply to binary fields, i.e. $GF(2^m)$, or prime fields, i.e. $GF(p)$, as well by using similar efficient parameters and methods. We identify investigation of such efficient parameters and methods, as well as architectures for their efficient implementations, as future research directions.

6 Conclusion

We introduced the DFT modular multiplication algorithm which performs modular multiplication in the frequency domain using Montgomery reduction. The DFT based multiplication technique, which performs multiplication in the frequency domain, is known to be very efficient. Nevertheless, due to the lack of a frequency domain reduction algorithm, for performing modular reductions conversion to the time domain is needed, which adds a significant overhead and makes the DFT based multiplication algorithms impractical for small operands, e.g. less than 1000 bits in length. In this work, by allowing for modular reductions in the frequency domain the overhead of back and forth conversions between the frequency and the time domains is avoided, and thus efficient finite field multiplication is made possible for cryptographic operand sizes. We have shown that with our method, 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 field sizes, e.g. ~ 160 bits, relevant to elliptic curve cryptography.

Acknowledgements

This work was supported by NSF CAREER award ANI-0133297.

References

1. E. R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, New York, New York, USA, 1968.

2. R. E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley, Reading, Massachusetts, USA, 1983.
3. R. E. Blahut. *Fast Algorithms for Digital Signal Processing*. Addison-Wesley, Reading, Massachusetts, USA, 1985.
4. I.F. Blake, X.H. Gao, R.C. Mullin, S.A. Vanstone, and T. Yaghoobin. *Applications of Finite Fields*. Kluwer Academic, 1999.
5. I.F. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, London Mathematical Society Lecture Notes Series 265, 1999.
6. J. Cooley and J. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19:297–301, 1965.
7. R. Crandall and C. Pomerance. *Prime Numbers*. Springer-Verlag, New York, NY, USA, 2001.
8. A. Karatsuba and Y. Ofman. Multiplication of Multidigit Numbers on Automata. *Sov. Phys. Dokl. (English translation)*, 7(7):595–596, 1963.
9. Ç. K Koç and T. Acar. Montgomery Multiplication in $GF(2^k)$. *Design, Codes, and Cryptography*, 14(1):57–69, 1998.
10. Ç. K Koç, T. Acar, and B. Kaliski. Analyzing and Comparing Montgomery Multiplication Algorithms. *IEEE Micro*, pages 26–33, June 1996.
11. R. J. McEliece. *Finite Fields for Computer Scientists and Engineers*. Kluwer Academic Publishers, 2nd edition, 1989.
12. P. L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, April 1985.
13. C. Paar. *Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields*. PhD thesis, (Engl. transl.), Institute for Experimental Mathematics, University of Essen, Essen, Germany, June 1994. ISBN 3–18–332810–0.
14. J. M. Pollard. The Fast Fourier Transform in a Finite Field. *Mathematics of Computation*, 25:365–374, 1971.
15. C. M. Rader. Discrete Convolutions via Mersenne Transforms. *IEEE Transactions on Computers*, C-21(12):1269–1273, December 1972.
16. A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.
17. A. Weimerskirch and C. Paar. Generalizations of the Karatsuba Algorithm for Efficient Implementations. Technical report, Department of Electrical Engineering and Information Sciences, Ruhr-Universität Bochum, Germany.